**Universidad Autónoma de la Ciudad de México**

# Notas de Métodos Numéricos

*Apuntes y ejercicios seleccionados*

Carlos E Martínez-Rodríguez
Academia de Matemáticas
Plantel Casa Libertad
carlos.martinez@uacm.edu.mx

---

2025

# Índice general

# Capítulo 1

# Introducción

## 1.1. Sobre el curso

El curso se llevará a cabo tres veces a la semana, con sesiones de 1.5 horas, de las cuales una de ellas se realizará en el laboratorio de cómputo 3.

## 1.2. Requisitos para acreditar la materia

El curso por su naturaleza implica que la/el estudiante implemente los distintos algoritmos que se revisan en el curso, por lo tanto es importante que demuestre que efectivamente puede implementar, revisar y mejorar los algoritmos ya existentes.

Hay dos maneras de certificar la materia: a) Portafolio y b) Examen de certificación. Al menos una semana antes de que termine el curso las y los estudiantes tendrán conocimiento de sus calificaciones parciales y por tanto de la calificación promedio obtenida al momento, para que sea el/la mismo(a) estudiante quién decida si certifica por la modalidad de portafolio, o por la modalidad de examen de certificación.

El portafolio se conforma de evaluaciones (40 %), programas, (40 %) tareas (10 %), tareitas (5 %)y trabajos adicionales (5 %). Mientras que la certificación es un examen elaborado por el comité de certificación y que será presentado por todas y todos los estudiantes que se inscriban en esta modalidad en las fechas establecidas por la Coordinación de Certificación y Registro. En cualquiera de las dos modalidades es indispensable que el/la estudiante se registre a este proceso para que su calificación pueda ser asignada al final del proceso de Certificación.

## 1.3. De la naturaleza del curso

El curso tiene un fuerte sustento en la programación constante, sin embargo, es importante resaltar que los conceptos teóricos deben ser dominados por las y los estudiantes, por lo tanto las evaluaciones y las tareas tendrán estas dos componentes principales. Al contrario de lo que pueda pensarse la asistencia al curso es obligatoria pero no influye directamente en la calificación obtenida. Sin embargo, hay que mencionar que si la asistencia

se realiza de manera intermitente es probable que cueste un poco de trabajo reincorporarse a la dinámica de trabajo que se irá construyendo con el grupo con el transcurso de las clases.

## 1.4. Introducción

En este curso estudiaremos los elementos básicos de los métodos numéricos, utilizando el programa de distribución libre *R*. Antes de iniciar propiamente con el estudio de los métodos numéricos realizaremos un breve repaso de algunos conceptos de álgebra lineal, cálculo diferencial mismos que son fundamentales en esta materia y de la que se supone las y los estudiantes se encuentran familiarizados con ellos.

*Análisis Numérico* es una rama de las matemáticas que, mediante el uso de algoritmos iterativos, obtiene soluciones numéricas a problemas en los cuales la matemática simbólica (o analítica) resulta poco eficiente o no puede ofrecer un resultado. En particular, a estos algoritmos se les denomina *métodos numéricos*.Por lo general los métodos numéricos se componen de un número de pasos finitos que se ejecutan de manera lógica, mejorando aproximaciones iniciales a cierta cantidad, tal como la raíz de una ecuación, hasta que se cumple con cierta cota de error. A esta operación cíclica de mejora del valor se le conoce como *iteración*. El análisis numérico es una alternativa muy eficiente para la resolución de ecuaciones, tanto algebraicas (polinomios) como trascendentes teniendo una ventaja muy importante respecto a otro tipo de métodos: La repetición de instrucciones lógicas (iteraciones), proceso que permite mejorar los valores inicialmente considerados como solución. Dado que se trata siempre de la misma operación lógica, resulta muy pertinente el uso de recursos de cómputo para realizar esta tarea. Sin embargo, debe haber claridad en el sentido de que el análisis numérico no es la panacea en la solución de problemas matemáticos; los métodos numéricos arrojan *aproximaciones*, es decir, están sujetos a un error. Esto quiere decir que si se puede ser tan preciso como los recursos de cálculo lo permitan, siempre está presente y debe considerarse su manejo en el desarrollo de las soluciones requeridas. El uso de diversos sistemas de cómputo determina qué soluciones analítico-numéricas son viables en la práctica, lo que implica que se deben tomar en cuenta el proceso iterativo, el costo de los recursos físicos que se emplean en el análisis, y el tipo de práctica de la Ingeniería.

## 1.5. Revisión de temas importantes

### 1.5.1. Cálculo

Comenzamos el capítulo con un repaso de algunos aspectos importantes del cálculo que son necesarios a lo largo del texto. Suponemos que los estudiantes que lean este texto conocen la terminología, la notación y los resultados que se dan en un curso típico de cálculo.

### Límites y continuidad

El límite de una función nos dice qué tan cerca se encuentran las imágenes de una función si dos elementos de su dominio se encuentran lo suficientemente cerca, es decir, decimos que una función $f(x)$ definida en el intervalo $(a, b)$ tiene **límite** $L$ en el punto $x = x_0$, lo que denotamos por

$$\lim_{x \to x_0} f(x) = L,$$

si para cualquier $\varepsilon > 0$, existe un número real $\delta > 0$ tal que $|f(x) - L| < \varepsilon$ siempre que $0 < |x - x_0| < \delta$. Es decir, que los valores de la función estarán cerca de $L$ siempre que $x$ esté suficientemente cerca de $x_0$.

Se dice que una función $f$ es continua en $a$ si cuando $x$ se aproxima al valor de $a$, entonces también $f(x)$ se aproxima a $f(a)$, es decir, $f(x)$ es **continua** en el punto $x = a$ si

$$\lim_{x \to a} f(x) = f(a),$$

y se dice que $f$ es **continua en el conjunto** $(a, b)$ si es continua en cada uno de los puntos del intervalo. Denotaremos el conjunto de todas las funciones $f$ que son continuas en $E = (a, b)$ por $C(E)$.

Se dice que una sucesión $\{x_n\}_{n=1}^{\infty}$ **converge** a un número $x$, si $\lim_{n\to\infty} x_n = x$ o bien, $x_n \to x$ cuando $n \to \infty$, si para cualquier $\varepsilon > 0$, existe un número natural $N(\varepsilon)$ tal que $|x_n - x| < \varepsilon$ para cada $n > N(\varepsilon)$. Cuando una sucesión tiene límite, se dice que la **sucesión converge**.

### Continuidad y convergencia de sucesiones

Si $f(x)$ es una función definida en un conjunto $S$ de números reales y $x_0 \in S$, entonces las siguientes afirmaciones son equivalentes:

1. $f(x)$ es continua en $x = x_0$,

2. Si $\{x_n\}_{n=1}^{\infty}$ es cualquier sucesión en $S$ que converge a $x_0$, entonces $\lim_{n\to\infty} f(x_n) = f(x_0)$.

**Teorema 1** (Teorema del valor intermedio o de Bolzano.)**.** *Si $f \in C[a, b]$ y $\ell$ es un número cualquiera entre $f(a)$ y $f(b)$, entonces existe al menos un número $c \in (a, b)$ tal que $f(c) = \ell$. Véase la Figura 1.1.*

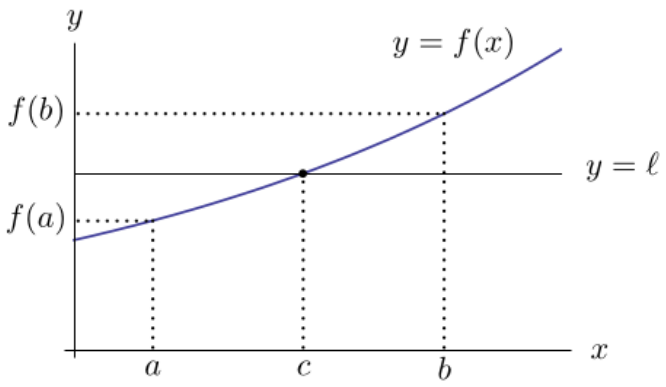


Figura 1.1: Teorema del valor intermedio o de Bolzano

**Nota 1.** *Todas las funciones con las que se van a trabajar en este curso de métodos numéricos serán continuas, ya que esto es lo mínimo que debemos exigir para asegurar que la conducta de un método se puede predecir.*

### 1.5.2. Derivabilidad

Si $f(x)$ es una función definida en un intervalo abierto que contiene un punto $x_0$, entonces se dice que $f(x)$ es **derivable** en $x = x_0$ cuando existe el límite

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}.$$

El número $f'(x_0)$ se llama **derivada** de $f$ en $x_0$ y coincide con la pendiente de la recta tangente a la gráfica de $f$ en el punto $(x_0, f(x_0))$, Figura 1.2.

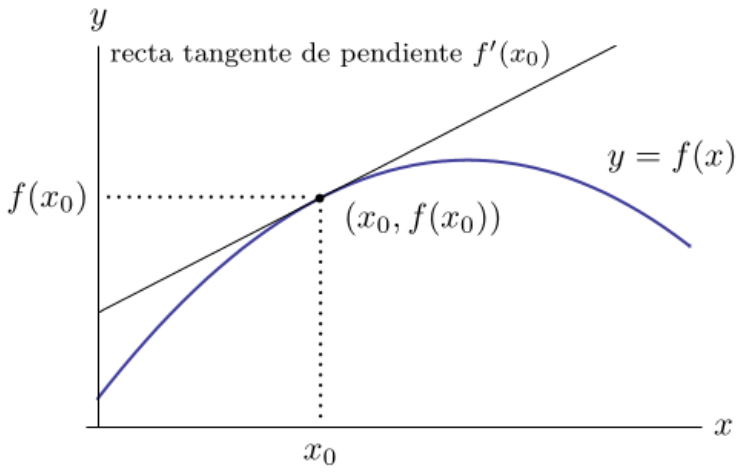


Figura 1.2: Derivada de una función en un punto

**Derivabilidad implica continuidad.** Si la función $f(x)$ es derivable en $x = x_0$, entonces $f(x)$ es continua en $x = x_0$. El conjunto de todas las funciones que admiten

$n$ derivadas continuas en $S$ se denota por $C^n(S)$, mientras que el conjunto de todas las funciones indefinidamente derivables en $S$ se denota por $C^\infty(S)$. Las funciones polinómicas, racionales, trigonométricas, exponenciales y logarítmicas están en $C^\infty(S)$, siendo $S$ el conjunto de puntos en los que están definidas.

**Teorema 2** (Teorema del valor medio o de Lagrange.). *Si $f \in C[a,b]$ y es derivable en $(a,b)$, entonces existe un punto $c \in (a,b)$ tal que*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

Geométricamente hablando, Figura 1.3, el teorema del valor medio dice que hay al menos un número $c \in (a,b)$ tal que la pendiente de la recta tangente a la curva $y = f(x)$ en el punto $(c, f(c))$ es igual a la pendiente de la recta secante que pasa por los puntos $(a, f(a))$ y $(b, f(b))$.

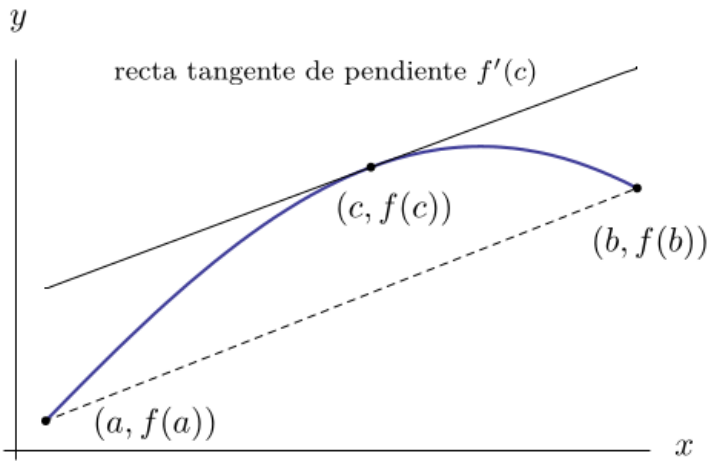


Figura 1.3: Teorema del valor medio o de Lagrange

**Teorema 3** (Teorema de los valores extremos.). *Si $f \in C[a,b]$, entonces existen $c_1$ y $c_2$ en $(a,b)$ tales que $f(c_1) \leq f(x) \leq f(c_2)$ para todo $x \in [a,b]$. Si además, $f$ es derivable en $(a,b)$, entonces los puntos $c_1$ y $c_2$ están en los extremos de $[a,b]$ o bien son puntos críticos.*

### 1.5.3. Integración

**Teorema 4** (Primer teorema fundamental o regla de Barrow.). *Si $f \in C[a,b]$ y $F$ es una primitiva cualquiera de $f$ en $[a,b]$ (es decir, $F'(x) = f(x)$), entonces*

$$\int_a^b f(x)\,dx = F(b) - F(a).$$

**Teorema 5** (Segundo teorema fundamental.). *Si $f \in C[a,b]$ y $x \in (a,b)$, entonces*

$$\frac{d}{dx}\int_a^x f(t)\,dt = f(x).$$

**Teorema 6** (Teorema del valor medio para integrales.)**.** *Si $f \in C[a,b]$, $g$ es integrable en $[a,b]$ y $g(x)$ no cambia de signo en $[a,b]$, entonces existe un punto $c \in (a,b)$ tal que*

$$\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx.$$

**Nota 2.** *Cuando $g(x) = 1$, véase la Figura 1.4, este resultado es el habitual teorema del valor medio para integrales y proporciona el valor medio de la función $f$ en el intervalo $[a,b]$, que está dado por*

$$f(c) = \frac{1}{b-a}\int_a^b f(x)\,dx.$$

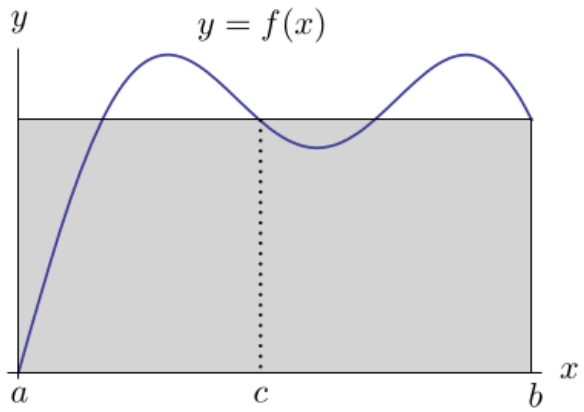


Figura 1.4: Teorema del valor medio para integrales

### 1.5.4. Polinomios de Taylor

**Teorema 7** (Teorema de Taylor.)**.** *Supongamos que $f \in C^{(n)}[a,b]$ y que $f^{(n+1)}$ existe en $[a,b]$. Sea $x_0$ un punto en $[a,b]$. Entonces, para cada $x$ en $[a,b]$, existe un punto $\xi(x)$ entre $x_0$ y $x$ tal que*

$$f(x) = P_n(x) + R_n(x),$$

*donde*

$$P_n(x) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}h^n = \sum_{k=0}^{n}\frac{f^{(k)}(x_0)}{k!}h^k, \quad (1.1)$$

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}h^{n+1}, \quad y \quad h = x - x_0. \quad (1.2)$$

El polinomio $P_n(x)$ se llama **n-ésimo polinomio de Taylor** de $f$ alrededor de $x_0$ (véase la Figura 1.5). $R_n(x)$ se llama **error de truncamiento** (o *resto de Taylor*) asociado a $P_n(x)$. Como el punto $\xi(x)$ en el error de truncamiento $R_n(x)$ depende del punto $x$ en el que se evalúa el polinomio $P_n(x)$, podemos verlo como una función de la variable $x$.

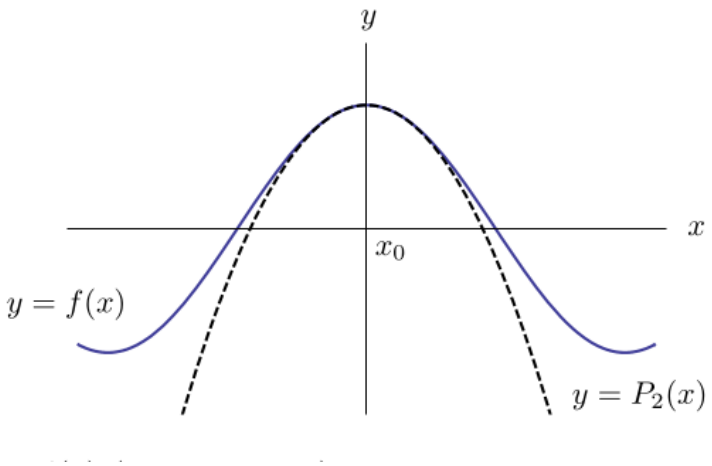


Figura 1.5: Gráficas de $y = f(x)$ y de su polinomio de Taylor $y = P_2(x)$ alrededor de $x_0$.

**Nota 3.** *La serie infinita que resulta al tomar límite en la expresión de $P_n(x)$ cuando $n \to \infty$ se llama* ***serie de Taylor*** *de $f$ alrededor de $x_0$. Cuando $x_0 = 0$, el polinomio de Taylor se suele denominar* ***polinomio de Maclaurin****, y la serie de Taylor se llama* ***serie de Maclaurin****.*

*La denominación error de truncamiento en el teorema de Taylor se refiere al error que se comete al usar una suma truncada al aproximar la suma de una serie infinita.*

### 1.5.5. Teoremas adicionales

A continuación enunciaremos algunas de las definiciones y teoremas básicos que utilizaremos a lo largo de estas notas.

**Definición 1.** *$f$ es de clase $C^1$ en el intervalo $[a; b]$ si $f'$ es continua en $[a; b]$.*

**Definición 2.** *$f$ es de clase $C^n$ en el intervalo $[a; b]$ si $f^{(n)}$ es continua en $[a; b]$.*

**Definición 3.** *$f$ es de clase $C^\infty$ en el intervalo $I$ si $f$ es infinitas veces derivable y continua en $I$.*

**Teorema 8.** *(Teorema de los valores intermedios). Sea $f$ continua en el intervalo $[a; b]$. Si $k \in \mathbb{R}$ es un número comprendido entre $f(a)$ y $f(b)$, entonces existe al menos un punto $\xi$ perteneciente al intervalo $(a; b)$ tal que $f(\xi) = k$.*

**Teorema 9** (Bolzano)**.** *Si $f$ es continua en el intervalo $[a; b]$ y $f(a) \cdot f(b) < 0$, entonces existe un $\xi$ perteneciente al $(a; b)$ tal que $f(\xi) = 0$.*

**Teorema 10** (Teorema de acotabilidad)**.** *Si $f : [a; b] \mapsto \mathbb{R}$ es continua en $[a; b]$, entonces $f$ está acotada en $[a; b]$.*

**Teorema 11** (Teorema de Weierstrass)**.** *Si $f : [a; b] \mapsto \mathbb{R}$ es continua en $[a; b]$, entonces $f$ tiene un máximo global y un mínimo global en $[a; b]$.*

**Teorema 12** (Teorema Generalizado de Rolle)**.** *Si $f$ continua en $[a;b]$, y existen las derivadas $f'(x), f''(x), \ldots, f^{(n)}(x)$ en $(a;b)$ y $f(x_0) = f(x_1) = \cdots = f(x_n) = 0$ (con $x_0, x_1, \ldots, x_n \in [a;b]$) entonces existe $\xi$ perteneciente al $(a;b)$ tal que $f^{(n)}(\xi) = 0$.*

**Teorema 13** (Teorema de Lagrange)**.** *Si $f$ continua en $[a;b]$ y derivable en $(a;b)$ entonces existe $\xi$ perteneciente al $(a;b)$ tal que $f(b) - f(a) = f'(\xi)(b-a)$.*

**Teorema 14** (Teorema del valor medio ponderado)**.** *Sea $f$ continua en $[a,b]$ y $g$ una función integrable Riemann en $[a,b]$. Si $g$ no cambia de signo en $[a,b]$, entonces existe un número $c \in (a,b)$ tal que:*

$$\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx$$

## 1.6. álgebra Lineal

### 1.6.1. Matrices

Una **matriz** es un arreglo multidimensional de escalares, llamados *elementos*, ordenados en filas y columnas. Una matriz de $m$ filas y $n$ columnas, o *matriz (de orden)* $m \times n$, es un conjunto de $m \cdot n$ elementos $a_{ij}$, con $i = 1, 2, \ldots, m$ y $j = 1, 2, \ldots, n$, que se representa de la siguiente forma:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Se puede abreviar la representación de la matriz anterior de la forma $A = (a_{ij})$ con $i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$.

Hay una relación directa entre matrices y vectores puesto que podemos pensar una matriz como una composición de vectores fila o de vectores columna. Además, un vector es un caso especial de matriz: un *vector fila* es una matriz con una sola fila y varias columnas, y un *vector columna* es una matriz con varias filas y una sola columna.

### 1.6.2. Operaciones con matrices

- Si $A = (a_{ij})$ y $B = (b_{ij})$ son dos matrices que tienen el mismo orden, $m \times n$, decimos que $A$ y $B$ son **iguales** si $a_{ij} = b_{ij}$ para todo $i = 1, \ldots, m$ y $j = 1, \ldots, n$.

- Si $A = (a_{ij})$ y $B = (b_{ij})$ son dos matrices que tienen el mismo orden $m \times n$, la **suma** de $A$ y $B$ es una matriz $C = (c_{ij})$ del mismo orden con $c_{ij} = a_{ij} + b_{ij}$ para todo $i = 1, \ldots, m$ y $j = 1, \ldots, n$.

- Si $A = (a_{ij})$ es una matriz de orden $m \times n$, la **multiplicación de $A$ por un escalar** $\lambda$ es una matriz $C = (c_{ij})$ del mismo orden $m \times n$ con $c_{ij} = \lambda a_{ij}$ para todo $i = 1, \ldots, m$ y $j = 1, \ldots, n$.

- Si $A = (a_{ij})$ es una matriz de orden $m \times n$, la **matriz traspuesta** de $A$ es la matriz que resulta de intercambiar sus filas por sus columnas, se denota por $A^T$ y es de orden $n \times m$.
- Si $A = (a_{ij})$ es una matriz de orden $m \times p$ y $B = (b_{ij})$ es una matriz de orden $p \times n$, el **producto de $A$ por $B$** es una matriz $C = (c_{ij})$ de orden $m \times n$ con

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, \quad \text{para todo } i = 1, \ldots, m \text{ y } j = 1, \ldots, n.$$

  Obsérvese que el producto de dos matrices solo está definido si el número de columnas de la primera matriz coincide con el número de filas de la segunda.

### 1.6.3. Matrices especiales

- Una matriz $A = (a_{ij})$ es **cuadrada** si tiene el mismo número de filas que de columnas, y de orden $n$ si tiene $n$ filas y $n$ columnas. Se llama **diagonal principal** al conjunto de elementos $a_{11}, a_{22}, \ldots, a_{nn}$.
- Una **matriz diagonal** es una matriz cuadrada que tiene algún elemento distinto de cero en la diagonal principal y ceros en el resto de elementos.
- Una matriz cuadrada con ceros en todos los elementos por encima (debajo) de la diagonal principal se llama **matriz triangular inferior (superior)**.
- Una matriz diagonal con unos en la diagonal principal se denomina **matriz identidad** y se denota por $I$. Es la única matriz cuadrada tal que $AI = IA = A$ para cualquier matriz cuadrada $A$.
- Una **matriz simétrica** es una matriz cuadrada $A$ tal que $A = A^T$.
- La **matriz cero** es una matriz con todos sus elementos iguales a cero.
- Decimos que una matriz cuadrada $A$ es **invertible** (o *regular* o *no singular*) si existe una matriz cuadrada $B$ tal que $AB = BA = I$. Se dice entonces que $B$ es la **matriz inversa** de $A$ y se denota por $A^{-1}$. (Una matriz que no es invertible se dice *singular*.)
- Si una matriz $A$ es invertible, su inversa también lo es y $A^{-1})^{-1} = A$.
- Si $A$ y $B$ son dos matrices invertibles, su producto también lo es y $(AB)^{-1} = B^{-1}A^{-1}$.

### 1.6.4. Determinante de una matriz

El **determinante** de una matriz solo está definido para matrices cuadradas y su valor es un escalar. El determinante de una matriz $A$ cuadrada de orden $n$ se denota por $|A|$ o $\det(A)$, y se define como

$$\det(A) = \sum_{j} (-1)^{i+j} a_{ij} \cdot \det(A_{ij}),$$

donde la suma se toma para todas las $n!$ permutaciones de grado $n$ y $s$ es el número de intercambios necesarios para poner el segundo subíndice en el orden $1, 2, \ldots, n$.

**Algunas propiedades de los determinantes son:**

- $\det(A^T) = \det(A)$
- $\det(AB) = \det(A)\det(B)$
- $\det(A^{-1}) = \frac{1}{\det(A)}$
- Si dos filas o dos columnas de una matriz coinciden, el determinante de esta matriz es cero.
- Cuando se intercambian dos filas o dos columnas de una matriz, su determinante cambia de signo.
- El determinante de una matriz diagonal es el producto de los elementos de la diagonal.
- Si denotamos por $A_{ij}$ la matriz de orden $(n-1)$ que se obtiene de eliminar la fila $i$ y la columna $j$ de la matriz $A$, llamamos **menor complementario** asociado al elemento $a_{ij}$ de la matriz $A$ al $\det(A_{ij})$.
- Se llama **$k$-ésimo menor principal** de la matriz $A$ al determinante de la submatriz principal de orden $k$.
- Definimos el **cofactor** del elemento $a_{ij}$ de la matriz $A$ por $\Delta_{ij} = (-1)^{i+j}\det(A_{ij})$.
- Si $A$ es una matriz invertible de orden $n$, entonces

$$A^{-1} = \frac{1}{\det(A)}C,$$

donde $C$ es la matriz de elementos $\Delta_{ij}$ para todo $i, j = 1, 2, \ldots, n$. Obsérvese entonces que una matriz cuadrada es invertible si y sólo si su determinante es distinto de cero.

### 1.6.5. Valores propios y vectores propios

- Si $A$ es una matriz cuadrada de orden $n$, un número $\lambda$ es un **valor propio** de $A$ si existe un vector no nulo $v$ tal que $Av = \lambda v$. Al vector $v$ se le llama **vector propio** asociado al valor propio $\lambda$.
- El valor propio $\lambda$ es solución de la **ecuación característica**

$$\det(A - \lambda I) = 0,$$

donde $\det(A - \lambda I)$ se llama **polinomio característico**. Este polinomio es de grado $n$ en $\lambda$ y tiene $n$ valores propios (no necesariamente distintos).

### 1.6.6. Normas vectoriales y normas matriciales

Para medir la **longitud** de los vectores y el **tamaño** de las matrices se suele utilizar el concepto de **norma**, que es una función que toma valores reales. Un ejemplo simple en el espacio euclidiano tridimensional es un vector $v = (v_1, v_2, v_3)$, donde $v_1, v_2$ y $v_3$ son las distancias a lo largo de los ejes $x, y, z$ respectivamente.

La **longitud del vector** $v$ (es decir, la distancia del punto $(0, 0, 0)$ al punto $(v_1, v_2, v_3)$) se calcula como

$$\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2},$$

donde la notación $\|v\|$ indica que esta longitud se refiere a la *norma euclidiana* del vector $v$. De forma similar, para un vector $v$ de dimensión $n$, $v = (v_1, v_2, \ldots, v_n)$, la norma euclidiana se calcula como

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}.$$

Este concepto puede extenderse a una matriz $m \times n$, $A = (a_{ij})$, de la siguiente manera:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2},$$

que recibe el nombre de **norma de Frobenius**.

Hay otras alternativas a las normas euclidiana y de Frobenius. Dos normas usuales son la **norma 1** y la **norma infinito**:

- La **norma 1** de un vector $v = (v_1, v_2, \ldots, v_n)$ se define como $\|v\|_1 = \sum_{i=1}^{n} |v_i|$. De forma similar, la norma 1 de una matriz $m \times n$, $A = (a_{ij})$, se define como

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|.$$

- La **norma infinito** de un vector $v = (v_1, v_2, \ldots, v_n)$ se define como $\|v\|_\infty = \max_i |v_i|$. La norma infinito de una matriz $m \times n$, $A = (a_{ij})$, se define como

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|.$$

- Todas las normas son equivalentes en un espacio vectorial de dimensión finita.

## 1.7. Breve historia de los Métodos Numéricos

Un *método numérico* es un proceso matemático *iterativo* cuyo objetivo es encontrar la aproximación a una solución específica con un cierto error previamente determinado. Los métodos numéricos requieren de una aproximación a la solución real al problema, misma

que es corregida a través de la repetición de un cierto proceso que debe arrojar soluciones cada vez más cercanas al valor real. Cada corrección de un valor inicial se conoce como *iteración*. El proceso es controlado por medio de la medición de una cantidad de error predefinido entre dos aproximaciones sucesivas.

La historia de los métodos numéricos es la colección de acontecimientos matemáticos en los que se resuelven problemas sin el uso de la matemática analítica. Algunos de los métodos más utilizados en la actualidad fueron creados mucho antes de la invención de la computadora; su aplicación era extenuante y complicada porque cada iteración requería de una diversidad de operaciones aritméticas que se realizaban por grupos enteros de calculistas, evidentemente, de forma manual. La historia de los métodos numéricos es paralela, al menos desde la mitad del siglo XIX, a la historia de la computación. Las contribuciones más actuales radican en la creación de software que minimiza los errores y mejora las aproximaciones de los resultados [**?**].

- 1650 a.C. Se crean los Papiros de Rhynd en los que se escribe un método para resolver expresiones matemáticas sin álgebra.
- 250 a.C. Euclides crea el Método de Exhaustión, que consiste en aproximar figuras geométricas (triángulos, cuadrados, pentágonos, etc.) consecutivamente dentro de un círculo para obtener una aproximación a $\pi$.
- Siglo IX d.C. Al Juarismi crea los *algoritmos*.
- 1623. John Napier inventa los *huesos de Napier*, que son arreglos prácticos de logaritmos en tablas.
- Siglo XVII. Isaac Newton crea los procesos de interpolación polinomial.
- Siglo XVIII. Leibnitz crea el Cálculo diferencial.
- 1768. Euler crea soluciones aproximadas a ecuaciones diferenciales con el principio de la integración numérica. Jacob Stirling y Brook Taylor presentan el Cálculo de diferencias finitas.
- 1822. Charles Babbage inventa la *Máquina diferencial*.
- 1843. Ada, condesa de Lovelace, publica sus notas sobre la máquina analítica de Charles Babbage.
- 1890. (IBM) Tabula el censo estadounidense empleando las máquinas de tarjetas perforadas de Herman Hollerith.
- 1931. Vannebar Bush diseña el analizador diferencial, un computador analógico electromecánico. En 1945 publicará el artículo *Cómo podremos pensar* en el que describe la computación personal.
- 1937. Alan Turing publica *Sobre los números computables*, en el que describe un computador universal. En este mismo año, Howard Aiken propone la construcción de un gran computador y descubre partes de la máquina diferencial de Babbage en Harvard; también John Vincent Atanasoff conceptualiza el computador electrónico (la cual completará en 1939).

- 1938. William Hewlett y David Packard crean su empresa en Palo Alto, California, Estados Unidos.
- 1939. Turing comienza a descifrar los códigos secretos alemanes.
- 1944. John Von Neumann redacta el primer informe sobre EDVAC. En distintas universidades de Estados Unidos se desarrollan proyectos sobre computadoras cuya aplicación (secreta) será apoyar a la milicia en cálculos balísticos (ecuaciones diferenciales).
- 1950. Turing crea su famosa prueba sobre la inteligencia artificial; se suicidará en 1954. J.H. Wilkinson acudió al Laboratorio Nacional de Física de Reino Unido para construir una versión más simple de la máquina de Turing; construyó la *ACE (Automatic Computing Engine)* para resolver cálculos con matrices.
- 1953. John W. Backus, empleado de IBM, desarrolla *FORTRAN (Formulae Translating)*, como una alternativa al uso del lenguaje ensamblador; se usó por primera vez en una IBM 704.
- 1958. Se anuncia la creación de la Agencia de Proyectos de Investigación Avanzada (ARPA).
- 1962. Doug Engelbart publica *Aumentar el intelecto humano*; en 1963, junto con Bill English inventará el ratón.
- 1968. Noyce y Moore fundan *INTEL*.
- 1969. Misión Apolo 11. Katherine Johnson calcula la trayectoria del cohete Mercurio. Dorothy Vaughan se convierte en la supervisora de IBM dentro de la NASA. Mary Jackson es la primera ingeniera aeroespacial en Estados Unidos. Margaret Hamilton escribe el código del programa que controló la nave. Todas ellas tuvieron una participación fundamental para que la misión fuera un éxito.
- 1970. Investigadores visitantes en el *Argone National Laboratory* de Estados Unidos traducen códigos de *ALGOL* para obtener eigenvalores planteados por Wilkinson para incluirlos en *FORTRAN*. De esta labor nace *EISPACK* en 1976 y posteriormente *LINPACK* en 1976.
- 1973. Vint Cerf y Bob Kahn completan los protocolos TCP/IP.
- 1975. Bill Gates y Paul Allen desarrollan el lenguaje de programación *BASIC*; fundan *Microsoft*. Steve Jobs y Steve Wozniak lanzan el *Apple I*.
- 1983. Richard Stallman empieza a desarrollar el proyecto *GNU*.
- 1986. Cleve Moler, a partir de *EISPACK* y *LINPACK*, crea *MATLAB*; funda la empresa *MathWorks*.
- 1991. Linus Torvalds lanza la primera versión de *Linux*. Tim Berbers-Lee anuncia la *World Wide Web*.

# Capítulo 2

# Sistemas de Punto Flotante

## 2.1. Operaciones de punto flotante

### 2.1.1. Sistemas decimal y binario

El sistema numérico que se utiliza frecuentemente es el sistema decimal, en la que la base de expresión es el 10. Sin embargo las computadoras utilizan el sistema binario, sistema de base 2, es decir solamente $\{0, 1\}$.

**Proposición 1.** *Para cualquier número natural $N$, existen $a_0, a_1, a_2, \ldots, a_K$, con $a_i \in \mathbb{R}$ tales que*

$$N = a_K \times 2^K + a_{K-1} \times 2^{K-1} + a_{K-2} \times 2^{K-2} + \cdots + a_1 \times 2 + a_0 \times 2^0 \tag{2.1}$$

*Para ver lo anterior lo que tenemos que hacer es calcula $\frac{N}{2}$, es decir, $\frac{N}{2} = P_0 + \frac{a_0}{2}$, donde $P_0 = a_K \times 2^{K-1} + a_{K-1} \times 2^{K-2} + a_{K-2} \times 2^{K-3} + \cdots + a_1 \times 2^0$, es decir $a_0$ es el resto de dividir $N$ entre 2. Ahora hagamos lo mismo para $P_0$:*

$$\frac{P_0}{2} = a_K \times 2^{K-2} + a_{K-1} \times 2^{K-3} + a_{K-2} \times 2^{K-4} + \cdots + a_2 \times 2^0 + \frac{a_1}{2},$$

*por lo tanto*

$$\frac{P_0}{2} = P_1 + \frac{a_1}{2},$$

*donde*

$$P_1 = a_K \times 2^{K-2} + a_{K-1} \times 2^{K-3} + a_{K-2} \times 2^{K-4} + \cdots + a_2 \times 2^0,$$

*es decir $P_1$ es el resto de dividir $P_0$ entre 2. Siguiendo este procedimiento de manera análoga hasta que encontremos un valor $K$ tal que $P_K = 0$. Por lo tanto tenemos el siguiente algoritmo:*

**Algoritmo 1.** *Para un valor $N$ natural, los términos $a_k$ en la ecuación 2.1 se encuentran*

$$\begin{aligned} N &= 2P_0 + a_0, \\ P_0 &= 2P_1 + a_1, \\ &\vdots \\ P_{K-2} &= 2P_{K-1} + a_{K-1}, \\ P_{K-1} &= 2P_K + a_K, \\ P_K &= 0. \end{aligned} \tag{2.2}$$

**Ejemplo 1.** *Convertir* 24563

$$\begin{aligned}
24563 &= 12281 \times 2 + 1, \quad a_0 = 1 \\
12281 &= 6140 \times 2 + 1, \quad a_1 = 1 \\
6140 &= 3070 \times 2 + 0, \quad a_2 = 0 \\
3070 &= 1535 \times 2 + 0, \quad a_3 = 0 \\
1535 &= 767 \times 2 + 1, \quad a_4 = 1 \\
767 &= 383 \times 2 + 1, \quad a_5 = 1 \\
383 &= 191 \times 2 + 1, \quad a_6 = 1 \\
191 &= 95 \times 2 + 1, \quad a_7 = 1 \\
95 &= 47 \times 2 + 1, \quad a_8 = 1 \\
47 &= 23 \times 2 + 1, \quad a_9 = 1 \\
23 &= 11 \times 2 + 1, \quad a_{10} = 1 \\
11 &= 5 \times 2 + 1, \quad a_{11} = 1 \\
5 &= 2 \times 2 + 1, \quad a_{12} = 1 \\
2 &= 1 \times 2 + 0, \quad a_{13} = 0 \\
1 &= 0 \times 2 + 1, \quad a_{14} = 1
\end{aligned}$$

*Por lo tanto el número binario es:* $(24563)_{10} = (101111111110011)_2$.

**Proposición 2.** *Sea* $Q \in \mathbb{R}$*, tal que* $0 < Q < 1$*, entonces existen términos* $b_1, b_2, \ldots, b_k$ *tales que* $Q = 0.b_1b_2b_3 \cdots b_k$*, y por tanto*

$$Q = b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \cdots + b_k \times 2^{-k} + \cdots \tag{2.3}$$

*Si multiplicamos* $Q$ *por* 2*, se tiene que*

$$2Q = b_1 + b_2 \times 2^{-1} + b_3 \times 2^{-2} + b_4 \times 2^{-3} + \cdots + b_k \times 2^{-k+1} + \cdots$$

*Si* $F_1 = frac(2Q)$*, con* $frac(x)$ *la parte fraccionaria de* $x$*, y* $b_1 = [[2Q]]$*, donde* $[[x]]$ *es la parte entera de* $x$*, entonces*

$$F_1 = b_2 \times 2^{-1} + b_3 \times 2^{-2} + b_4 \times 2^{-3} + \cdots + b_k \times 2^{-k+1} + \cdots ,$$

*de donde*

$$2F_1 = b_2 \times 2^0 + b_3 \times 2^{-1} + b_4 \times 2^{-2} + \cdots + b_k \times 2^{-k+2} + \cdots = b_2 + F_2,$$

*donde* $F_2 = frac(2F_1)$*, y* $b_2 = [[2F_1]]$*. Procediendo de manera análoga para el resto de los términos se tienen las succeciones* $\{b_k\}$ *y* $\{F_k\}$*, dadas por* $b_k = [[2F_{k-1}]]$ *y* $F_k = frac(2F_{k-1})$*, con* $b_1 = [[2Q]]$ *y* $F_1 = frac(2Q)$*. Por lo tanto se tiene la representación binaria de* $Q$ *dada por*

$$Q = \sum_{i=1}^{\infty} b_i 2^{-i} \tag{2.4}$$

**Ejemplo 2.** *Convertir el número* 3.5786. *Sea* $Q = 0.5786$, *entonces*

$$\begin{aligned}
2Q &= 1.1572, b_1 = [[1.1572]] = 1, F_1 = frac(1.1572) = 0.1572 \\
2F_1 &= 0.3144, b_2 = [[0.3144]] = 0, F_2 = frac(0.3144) = 0.3144 \\
2F_2 &= 0.6288, b_3 = [[0.6288]] = 0, F_3 = frac(0.6288) = 0.6288 \\
2F_3 &= 1.2576, b_4 = [[1.2576]] = 1F_4 = frac(1.2576) = 0.2576 \\
2F_4 &= 0.5152, b_5 = [[0.5152]] = 0, F_5 = frac(0.5152) = 0.5152 \\
2F_5 &= 1.0304, b_6 = [[1.0304]] = 1F_6 = frac(1.0304) = 0.0304 \\
2F_6 &= 0.0608, b_7 = [[0.0608]] = 0, F_7 = frac(0.0608) = 0.0608 \\
2F_7 &= 0.1216, b_8 = [[0.1216]] = 0, F_8 = frac(0.1216) = 0.1216
\end{aligned}$$

*De lo anterior se tiene que:*

$$0.5786 = (0.10010100\ldots)_2$$

*Por lo tanto:*

$$3.5786 = (11.10010100\ldots)_2.$$

**Ejercicio 1.** *Convertir los siguientes números de base 10 a base 2.*

1. 324
2. 27
3. 1423
4. 235.25
5. 41.596

### 2.1.2. Números en punto flotante

**Definición 4.** *Los números en punto flotante son número reales de la forma*

$$\pm\alpha \times \beta^e, \tag{2.5}$$

*donde* $\alpha$ *tiene un número de dígitos limitados,* $\beta$ *es la base y* $e$ *es el exponente que modifica la posición del punto decimal.*

**Definición 5.** *Un número real* $x$ *tiene una representación punto flotante normalizada si*

$$\pm\alpha \times \beta^e, \tag{2.6}$$

*con* $\frac{1}{\beta} < |\alpha| < 1$

**Nota 4.** *En este caso* $x$ *se puede escribir en la forma*

$$x = \pm 0, d_1 d_2 \cdots d_k \times \beta^e, \tag{2.7}$$

*donde si* $x \neq 0$,$d_1 \neq 0$, *y además* $0 \leq d_i < \beta$, *para* $i = 1, 2, 3 \ldots, k$ *y* $L \leq e \leq U$.

**Definición 6.** *El conjunto de los números en punto flotante se le llama* ***conjunto de números máquina****.*

**Nota 5.** *El conjunto de números máquina es finito. Para ver esto consideremos que si $x$ es de la forma*

$$\pm 0, d_1 d_2 \cdots d_t \times \beta^e, \tag{2.8}$$

*dado que $d_1 \neq 0$ y $0 \leq d_i < \beta$ entonces $d_1$ puede tomar $\beta - 1$ valores distintos, mientras que para $d_i$ hay $\beta$ posibilidades. Por lo tanto se tienen $(\beta-1)\beta\beta\cdots\beta = (\beta-1)\beta^t$. El número de exponentes posibles son $U - L + 1$, en total hay $(\beta-1)\beta^t(U-L+1)$ números máquina positivos, por lo tanto, considerando positivos y negativos hay $2(\beta-1)\beta^t(U-L+1)$ números máquina, considerando que el creo tambiés es un número de máquina hay en realidad $2(\beta-1)\beta^t(U-L+1)+1$. Es decir, cualquier número real puede ser representado por uno de los $2(\beta-1)\beta^t(U-L+1)+1$ números de máquina.*

**Ejemplo 3.** *Recordemos la expresión (2.8), consideremos $\beta = 2$, $t = 3$, $L = -2$ y $U = 2$. Entonces $x = \pm d_1 d_2 d_3 \times (2)^e$, con $-2 \leq e \leq 2$ y $0 \leq d_1, d_2 < 2$. Por lo tanto se tiene que $d_1, d_2, d_3 = 1$; $e = \{-2, -1, 0, 1, 2\}$. Entonces $d_1 d_2 d_3 = \{100, 101, 110, 111\} = \left\{\frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}\right\}$*

| $-2$ | $-1$ | $0$ | $1$ | $2$ |
|---|---|---|---|---|
| $0.111 \times 2^{-2}$ | $0.111 \times 2^{-1}$ | $0.111 \times 2^{0}$ | $0.111 \times 2^{1}$ | $0.111 \times 2^{2}$ |
| $0.110 \times 2^{-2}$ | $0.110 \times 2^{-1}$ | $0.110 \times 2^{0}$ | $0.110 \times 2^{1}$ | $0.110 \times 2^{2}$ |
| $0.101 \times 2^{-2}$ | $0.101 \times 2^{-1}$ | $0.101 \times 2^{0}$ | $0.101 \times 2^{1}$ | $0.101 \times 2^{2}$ |
| $0.100 \times 2^{-2}$ | $0.100 \times 2^{-1}$ | $0.100 \times 2^{0}$ | $0.100 \times 2^{1}$ | $0.100 \times 2^{2}$ |

(2.9)

*sustituyendo y resolviendo las operaciones*

| $-2$ | $-1$ | $0$ | $1$ | $2$ |
|---|---|---|---|---|
| $\frac{7}{8} \times 2^{-2} = \frac{7}{32}$ | $\frac{7}{8} \times 2^{-1} = \frac{7}{16}$ | $\frac{7}{8} \times 2^{0} = \frac{7}{8}$ | $\frac{7}{8} \times 2^{1} = \frac{7}{4}$ | $\frac{7}{8} \times 2^{2} = \frac{7}{2}$ |
| $\frac{3}{4} \times 2^{-2} = \frac{3}{16}$ | $\frac{3}{4} \times 2^{-1} = \frac{3}{8}$ | $\frac{3}{4} \times 2^{0} = \frac{3}{4}$ | $\frac{3}{4} \times 2^{1} = \frac{3}{2}$ | $\frac{3}{4} \times 2^{2} = 3$ |
| $\frac{5}{8} \times 2^{-2} = \frac{5}{32}$ | $\frac{5}{8} \times 2^{-1} = \frac{5}{16}$ | $\frac{5}{8} \times 2^{0} = \frac{5}{8}$ | $\frac{5}{8} \times 2^{1} = \frac{5}{4}$ | $\frac{5}{8} \times 2^{2} = \frac{5}{2}$ |
| $\frac{1}{2} \times 2^{-2} = \frac{1}{8}$ | $\frac{1}{2} \times 2^{-1} = \frac{1}{4}$ | $\frac{1}{2} \times 2^{0} = \frac{1}{2}$ | $\frac{1}{2} \times 2^{1} = 1$ | $\frac{1}{2} \times 2^{2} = 2$ |

(2.10)

*Por tanto el número total de números máquina son 41.*

**Ejercicio 2.**

1. *Describir todos los números máquina para $\beta = 2$, $t = 4$, $L = -3$ y $U = 3$.*
2. *Escribir el número* $732.5051$ *en notación de punto flotante. Respuesta* $0.7325051 \times 10^{-3}$
3. *Escribir el número* $0.006521$ *en notación de punto flotante. Respuesta* $0.06521 \times 10^{-2}$
4. $(101.01)_2$. *Respuesta* $0.10101 \times 2^3$
5. $(0.00101111)_2$. *Respuesta* $0.101111 \times 2^{-2}$

**Nota 6.** $(0.1)_2 = 1 \times 2^{-1} = 0.5$

### 2.1.3. Representación

En una computadora los números se expresan como se ha descrito, pero con restricciones sobre $q$ y $m$ dadas por la longitud de la palabra. Supongamos que la longitud de la palabra es de 32 bits, los cuales se distribuyen de la siguiente manera: los dos primeros se reservan para los signos: es 0 si es positivo y 1 si es negativo; los siguientes 7 espacios para el exponente, y los restantes para la mantisa. Considerando que cualquier número puede normalizarse, recordar $x = \pm q \times 2^m$, con $\frac{1}{2} \leq q < 1$, se puede asumir que el primer bit en $q$ es 1 y por tanto no se requeire almacenar,.

**Ejemplo 4.** *Representar y almacenar en punto flotante normalizado el número* $-0.125$. *A saber* $(-0.125) = (-0.001)_2 = (-0.1)_2 \times 2^{-2}$, *además* $2 = (10)_2$*; por lo tanto su representación es:* $1, 1|, 0, 0, 0, 0, 0, 1, 0|\ 0, 0, 0, 0, 0, 0, 0, 0, \ldots 0$.

**Ejemplo 5.** *Represente y almacene el número* 117.125. *Respuesta* $117 = (1110101)_2$ *y* $0.125 = (0.001)_2$, *por tanto* $117.125 = (1110101.001)_2 = (0.1110101001)_2 \times 2^7$ *y* $y = (111)_2$, *por tanto se almacena:* $0, 0||0000111||110101 \ldots 0$

**Nota 7.** $|m|$ *no requiere más de* 7 *bits, es decir* $|m| \leq (1111111)_2 = 2^7 - 1 = 127$, *por tanto el exponente de* 7 *dígitos binarios proporciona un intervalo de* 0 *a* 127, *para números pequeños se toma el exponente en el intervalo* $[-63, 64]$. *Además q requiere de a lo más 24 bits, por tanto la máquina de* 32 *bits tienen una precisión limitada entre* 7 *y* 8 *dígitos decimales, ya que el bit menos significativo en la mantisa representa unidades del orden* $2^{-24} \approx 10^{-7}$. *Esto quiere decir que números expresados por más de siete dígitos decimales serán aproximados cuando se dan como datos de entrada o como resultados de operaciones.*

### 2.1.4. Errores

A la hora de realizar un cálculo es importante asegurarse de que los números que intervienen en el cálculo se pueden utilizar con confianza. Para ello, se introduce el concepto de **cifras significativas**, que designa formalmente la notación de un valor numérico, y se usa en aquellos que guían visualmente la precisión.

Los **errores de truncamiento** se producen cuando utilizamos una aproximación en lugar de un procedimiento matemático exacto. Para conocer las características de estos errores se suelen considerar los polinomios de Taylor. Cuando se aproxima un proceso continuo por uno discreto, para errores provocados por un tamaño de paso finito $h$, resulta a menudo útil describir la dependencia del error $e$ con $h$ cuando $h$ tiende a cero.

Decimos que una función $f(h)$ es una $\mathcal{O}$ grande de $h^n$ si $|f(h)| \leq ch^n$ para alguna constante $c$, cuando $h$ es cercano a cero; se escribe

$$f(h) = \mathcal{O}(h^n) \tag{2.11}$$

.

Si un método tiene un término de error que es $\mathcal{O}(h^n)$, se suele decir que es un **método de orden** $n$. Por ejemplo, si utilizamos un polinomio de Taylor para aproximar la función

$g$ en $x = a + h$, tenemos:v

$$g(x) = g(a+h) = g(a) + hg'(a) + \frac{h^2}{2!}g''(a) + \frac{h^3}{3!}g^{(3)}(\xi), \quad \text{para algún } \xi \in [a, a+h].$$

Suponiendo que $g$ es suficientemente derivable, la aproximación anterior es $\mathcal{O}(h^3)$, puesto que el error

$$\frac{h^3}{3!}g^{(3)}(\xi), \quad \text{satisface} \quad \left|\frac{h^3}{3!}g^{(3)}(\xi)\right| \leq \frac{M}{3!}|h^3|,$$

donde $M$ es el máximo de $g^{(3)}(x)$ para $x \in [a, a+h]$.

Las diferencias (errores) son múltiples y de diversa naturaleza, aunque pueden separarse en dos grupos genéricos:

- **Los errores que provienen del modelado teórico** (o abstracción matemática) del fenómeno real; estos errores se denominan *Errores del modelo o inherentes*. Los errores inherentes son producto de factores intrínsecos a la naturaleza, al ambiente y las personas mismas. Los errores inherentes son imposibles de remediar aunque pueden minimizarse; en consecuencia, no pueden cuantificarse.

  > Se distinguen dos tipos de errores inherentes: **Las incertidumbres** hacen referencia a las dimensiones físicas que nunca podrán ser medidas en forma exacta debido a la naturaleza de la materia y a las imperfecciones de los instrumentos de medición. **Las verdaderas equivocaciones** son las situaciones que se producen en la lectura de instrumentos de medición o en el traslado de información y que son inadvertidas a las personas; un claro ejemplo de estas situaciones es la denominada *ceguera de taller*.

- **Los errores del método** son producto de la limitante en la representación y manipulación de cantidades numéricas utilizadas en los cálculos necesarios en el desarrollo del modelo matemático. Es de destacar que los dispositivos de cálculo (tales como calculadoras y computadoras) utilizan y manipulan cantidades en forma imprecisa.

Existen dos grandes tipos de errores del método: *El truncamiento* se provoca ante la imposibilidad de manipular, por parte de un instrumento de cómputo, una cantidad infinita de términos o cifras. Los términos o cifras omitidas (que son infinitas en número) introducen un error en los resultados calculados. *El redondeo* se produce por el mismo motivo que el truncamiento pero, a diferencia de éste, las cifras omitidas sí son consideradas en la cifra resultante

En general, si se incrementa el número de cifras significativas en el ordenador, se minimizan los errores de redondeo, y los errores de truncamiento disminuyen a medida que los errores de redondeo se incrementan. Por lo tanto, para disminuir uno de los dos sumandos del error total debemos incrementar el otro. Como el error total no se puede calcular en la mayoría de los casos, se suelen utilizar otras medidas para estimar la exactitud de un método numérico, que suelen depender del método específico. En algunos métodos el error numérico se puede acotar, mientras que en otros se determina una estimación del *orden de magnitud del error*. Cuando se buscanlas soluciones numéricas de un problema real los resultados que se obtienen por lo general no son exactos.

### 2.1.5. Cuantificación de errores

Los errores se cuantifican de dos formas diferentes:

1. **Error absoluto**. El error absoluto es la diferencia absoluta entre un valor real y un aproximado. Está dado por la siguiente fórmula:

$$E = |V_{real} - V_{aprox}|$$

   El error absoluto recibe este nombre ya que posee las mismas dimensiones que la variable bajo estudio.

2. **Error relativo**. Corresponde a la expresión en porcentaje de un error absoluto; en consecuencia, este error es adimensional.

$$e = \left| \frac{V_{real} - V_{aprox}}{V_{real}} \right| \cdot 100\,\%$$

La diferencia entre la preferencia en el uso de los dos tipos de error consiste precisamente en la presencia de las dimensiones físicas. Debido a las unidades de medición utilizadas, el manejo y la percepción del error absoluto suele ser engañoso o difícil de comprender rápidamente. Sin embargo, el manejo de porcentajes (o valores relativos) resulta más natural y sencillo de comprender. Sin embargo, el uso de estos dos tipos de errores está sujeto siempre al objetivo de las actividades desarrolladas.

Las expresiones que definen a los errores absoluto y relativo requieren del conocimiento de la variable $V_{real}$ que representa un valor ideal que no posee error alguno. Como podrá suponerse, en la realidad resulta imposible determinar este valor. Una práctica común en los análisis elementales sobre errores es considerar como un valor real a los resultados arrojados por la medición experimental de los fenómenos y a los valores aproximados como los proporcionados por los modelos matemáticos (o viceversa). En realidad, ambos valores son valores aproximados. Para lograr un resultado coherente, en la práctica debe sustituirse al valor real por un valor que se considere posee un error menor.

En el caso del análisis numérico, dado que los resultados se obtienen a partir de procesos iterativos que se mejoran los inicialmente obtenidos, debe partirse del supuesto que el último valor obtenido posee un nivel menor de error que el valor previo. Dado lo anterior, los errores absoluto y relativo se calcularán de la siguiente forma:

**Error absoluto:**

$$E = |V_i - V_{i-1}|$$

**Error relativo:**

$$e = \left| \frac{V_i - V_{i-1}}{V_i} \right| \cdot 100\,\%$$

En ambas ecuaciones, $V_i$ es el valor de la última iteración y $V_{i-1}$ es el valor de la iteración anterior $i - 1$.

- **Error absoluto:** Se define como la diferencia entre el valor real (experimental o exacto) y el valor aproximado (teórico o calculado):

$$E_a = |x_{\text{real}} - x_{\text{aproximado}}|$$

- **Error relativo:** Es la relación entre el error absoluto y el valor real:

$$E_r = \frac{E_a}{|x_{\text{real}}|}$$

- **Error porcentual:** Es el error relativo expresado en porcentaje:

$$E_p = E_r \cdot 100 = \frac{E_a}{|x_{\text{real}}|} \cdot 100$$

### 2.1.6. Errores en punto flotante

Un número real $x \in \mathbb{R}$, cuando no pertenece a $F$, se representa mediante una aproximación flotante $fl(x) \in F$, de modo que:

$$x = fl(x)(1 + \varepsilon), \quad |\varepsilon| \leq \epsilon_{\text{mach}}.$$

Este $\varepsilon$ se llama **error relativo** y $\epsilon_{\text{mach}}$ es la **precisión de la máquina**. En general se tiene que:

$$\epsilon_{\text{mach}} = \frac{1}{2}\beta^{1-t}.$$

**Nota 8.** *Los ordenadores almacenan los números reales en forma binaria (base 2). Cada dígito binario (0 ó 1) se llama **bit** (por digital binary). La memoria de los ordenadores está organizada en **bytes**, siendo un byte = 8 bits. En el estándar IEEE, los ordenadores representan números reales en precisión simple (32 bits) y en precisión doble (64 bits). Este estándar también define los códigos para representar valores especiales como NaN (Not a Number), infinitos, y ceros con signo. Para representar un número real, se utiliza la forma normalizada:*

$$x = (-1)^s \cdot (1 + f) \cdot 2^e,$$

*donde: $s$: es el bit de signo (0 para positivo, 1 para negativo), $f$: es la fracción, $e$: es el exponente con sesgo.*

*Por ejemplo, el número 0.15625 en binario es 0.00101, que se escribe como $1.01 \cdot 2^{-3}$ y se almacena como: signo = 0, exponente = 124 (con sesgo de 127), y mantisa = 010000... En este formato, la precisión depende de la cantidad de bits reservados a la fracción $f$. En doble precisión (64 bits), se reservan 52 bits para la fracción, 11 para el exponente y 1 para el signo.*

### 2.1.7. Aproximación numérica y errores

Una *aproximación* es un valor cercano a uno considerado como real o verdadero. Esta cercanía, o diferencia, se conoce como *error*. Normalmente, la consideración de la validez de una aproximación depende de la cota de error que el experimentador considere pertinente en función del contexto del fenómeno bajo estudio. Esto implica que también debe considerarse que una magnitud debe ser un valor real, que en el ámbito de la Ingeniería pocas veces se conoce, lo que obliga a adoptar convenciones. En Ingeniería, se denomina *exactitud* a la capacidad de un instrumento de medir un valor cercano al de la magnitud real. Exactitud implica precisión, pero no al contrario. Exactitud y precisión no son equivalentes. Exactitud es capacidad para acercarse a la magnitud real, y precisión es la capacidad de generar resultados similares. La precisión se logra cuando un instrumento para repetir mediciones exactas cuando éstas se realizan consecutivamente. De acuerdo con la definición de aproximación numérica, la exactitud se aplica en los métodos numéricos en cuanto a la capacidad del método de generar un resultado muy cercano al valor real; se percibe la cercanía entre la exactitud y el concepto de error. Por otra parte, los métodos numéricos a través de iteraciones generan valores aproximados cada vez más exactos, es decir, estas iteraciones deberán ser precisas. Dado lo anterior, los métodos numéricos deberán tener como cualidades la exactitud y la precisión. Matemáticamente, la **convergencia** es la propiedad de algunas sucesiones y series de tender progresivamente a un límite, de tal forma, si este límite existe, se dice que la sucesión o la serie *converge*. En forma análoga, si un método numérico en su funcionamiento iterativo nos proporciona aproximaciones cada vez más cercanas al valor buscado, se dice que el método converge. La convergencia se mide a través de los errores; si el error entre dos aproximaciones sucesivas se reduce, el método converge; se debe cumplir que:

$$|x_n - x_{n-1}| \leq |x_{n-1} - x_{n-2}|$$

Es decir, la diferencia enésima $(x_n - x_{n-1})$ debe ser menor que la diferencia $(n-1)$ésima $x_{n-1} - x_{n-2}$. Se dice que un sistema (o un proceso) es **estable** si a pequeñas variaciones en la entrada o en la excitación corresponden pequeñas variaciones en la salida o en la respuesta. La estabilidad de un método numérico tiene que ver con la manera en que los errores numéricos se propagan a lo largo del algoritmo. Cuando un método converge, lo más deseable es que en los resultados que se obtengan, los niveles de error se disminuyan en la forma más rápida posible. Sin embargo, ocurre que durante la operación del algoritmo, ya sea por el manejo de los datos numéricos o bien por la naturaleza propia del modelo matemático con el que se esté trabajando, los errores entre aproximaciones no disminuyan en forma progresiva, sino que incluso aumenten en alguna etapa del proceso para después reducirse mostrando un comportamiento aleatorio.

La **robustez** de un método numérico radica en su convergencia y su estabilidad. Pueden utilizarse métodos cuya prueba de convergencia indique la pertinencia de su uso, pero que durante su aplicación se obtengan resultados inestables que repercutan en el número de iteraciones y en consecuencia en el tiempo invertido en la solución. El ideal lo constituyen métodos que a la vez de ser convergentes resulten estables.

**Nota 9.** ***Convergencia*** *de un método se refiere a que sea posible la obtención del valor buscado cuando el número de pasos tiende a infinito. Típicamente, la convergencia se*

*analiza en métodos iterativos, es decir, aquellos en los que el resultado final se obtiene tras una repetición de cálculos. Cuando repetimos estas iteraciones, los datos iniciales producen valoraciones progresivas del resultado.*

### 2.1.8. Ejercicios

1. Realizar una revisión de la historia de los métodos numéricos, elaborar un documento de hasta dos cuartillas.

2. Realiza las siguientes conversiones de base 10 a base 2:

   *a*) 246

   *b*) 345.68

   *c*) 4586632.2846

   *d*) 984365.27463

   *e*) 79905523

3. Elabora el código en R para realizar la conversión de base 10 a base 2.

# Capítulo 3

# Solución de Sistemas de Ecuaciones Lineales

## 3.1. Definiciones sobre matrices

**Definición 7** (Matriz transpuesta)**.** *Sea $A = (a_{ij}) \in \mathbb{R}^{m\times n}$. La **matriz transpuesta** de $A$, denotada $A^\top$, es la matriz $n \times m$ definida por*

$$(A^\top)_{ij} = a_{ji}.$$

**Definición 8** (Matriz simétrica)**.** *Una matriz $A \in \mathbb{R}^{n\times n}$ es **simétrica** si*

$$A^\top = A.$$

**Definición 9** (Matriz definida positiva)**.** *Una matriz simétrica $A \in \mathbb{R}^{n\times n}$ es **definida positiva** si*

$$x^\top Ax > 0 \quad \textit{para todo } x \in \mathbb{R}^n,\ x \neq 0.$$

**Definición 10** (Matriz semidefinida positiva)**.** *Una matriz simétrica $A \in \mathbb{R}^{n\times n}$ es **semi-definida positiva** si*

$$x^\top Ax \geq 0 \quad \textit{para todo } x \in \mathbb{R}^n.$$

Hasta ahora hemos visto métodos **directos** (eliminación Gaussiana, factorizaciones $LU$, Doolittle, Cholesky). En problemas de gran tamaño, estos métodos pueden ser muy costosos en tiempo y memoria. Por ello se usan **métodos iterativos**, que generan una sucesión de aproximaciones $\{x^{(k)}\}$ que converge a la solución exacta $x$ de

$$Ax = b.$$

Un sistema de ecuaciones lineales puede escribirse como:

$$Ax = b, \qquad A = (a_{ij}) \in \mathbb{R}^{n\times n}, \quad x, b \in \mathbb{R}^n.$$

Nuestro objetivo es encontrar $x$. La eliminación gaussiana consiste en aplicar operaciones de filas equivalentes que transforman $A$ en una matriz triangular superior $U$, de modo que el sistema resultante pueda resolverse por sustitución regresiva.

## 3.2. Eliminación gaussiana simple

**Planteamiento.** Sea $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ y $b = (b_i) \in \mathbb{R}^n$. El sistema $Ax = b$ se expresa como:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \qquad i = 1, \ldots, n.$$

**Etapas de eliminación.** En el paso $k$ se anulan las entradas de la columna $k$ por debajo del pivote $a_{kk}^{(k)}$. Para cada $i = k+1, \ldots, n$:

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \qquad a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}\, a_{kj}^{(k)}, \quad b_i^{(k+1)} = b_i^{(k)} - m_{ik}\, b_k^{(k)}.$$

**Ejemplo simbólico** $3 \times 3$**.**

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

*Paso 1 ($k = 1$):*

$$m_{21} = \tfrac{a_{21}}{a_{11}}, \qquad m_{31} = \tfrac{a_{31}}{a_{11}}.$$

$$\begin{aligned} a_{22}^{(2)} &= a_{22} - m_{21}a_{12}, \quad a_{23}^{(2)} = a_{23} - m_{21}a_{13}, \quad b_2^{(2)} = b_2 - m_{21}b_1, \\ a_{32}^{(2)} &= a_{32} - m_{31}a_{12}, \quad a_{33}^{(2)} = a_{33} - m_{31}a_{13}, \quad b_3^{(2)} = b_3 - m_{31}b_1. \end{aligned}$$

*Paso 2 ($k = 2$):*

$$m_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}, \quad a_{33}^{(3)} = a_{33}^{(2)} - m_{32}a_{23}^{(2)}, \quad b_3^{(3)} = b_3^{(2)} - m_{32}b_2^{(2)}.$$

Resultado: sistema triangular superior $Ux = c$.
En el paso $k$ de la eliminación:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik}a_{kj}^{(k)}, \qquad b_i^{(k+1)} = b_i^{(k)} - m_{ik}b_k^{(k)},$$

para $i = k+1, \ldots, n$ y $j = k, \ldots, n$, donde

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

Es decir:

- $m_{ik}$ mide cuántas veces la fila $k$ debe restarse a la fila $i$ para anular $a_{ik}$.
- Cada $m_{ik}$ se almacena en la matriz $L$.

**Idea general.** Transformar el sistema $Ax = b$ en uno triangular superior $Ux = c$ mediante operaciones elementales de filas, y resolver luego por sustitución regresiva.

**Ejemplo.** Resolver el sistema

$$\begin{cases} 2x + y - z = 8, \\ -3x - y + 2z = -11, \\ -2x + y + 2z = -3. \end{cases}$$

Forma matricial:

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}.$$

*Paso 1.* Pivote en $a_{11} = 2$. Eliminamos en la primera columna:

$$m_{21} = \tfrac{-3}{2}, \quad m_{31} = \tfrac{-2}{2} = -1.$$

Se obtiene

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 2 & 1 & 5 \end{array}\right].$$

*Paso 2.* Pivote en $a_{22} = 0.5$. Eliminamos debajo:

$$m_{32} = \tfrac{2}{0.5} = 4.$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & 0.5 & 0.5 & 1 \\ 0 & 0 & -1 & 1 \end{array}\right].$$

*Paso 3.* Sustitución regresiva:

$$z = -1, \quad 0.5y + 0.5z = 1 \Rightarrow y = 3, \quad 2x + y - z = 8 \Rightarrow x = 2.$$

$$\boxed{x = 2,\ y = 3,\ z = -1}.$$

## 3.3. Eliminación con pivoteo y escalamiento

Mismo proceso, pero en cada paso $k$ se intercambia la fila $k$ con la fila $p$ tal que

$$|a_{pk}^{(k)}| = \max_{i \geq k} |a_{ik}^{(k)}|.$$

Esto asegura que $a_{kk}^{(k)}$ sea lo suficientemente grande.

Antes de seleccionar pivote, cada fila $i$ se escala por

$$s_i = \max_j |a_{ij}|,$$

y se elige como pivote aquel que maximiza

$$\frac{|a_{ik}^{(k)}|}{s_i}.$$

**Idea.** Si en algún paso el pivote es 0 o muy pequeño, se intercambia la fila pivote con otra fila inferior que tenga el mayor valor absoluto en la misma columna. Esto aumenta la estabilidad.

**Ejemplo.**

$$\left[\begin{array}{ccc|c} 0 & 2 & 1 & 4 \\ 1 & -1 & 2 & 6 \\ 2 & 1 & -1 & 1 \end{array}\right].$$

*Problema:* $a_{11} = 0$. *Solución:* intercambiamos fila 1 con fila 2.

Nuevo sistema:

$$\left[\begin{array}{ccc|c} 1 & -1 & 2 & 6 \\ 0 & 2 & 1 & 4 \\ 2 & 1 & -1 & 1 \end{array}\right].$$

Ahora se procede con eliminación gaussiana simple. El pivoteo evita la división por cero.

### 3.3.1. Eliminación con Pivoteo y Escalamiento

**Idea.** Si los coeficientes varían mucho en magnitud, usar sólo pivoteo puede ser insuficiente. *Escalamiento*: dividir cada fila por su elemento de mayor valor absoluto antes de seleccionar pivote. Así, se compara $|a_{ik}|/s_i$, donde $s_i = \max_j |a_{ij}|$ es el factor de escala de la fila $i$.

**Comentario.** Esto evita que números muy grandes o muy pequeños enmascaren el pivote real, aumentando la precisión numérica en cómputo.

### 3.3.2. Pivoteo y escalamiento

- **Pivoteo parcial:** intercambiar la fila $k$ con la fila $p$ tal que $|a_{pk}^{(k)}| = \max_{i \geq k} |a_{ik}^{(k)}|$.
- **Escalamiento:** definir factores $s_i = \max_j |a_{ij}|$ y escoger como pivote el mayor cociente

$$\frac{|a_{ik}^{(k)}|}{s_i}.$$

  Esto controla la propagación de errores de redondeo.

## 3.4. Gauss–Jordan y cálculo de inversas

**Idea.** Extender la eliminación hasta reducir $A$ a la matriz identidad. Si se parte de la matriz aumentada $(A|I)$, el resultado final es $(I|A^{-1})$.

Si extendemos la eliminación a toda la matriz (no solo a triangular superior), obtenemos:

$$A \longrightarrow I, \quad (A|I) \longrightarrow (I|A^{-1}).$$

Algebraicamente:

$$A^{-1} = M^{(n-1)^{-1}} \cdots M^{(2)^{-1}} M^{(1)^{-1}},$$

donde cada $M^{(k)}$ es la matriz elemental usada en la etapa $k$ de la eliminación.

### 3.4.1. Gauss–Jordan e inversas

En vez de detener la eliminación en forma triangular superior, se continúa hasta obtener la matriz identidad:

$$(A|I) \longrightarrow (I|A^{-1}).$$

Así, cada paso se formaliza como multiplicación por matrices elementales:

$$A^{-1} = M_1^{-1} M_2^{-1} \cdots M_r^{-1}.$$

**Ejemplo.** Calcular $A^{-1}$ con

$$A = \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix}.$$

Matriz aumentada:

$$\left[\begin{array}{cc|cc} 2 & 1 & 1 & 0 \\ 5 & 3 & 0 & 1 \end{array}\right].$$

*Paso 1.* Pivote 2. Normalizar fila 1:

$$\left[\begin{array}{cc|cc} 1 & 0.5 & 0.5 & 0 \\ 5 & 3 & 0 & 1 \end{array}\right].$$

*Paso 2.* Eliminar columna 1 de fila 2:

$$R_2 \leftarrow R_2 - 5R_1.$$

$$\left[\begin{array}{cc|cc} 1 & 0.5 & 0.5 & 0 \\ 0 & 0.5 & -2.5 & 1 \end{array}\right].$$

*Paso 3.* Normalizar fila 2:

$$\left[\begin{array}{cc|cc} 1 & 0.5 & 0.5 & 0 \\ 0 & 1 & -5 & 2 \end{array}\right].$$

*Paso 4.* Eliminar columna 2 de fila 1:

$$R_1 \leftarrow R_1 - 0.5R_2.$$

$$\left[\begin{array}{cc|cc} 1 & 0 & 3 & -1 \\ 0 & 1 & -5 & 2 \end{array}\right].$$

**Resultado.**

$$A^{-1} = \begin{bmatrix} 3 & -1 \\ -5 & 2 \end{bmatrix}.$$

## 3.5. Factorización $LU$

La **factorización** $LU$ consiste en descomponer una matriz cuadrada

$$A \in \mathbb{R}^{n\times n}$$

en el producto de dos matrices:

$$A = LU, \tag{3.1}$$

donde:

- $L$ es una matriz **triangular inferior** con unos en la diagonal.
- $U$ es una matriz **triangular superior**.

**Nota 10.** *En algunos casos es necesario introducir una matriz de permutación $P$ para realizar el proceso de eliminación gaussiana de manera estable:*

$$PA = LU. \tag{3.2}$$

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ m_{n1} & \cdots & m_{n,n-1} & 1 \end{bmatrix}, \tag{3.3}$$

$U =$ matriz triangular superior obtenida tras la eliminación.

Por tanto:

$$A = LU.$$

El procedimiento se basa en la eliminación gaussiana. Dada

$$A = (a_{ij}) \in \mathbb{R}^{n\times n}, \tag{3.4}$$

se definen los **multiplicadores**:

$$m_{ij} = \frac{a_{ij}^{(k)}}{a_{kk}^{(k)}} \quad \text{para } i > k, \tag{3.5}$$

donde $a_{ij}^{(k)}$ denota la entrada de la matriz en la etapa $k$ de la eliminación.

Los pasos son:

1. Inicialmente $L = I_n$ y $U = A$.
2. Para cada paso $k = 1, 2, \ldots, n-1$:
   *a*) Calcular los multiplicadores
   $$l_{ik} = \frac{u_{ik}}{u_{kk}}, \quad i = k+1, \ldots, n. \tag{3.6}$$
   *b*) Restar $l_{ik}$ veces la fila $k$ a la fila $i$ de $U$.
   *c*) Almacenar cada $l_{ik}$ en la posición correspondiente de $L$.

De este modo se obtiene:

$$A = LU, \qquad L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ l_{n1} & \cdots & l_{n,n-1} & 1 & \end{bmatrix}, \qquad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix}. \tag{3.7}$$

**Nota 11.**
- *La factorización LU es el corazón de la eliminación gaussiana.*
- *Permite resolver sistemas lineales $Ax = b$ mediante:*
  $$Ax = b \quad \Rightarrow \quad LUx = b.$$
- *Se resuelve en dos etapas:*
  $$Ly = b, \qquad Ux = y,$$
  *usando sustitución progresiva y regresiva.*

**Nota 12.** *En los métodos directos como LU o Cholesky, al factorizar la matriz A en dos bloques (p.ej. $A = LU$ o $A = LL^\top$), la resolución de $Ax = b$ se divide en dos etapas fundamentales:*

**Nota 13** (Sustitución hacia adelante (forward substitution).)**.** *Se aplica cuando se resuelve un sistema triangular inferior:*

$$Ly = b, \qquad L = (\ell_{ij}) \text{ triangular inferior con } \ell_{ii} \neq 0. \tag{3.8}$$

*El proceso es:*

$$y_1 = \frac{b_1}{\ell_{11}}, \qquad y_i = \frac{1}{\ell_{ii}} \left( b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j \right), \quad i = 2, \ldots, n. \tag{3.9}$$

**Nota 14** (Sustitución hacia atrás (backward substitution).)**.** *Se aplica cuando se resuelve un sistema triangular superior:*

$$Ux = y, \qquad U = (u_{ij}) \text{ triangular superior con } u_{ii} \neq 0. \tag{3.10}$$

*El proceso es:*

$$x_n = \frac{y_n}{u_{nn}}, \qquad x_i = \frac{1}{u_{ii}} \left( y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right), \quad i = n-1, \ldots, 1. \tag{3.11}$$

**Ejemplo 6.** *Sea*

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}. \tag{3.12}$$

*La sustitución hacia atrás da:*

$$x_3 = \tfrac{y_3}{u_{33}}, \quad x_2 = \tfrac{y_2 - u_{23}x_3}{u_{22}}, \quad x_1 = \tfrac{y_1 - u_{12}x_2 - u_{13}x_3}{u_{11}}. \tag{3.13}$$

### 3.5.1. Método con permutaciones

Si en algún paso se cumple $u_{kk} = 0$, el algoritmo falla. Para evitarlo se aplica **pivoteo parcial**: se intercambia la fila $k$ con otra fila $p > k$ tal que

$$|u_{pk}| = \max_{i \geq k} |u_{ik}|. \tag{3.14}$$

Esto se representa mediante una matriz de permutación $P$, resultando:

$$PA = LU.$$

**Ejemplo 7.** *$A = LU$] Sea*

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 4 & 7 & 7 \\ -2 & 4 & 5 \end{bmatrix}.$$

*Aplicamos eliminación gaussiana guardando los multiplicadores en $L$ (con 1's en la diagonal).*

- *Paso $k = 1$. Pivote $u_{11} = 2$. Multiplicadores:*

$$\ell_{21} = \frac{4}{2} = 2, \qquad \ell_{31} = \frac{-2}{2} = -1.$$

*Actualizamos filas de $U$:*

$$R_2 \leftarrow R_2 - 2R_1, \quad R_3 \leftarrow R_3 - (-1)R_1 = R_3 + R_1.$$

*Esto produce*

$$U^{(1)} = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 5 \\ 0 & 7 & 6 \end{bmatrix}, \qquad L^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

- *Paso $k = 2$. Pivote $u_{22} = 1$. Multiplicador:*

$$\ell_{32} = \frac{7}{1} = 7.$$

*Actualizamos fila 3 de $U$:*

$$R_3 \leftarrow R_3 - 7R_2 \implies U = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -29 \end{bmatrix}.$$

*Insertamos $\ell_{32}$ en $L$:*

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 7 & 1 \end{bmatrix}.$$

- *Resultado. Se tiene*

$$\boxed{A = LU, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 7 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -29 \end{bmatrix}.}$$

*(Verificación rápida: $LU = A$ por multiplicación directa).*

**Ejemplo 8** ($PA = LU$)**.** *Sea*

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 1 & -2 & -3 \\ -1 & 1 & 2 \end{bmatrix}.$$

*El pivote inicial $a_{11} = 0$ es inválido; aplicamos* ***pivoteo parcial*** *intercambiando $R_1 \leftrightarrow R_2$. La matriz de permutación que permuta las primeras dos filas es*

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad PA = \begin{bmatrix} 1 & -2 & -3 \\ 0 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix}.$$

- *Paso $k = 1$ sobre $PA$. Pivote $u_{11} = 1$. Multiplicador:*

$$\ell_{31} = \frac{-1}{1} = -1.$$

*Actualizamos fila 3:*

$$R_3 \leftarrow R_3 - (-1)R_1 = R_3 + R_1 \implies U^{(1)} = \begin{bmatrix} 1 & -2 & -3 \\ 0 & 2 & 1 \\ 0 & -1 & -1 \end{bmatrix}, \quad L^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

- *Paso $k = 2$. Pivote $u_{22} = 2$. Multiplicador:*

$$\ell_{32} = \frac{-1}{2} = -\tfrac{1}{2}.$$

*Actualizamos fila 3:*

$$R_3 \leftarrow R_3 - (-\tfrac{1}{2})R_2 = R_3 + \tfrac{1}{2}R_2 \implies U = \begin{bmatrix} 1 & -2 & -3 \\ 0 & 2 & 1 \\ 0 & 0 & -\frac{1}{2} \end{bmatrix}.$$

*Insertamos $\ell_{32}$ en $L$:*

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -\frac{1}{2} & 1 \end{bmatrix}.$$

- *Resultado. Se obtiene*

$$\boxed{PA = LU, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -\frac{1}{2} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 1 & -2 & -3 \\ 0 & 2 & 1 \\ 0 & 0 & -\frac{1}{2} \end{bmatrix}.}$$

*(Comprobación: $LU = PA$ y, por tanto, $A = P^\top LU$ ya que $P^{-1} = P^\top$.)*

**Ejemplo 9.** *Resolver $Ax = b$ vía LU Consideremos*

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 4 & 7 & 7 \\ -2 & 4 & 5 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

*Ya hemos obtenido la factorización*

$$A = LU, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 7 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -29 \end{bmatrix}.$$

*El sistema $Ax = b$ se resuelve en dos etapas:*

**1. Resolver** $Ly = b$**.**

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 7 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

*De aquí:*

$$y_1 = 1, \quad 2y_1 + y_2 = 2 \;\Rightarrow\; y_2 = 0, \quad -1 \cdot y_1 + 7y_2 + y_3 = 3 \;\Rightarrow\; y_3 = 4.$$

*Por lo tanto,*

$$y = \begin{bmatrix} 1 \\ 0 \\ 4 \end{bmatrix}.$$

**2. Resolver** $Ux = y$**.**

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 1 & 5 \\ 0 & 0 & -29 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 4 \end{bmatrix}.$$

*De abajo hacia arriba:*

$$-29x_3 = 4 \;\Rightarrow\; x_3 = -\tfrac{4}{29},$$

$$x_2 + 5x_3 = 0 \;\Rightarrow\; x_2 = -5x_3 = \tfrac{20}{29},$$

$$2x_1 + 3x_2 + x_3 = 1 \;\Rightarrow\; 2x_1 + 3 \cdot \tfrac{20}{29} - \tfrac{4}{29} = 1.$$

$$2x_1 + \tfrac{60-4}{29} = 1 \;\Rightarrow\; 2x_1 + \tfrac{56}{29} = 1.$$

$$2x_1 = \tfrac{29}{29} - \tfrac{56}{29} = -\tfrac{27}{29} \;\Rightarrow\; x_1 = -\tfrac{27}{58}.$$

**Solución final:**

$$\boxed{x = \begin{bmatrix} -\frac{27}{58} \\ begineqnarray*6pt]\frac{20}{29} \\ begineqnarray*6pt] -\frac{4}{29} \end{bmatrix}.}$$

### 3.5.2. Factorización de Cholesky

Si $A$ es simétrica y definida positiva, entonces admite una factorización especial:

$$A = LL^\top,$$

donde $L$ es triangular inferior con entradas reales y diagonales positivas. Esta factorización es más eficiente y estable que la $LU$ en estos casos.

Sea $A \in \mathbb{R}^{n\times n}$ simétrica definida positiva. La factorización de Cholesky busca

$$A = LL^\top,$$

con $L$ triangular inferior y diagonal positiva. Las fórmulas recursivas son, para $i = 1, \ldots, n$:

$$\ell_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} \ell_{ik}^2}\,, \qquad \ell_{ji} = \frac{1}{\ell_{ii}}\Big(a_{ji} - \sum_{k=1}^{i-1} \ell_{jk}\,\ell_{ik}\Big) \quad (j = i+1, \ldots, n).$$

**Ejemplo 10.** *Considérese*

$$A = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}.$$

*Es simétrica y definida positiva (sus menores principales son positivos). Apliquemos las fórmulas:*

- *Columna* $i = 1$.

$$\ell_{11} = \sqrt{4} = 2, \qquad \ell_{21} = \frac{2}{2} = 1, \qquad \ell_{31} = \frac{2}{2} = 1.$$

- *Columna* $i = 2$.

$$\ell_{22} = \sqrt{2 - \ell_{21}^2} = \sqrt{2-1} = 1, \qquad \ell_{32} = \frac{1 - \ell_{31}\ell_{21}}{\ell_{22}} = \frac{1 - 1 \cdot 1}{1} = 0.$$

- *Columna* $i = 3$.

$$\ell_{33} = \sqrt{2 - \ell_{31}^2 - \ell_{32}^2} = \sqrt{2 - 1 - 0} = 1.$$

- *Resultado.*

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \qquad \boxed{A = LL^\top}.$$

*(Verificación rápida:* $LL^\top = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$*.)*

**Ejemplo 11.** *Considérese*

$$A = \begin{bmatrix} 9 & 3 & 6 \\ 3 & 5 & 3 \\ 6 & 3 & 14 \end{bmatrix}.$$

*También es simétrica definida positiva. Procedamos:*

- *Columna* $i = 1$.

$$\ell_{11} = \sqrt{9} = 3, \qquad \ell_{21} = \frac{3}{3} = 1, \qquad \ell_{31} = \frac{6}{3} = 2.$$

- *Columna* $i = 2$.

$$\ell_{22} = \sqrt{5 - \ell_{21}^2} = \sqrt{5-1} = 2, \qquad \ell_{32} = \frac{3 - \ell_{31}\ell_{21}}{\ell_{22}} = \frac{3 - 2 \cdot 1}{2} = \tfrac{1}{2}.$$

- *Columna* $i = 3$.

$$\ell_{33} = \sqrt{14 - \ell_{31}^2 - \ell_{32}^2} = \sqrt{14 - 4 - \tfrac{1}{4}} = \sqrt{\tfrac{39}{4}} = \frac{\sqrt{39}}{2}.$$

- *Resultado.*

$$L = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & \frac{1}{2} & \frac{\sqrt{39}}{2} \end{bmatrix}, \qquad \boxed{A = LL^\top}.$$

*(Verificación: $LL^\top$ reproduce $A$; la última diagonal verifica $2^2 + (\frac{1}{2})^2 + (\frac{\sqrt{39}}{2})^2 = 14$.)*

- **Forma:** $A = LL^\top$, con $L$ triangular inferior y diagonal positiva.
- **Requisitos:** $A$ debe ser **simétrica definida positiva (SDP)**.
- **Pivoteo:** No requiere pivoteo si $A$ es SDP (bien condicionada).
- **Ventajas:** Más rápida y estable (sin pivoteo) para matrices SDP; mejor aprovechamiento de memoria (simetría).
- **Cuándo usarla:** Sistemas simétricos definidos positivos (p.ej. matrices de Gram, problemas de mínimos cuadrados regulares, discretizaciones elípticas).

**Nota 15.** *Si $A$ es SDP, use Cholesky. En caso general, use $LU$ con pivoteo.*

### 3.5.3. Factorización de Doolittle

Es una variante de la factorización $LU$ en la cual la matriz $L$ tiene 1's en la diagonal principal:

$$A = LU, \quad L \text{ con unos en la diagonal, } U \text{ triangular superior.}$$

- **Forma:** $A = LU$, con $L$ triangular inferior con 1 en la diagonal y $U$ triangular superior.
- **Requisitos:** Válida para matrices cuadradas no singulares (en la práctica, puede requerir permutaciones: $PA = LU$).
- **Pivoteo:** Suele emplear *pivoteo parcial* para estabilidad numérica.
- **Cuándo usarla:** Sistemas generales (no necesariamente simétricos ni definidos positivos), múltiples RHS $b$ con la misma $A$.

**Nota 16** (Esquema general)**.** *Si $A = LL^\top$ con $L$ triangular inferior y diagonal positiva, resolver $Ax = b$ equivale a:*

$$Ly = b \quad \textit{(sustitución progresiva)}, \qquad L^\top x = y \quad \textit{(sustitución regresiva)}.$$

**Ejemplo 12.** *Sea*

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

*Su factorización de Cholesky es*

$$L = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \Longrightarrow \quad A = LL^{\top}.$$

- *Paso 1:* $Ly = b$.

$$\begin{bmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{aligned} 2y_1 = 1 &\Rightarrow y_1 = \tfrac{1}{2}, \\ y_1 + y_2 = 2 &\Rightarrow y_2 = \tfrac{3}{2}, \\ y_1 + y_3 = 3 &\Rightarrow y_3 = \tfrac{5}{2}. \end{aligned}$$

  *Por tanto,* $y = \left[\frac{1}{2},\ \frac{3}{2},\ \frac{5}{2}\right]^{\top}$.

- *Paso 2:* $L^{\top}x = y$.

$$L^{\top} = \begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix} \Rightarrow \begin{cases} x_3 = \frac{5}{2}, \\ x_2 = \frac{3}{2}, \\ 2x_1 + x_2 + x_3 = \frac{1}{2} \ \Rightarrow \ 2x_1 = \frac{1}{2} - \frac{3}{2} - \frac{5}{2} = -\frac{7}{2} \ \Rightarrow \ x_1 = -\frac{7}{4}. \end{cases}$$

$$\boxed{x = \begin{bmatrix} -\frac{7}{4} \\ \frac{3}{2} \\ \frac{5}{2} \end{bmatrix}}$$

**Ejemplo 13.** *Sea*

$$A = \begin{bmatrix} 9 & 3 & 6 \\ 3 & 5 & 3 \\ 6 & 3 & 14 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

*Una factorización de Cholesky es*

$$L = \begin{bmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & \frac{1}{2} & \frac{\sqrt{39}}{2} \end{bmatrix} \quad \Longrightarrow \quad A = LL^{\top}.$$

- *Paso 1:* $Ly = b$.

$$\begin{bmatrix} 3 & 0 & 0 \\ 1 & 2 & 0 \\ 2 & \frac{1}{2} & \frac{\sqrt{39}}{2} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$\begin{aligned} 3y_1 &= 1 \Rightarrow y_1 = \tfrac{1}{3}, \\ y_1 + 2y_2 &= 2 \Rightarrow y_2 = \tfrac{2-\frac{1}{3}}{2} = \tfrac{5}{6}, \\ 2y_1 + \tfrac{1}{2}y_2 + \tfrac{\sqrt{39}}{2}y_3 &= 3 \Rightarrow \tfrac{\sqrt{39}}{2}y_3 = 3 - \tfrac{2}{3} - \tfrac{5}{12} = \tfrac{23}{12} \\ y_3 &= \frac{23}{12} \cdot \frac{2}{\sqrt{39}} = \frac{23}{6\sqrt{39}}. \end{aligned}$$

*Por tanto,* $y = \left[\frac{1}{3},\ \frac{5}{6},\ \frac{23}{6\sqrt{39}}\right]^\top$.

- *Paso 2:* $L^\top x = y$.

$$L^\top = \begin{bmatrix} 3 & 1 & 2 \\ 0 & 2 & \frac{1}{2} \\ 0 & 0 & \frac{\sqrt{39}}{2} \end{bmatrix},$$

$$\begin{bmatrix} 3 & 1 & 2 \\ 0 & 2 & \frac{1}{2} \\ 0 & 0 & \frac{\sqrt{39}}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ \frac{5}{6} \\ \frac{23}{6\sqrt{39}} \end{bmatrix} \Rightarrow \begin{cases} \frac{\sqrt{39}}{2}\, x_3 = \frac{23}{6\sqrt{39}} \ \Rightarrow\ x_3 = \frac{23}{117}, \\ 2x_2 + \frac{1}{2}x_3 = \frac{5}{6} \ \Rightarrow\ 2x_2 = \frac{5}{6} - \frac{1}{2} \cdot \frac{23}{117} = \frac{86}{117} \ \Rightarrow\ x_2 = \frac{43}{117}, \\ 3x_1 + x_2 + 2x_3 = \frac{1}{3} = \frac{39}{117} \ \Rightarrow\ 3x_1 = \frac{39-43-46}{117} = -\frac{50}{117} \ \Rightarrow \\ x_1 = -\frac{50}{351}. \end{cases}$$

$$\boxed{x = \begin{bmatrix} -\frac{50}{351} \\ \frac{43}{117} \\ \frac{23}{117} \end{bmatrix}}$$

**Ejemplo 14.**

$$A = \begin{pmatrix} 2 & -1 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & -2 \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = L^T, \qquad U = \begin{pmatrix} 2 & -1 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & -2 \end{pmatrix}.$$

$$A = LU = I \cdot U = U, \quad y\ L = L^T.$$

**Ejemplo 15.**

$$\textit{Tomemos } L = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \quad \textit{(simétrica y triangular inferior, } L = L^T\textit{)},$$

$$U = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -3 \end{pmatrix} \textit{ (triangular superior).}$$

$$A = LU = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} 1 & -1 & 2 \\ 0 & 2 & 5 \\ 0 & 0 & -3 \end{pmatrix} = \begin{pmatrix} 2 & -2 & 4 \\ 0 & 6 & 15 \\ 0 & 0 & -12 \end{pmatrix}.$$

**Ejemplo 16.**

$$A = \begin{pmatrix} 2 & -1 & 1 & 3 \\ 4 & 1 & 0 & 2 \\ -2 & 2 & 5 & 1 \\ 6 & 0 & 2 & 4 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 1 & 1 & 0 \\ 3 & -1 & 0 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 2 & -1 & 1 & 3 \\ 0 & 3 & -2 & -4 \\ 0 & 0 & 6 & 3 \\ 0 & 0 & 0 & -2 \end{pmatrix}.$$

## 3.6. Descomposición de $A$

Sea $A = D + L + U$, con:

$$\begin{aligned} D &= \text{diag}(a_{11}, \ldots, a_{nn}), \\ L &= \text{parte estrictamente triangular inferior}, \\ U &= \text{parte estrictamente triangular superior}. \end{aligned}$$

De $Ax = b$ obtenemos:

$$x = -D^{-1}(L + U)x + D^{-1}b, \tag{3.15}$$

lo que sugiere la iteración:

$$x^{(k+1)} = Tx^{(k)} + c, \tag{3.16}$$

$$T = -D^{-1}(L + U), \tag{3.17}$$

$$c = D^{-1}b. \tag{3.18}$$

### 3.6.1. Método de Jacobi

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right). \tag{3.19}$$

Se calcula cada $x_i^{(k+1)}$ sólo con valores de la iteración anterior.

En forma matricial:

$$T_J = -D^{-1}(L+U), \quad c_J = D^{-1}b. \tag{3.20}$$

**Ejemplo 17.**

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2, \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \end{cases}$$

*Iteración:*

$$\begin{aligned} x_1^{(k+1)} &= \tfrac{1}{a_{11}}(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)}), \\ x_2^{(k+1)} &= \tfrac{1}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)}), \\ &\dots \end{aligned}$$

### 3.6.2. Método de Gauss–Seidel

Aprovecha los valores nuevos tan pronto como se calculan:

$$x^{(k+1)} = (D+L)^{-1}\big(-Ux^{(k)} + b\big). \tag{3.21}$$

Por componentes:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right). \tag{3.22}$$

**Nota 17** (Condiciones de convergencia)**.** *Ambos métodos convergen si el radio espectral de la matriz de iteración es menor que uno:*

$$\rho(T) < 1.$$

*Casos suficientes:*

- *A diagonalmente dominante* $\Rightarrow$ *Jacobi y Gauss–Seidel convergen.*
- *A simétrica definida positiva* $\Rightarrow$ *Gauss–Seidel converge.*
- *Convergencia* $\iff \rho(T) < 1$*, donde* $\rho$ *es el radio espectral.*
- *Suficiente: A diagonalmente dominante estricta* $\Rightarrow$ *Jacobi y GS convergen.*
- *Suficiente: A simétrica definida positiva* $\Rightarrow$ *GS siempre converge.*

### 3.6.3. Método SOR (Successive Over-Relaxation)

Generaliza Gauss–Seidel:

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

donde

- $\omega = 1$ reproduce Gauss–Seidel.
- $1 < \omega < 2$: *sobrerrelajación*, posible aceleración.
- $0 < \omega < 1$: *sub-relajación*, usada en algunos problemas mal condicionados.

Reescribiendo $Ax = b$:

$$\begin{aligned} x &= Tx + c, && (3.23)\\ T &= -D^{-1}(L+U), && (3.24)\\ c &= D^{-1}b. && (3.25) \end{aligned}$$

De aquí nace el esquema iterativo:

$$x^{(k+1)} = Tx^{(k)} + c. \tag{3.26}$$

**Nota 18.**

$$\begin{aligned} T_J &= -D^{-1}(L+U), c_J = D^{-1}b,\\ T_{GS} &= -(D+L)^{-1}U, c_{GS} = (D+L)^{-1}b,\\ T_{SOR} &= (D+\omega L)^{-1}\big((1-\omega)D - \omega U\big), c_{SOR} = \omega(D+\omega L)^{-1}b. \end{aligned}$$

### 3.6.4. Descomposición de la matriz

Sea $A \in \mathbb{R}^{n\times n}$. Se descompone como

$$A = D - L - U,$$

donde:

- $D$: matriz diagonal de $A$,
- $-L$: parte estrictamente triangular inferior,
- $-U$: parte estrictamente triangular superior.

Así,

$$A = D + L + U, \quad Ax = b \iff (D+L+U)x = b.$$

### 3.6.5. Método de Jacobi

A partir de la descomposición, se obtiene el esquema iterativo de Jacobi:

$$x^{(k+1)} = D^{-1}\big(b - (L+U)x^{(k)}\big).$$

De manera componente a componente:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1,\ldots,n.$$

- Cada componente $x_i^{(k+1)}$ se calcula usando exclusivamente valores de la iteración previa $x^{(k)}$.
- Es fácil de implementar en paralelo.

### 3.6.6. Método de Gauss–Seidel

En Gauss–Seidel se aprovecha que, en cada paso, los nuevos valores calculados pueden usarse de inmediato:

$$x^{(k+1)} = (D+L)^{-1}\big(b - Ux^{(k)}\big).$$

En forma componente:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right).$$

- La diferencia clave con Jacobi: $x_j^{(k+1)}$ ya calculados se usan de inmediato.
- Suele converger más rápido que Jacobi.

### 3.6.7. Condiciones de convergencia

Un resultado clásico:

- Si $A$ es **diagonal dominante estricta** (es decir, $|a_{ii}| > \sum_{j\neq i}|a_{ij}|$ para todo $i$), entonces Jacobi y Gauss–Seidel convergen.
- Si $A$ es **simétrica definida positiva**, Gauss–Seidel siempre converge.
- En general, la convergencia depende de que el radio espectral de la matriz de iteración sea menor que 1:

$$\rho(T) < 1.$$

**Nota 19** (Esquema general)**.** *Se procede de la siguiente manera:*

1. *Elegir un vector inicial $x^{(0)}$ (p.ej. el vector nulo).*
2. *Repetir hasta convergencia:*
   a) *Calcular $x^{(k+1)}$ según el método elegido (Jacobi o Gauss–Seidel).*
   b) *Medir el error, p.ej. $\|x^{(k+1)} - x^{(k)}\|$.*
   c) *Detenerse cuando el error sea menor que una tolerancia $\varepsilon$.*

**Ejemplo 18.** *Sistema de prueba (diagonalmente dominante) Consideremos*

$$A = \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix},$$

*con vector inicial $x^{(0)} = \mathbf{0}$. Este sistema es* diagonalmente dominante*, por lo que Jacobi y Gauss–Seidel convergen.*

*La solución exacta es*

$$x^* = \begin{bmatrix} 1 \\ 2 \\ -1 \\ 1 \end{bmatrix}.$$

**Ejemplo 19** (Método de Jacobi)**.** *Recordemos que*

$$x^{(k+1)} = D^{-1}\big(b - (L+U)\,x^{(k)}\big), \qquad x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j\neq i} a_{ij}\,x_j^{(k)}\right).$$

$$x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad x^{(1)} = \begin{bmatrix} 0.6 \\ 2.272727 \\ -1.100000 \\ 1.875000 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 1.047273 \\ 1.715909 \\ -0.805227 \\ 0.885227 \end{bmatrix}, \quad x^{(3)} = \begin{bmatrix} 0.932636 \\ 2.053306 \\ -1.049341 \\ 1.130881 \end{bmatrix}.$$

*Errores (norma infinito)*

$$\begin{aligned} \|x^{(0)} - x^*\|_\infty &= 2.000000, \\ \|x^{(1)} - x^*\|_\infty &= 0.875000, \\ \|x^{(2)} - x^*\|_\infty &= 0.284091, \\ \|x^{(3)} - x^*\|_\infty &= 0.130881. \end{aligned}$$

**Ejemplo 20** (Método de Gauss–Seidel)**.**

$$\begin{aligned} x^{(k+1)} &= (D+L)^{-1}\big(b - U\,x^{(k)}\big), \\ x_i^{(k+1)} &= \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\,x_j^{(k+1)} - \sum_{j>i} a_{ij}\,x_j^{(k)}\right). \end{aligned}$$

*Iteraciones (redondeadas a 6 decimales).*

$$x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad x^{(1)} = \begin{bmatrix} 0.6 \\ 2.327273 \\ -0.987273 \\ 0.878864 \end{bmatrix}, \quad x^{(2)} = \begin{bmatrix} 1.030182 \\ 2.036938 \\ -1.014456 \\ 0.984341 \end{bmatrix}, \quad x^{(3)} = \begin{bmatrix} 1.006585 \\ 2.003555 \\ -1.002527 \\ 0.998351 \end{bmatrix}.$$

*Errores (norma infinito) frente a $x^*$.*

$$\begin{aligned} \|x^{(0)} - x^*\|_\infty &= 2.000000, \\ \|x^{(1)} - x^*\|_\infty &= 0.400000, \\ \|x^{(2)} - x^*\|_\infty &= 0.036938, \\ \|x^{(3)} - x^*\|_\infty &= 0.006585. \end{aligned}$$

**Nota 20.**

- *Ambos métodos convergen (la matriz es diagonalmente dominante), pero **Gauss–Seidel** reduce el error notablemente más rápido al reutilizar los valores nuevos dentro de la misma iteración.*
- *Criterio de paro típico: detener cuando $\|x^{(k+1)} - x^{(k)}\| \le \varepsilon$ o $\|Ax^{(k)} - b\| \le \varepsilon$.*
- *Para paralelización masiva, **Jacobi** es más simple; para rapidez en CPU única, **Gauss–Seidel** suele ser preferible.*

**Ejemplo 21** (SOR en una sola iteración (comparación con Gauss–Seidel))**.** *Consideremos el mismo sistema diagonalmente dominante:*

$$A = \begin{bmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 25 \\ -11 \\ 15 \end{bmatrix}, \qquad x^{(0)} = \mathbf{0}.$$

*con $\omega = 1.2$*

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

*Una iteración desde $x^{(0)} = \mathbf{0}$ (redondeado a 6 decimales).*

$$x_{SOR}^{(1)} = \begin{bmatrix} 0.720000 \\ 2.805818 \\ -1.156102 \\ 0.813967 \end{bmatrix}.$$

*Gauss–Seidel, una iteración*

$$x_{GS}^{(1)} = \begin{bmatrix} 0.600000 \\ 2.327273 \\ -0.987273 \\ 0.878864 \end{bmatrix}.$$

- *Con $\omega > 1$ (sobrerrelajación), SOR puede* acelerar *la convergencia, pero la* ***primera*** *iteración puede mostrar un sobre-disparo respecto a la solución exacta.*
- *La elección de $\omega$ es clave: en la práctica se prueba $\omega \in [1.1, 1.5]$ y se observa el comportamiento; si $\omega$ es muy grande, puede empeorar o incluso hacer divergir el método.*
- *Si A es SPD, métodos como* ***Cholesky*** *(directo) o iterativos tipo* ***Gradiente Conjugado*** *suelen ser preferibles por estabilidad y rapidez.*

**Nota 21** (Esquema general)**.** *Dado $Ax = b$ y un vector inicial $x^{(0)}$, el método SOR genera la sucesión:*

$$x_i^{(k+1)} = (1-\omega)\, x_i^{(k)} + \frac{\omega}{a_{ii}} \left( b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right), \quad i = 1, \ldots, n.$$

- *Si $\omega = 1$, se recupera exactamente el método de Gauss–Seidel.*
- *Si $0 < \omega < 1$, se llama* relajación sub-sucessiva*, útil para estabilizar ciertos casos.*
- *Si $1 < \omega < 2$, se llama* sobrerrelajación*, y puede acelerar notablemente la convergencia.*
- *La elección óptima de $\omega$ depende de la matriz A; en la práctica, se prueba con valores como $\omega \approx 1.1$ a 1.5.*
- *El método es útil en problemas grandes, como los provenientes de discretización de ecuaciones diferenciales.*
- *Jacobi: usa valores viejos.*
- *Gauss–Seidel: usa valores nuevos en cuanto están disponibles.*
- *SOR: Gauss–Seidel + parámetro $\omega$ que acelera la convergencia si se elige bien.*

## 3.7. Ejercicios SEL

**Ejercicio 3.** *Resolver por eliminación Gaussiana Simple los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} x_1 - 2x_2 + 0.5x_3 &= -5 \\ -2x_1 + 5x_2 - 1.5x_3 &= 0 \\ -0.2x_1 + 1.75x_2 - x_3 &= 10 \end{aligned}$$

*2.*

$$\begin{aligned} 3x_1 - x_2 + 6x_4 &= 2.3 \\ 4x_1 + 2x_2 - x_3 - 5x_4 &= 6.9 \\ -5x_1 + x_2 - 3x_3 &= -36 \\ 10x_2 - 4x_3 + 7x_4 &= -36 \end{aligned}$$

*3.*

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

*utilizar redondeo a 4 cifras significativas.*

*4.*

$$\begin{aligned} 4x_1 + 2x_2 &= 2 \\ 2x_1 + 3x_2 + x_3 &= -1 \\ x_2 + \frac{5}{2}x_3 &= 3 \end{aligned}$$

*5.*

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*6.*

$$\begin{aligned} 8x_1 + 2x_2 - 2x_3 &= -2 \\ 10x_1 + 2x_2 + 4x_3 &= 4 \\ 12x_1 + 2x_2 + 2x_3 &= 6 \end{aligned}$$

*7.*

$$\begin{aligned} 5x_1 + 2x_2 + x_3 - x_4 &= 1 \\ 2x_1 - x_2 + 3x_3 + 2x_4 &= 12 \\ 4x_1 + x_2 - 2x_3 + 3x_4 &= 5 \\ -2x_1 + 2x_2 + x_3 + x_4 &= 2 \end{aligned}$$

*8.*

$$\begin{aligned} 2x_1 + 3x_2 + 2x_3 + 4x_4 &= 4 \\ 4x_1 + 10x_2 - 4x_3 &= -8 \\ -3x_1 - 2x_2 - 5x_3 - 2x_4 &= -4 \\ -2x_1 + 4x_2 + 4x_3 - 7x_4 &= -1 \end{aligned}$$

*9.*

$$\begin{aligned} 1.133x_1 + 5.281x_2 - 2.454x_3 &= 6.414 \\ 24.14x_1 - 1.21x_2 + 5.281x_3 &= 113.8 \\ -10.123x_1 + 6.387x_2 - x_3 &= 1 \end{aligned}$$

*10.* $A = \begin{pmatrix} 2 & 3 & 2 & 4 \\ 4 & 10 & -4 & 0 \\ -3 & -2 & -5 & -2 \\ -2 & 4 & 4 & -7 \end{pmatrix}$ *y* $b = \begin{pmatrix} 9 \\ -15 \\ 6 \\ 2 \end{pmatrix}$

**Ejercicio 4.** *Resolver por eliminación gaussiana con pivoteo parcial los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} 0.4x_1 - 1.5x_2 + 0.75x_3 &= -20 \\ -0.5x_1 - 15x_2 + 10x_3 &= -10 \\ -10x_1 - 9x_2 + 2.5x_3 &= 30 \end{aligned}$$

*2.*

$$\begin{aligned} 5x_1 - 8x_2 + x_3 &= -71 \\ -2x_1 + 6x_2 - 9x_3 &= 134 \\ 3x_1 - 5x_2 + 2x_3 &= -58 \end{aligned}$$

*3.*

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

*utilizar redondeo a 4 cifras significativas.*

*4.*

$$\begin{aligned} -x_2 + 4x_3 - x_4 &= -1 \\ -x_1 + 4x_2 - x_3 &= 2 \\ -x_1 - x_3 + 4x_4 &= 4 \\ 4x_1 - x_2 &= 10 \end{aligned}$$

*5.*

$$\begin{aligned} 0.00031000x_1 + 1.000000x_2 &= 3.000000 \\ 1.00045534x_1 + 1.00034333x_2 &= 7.000 \end{aligned}$$

6. *Resolver para* $A = \begin{pmatrix} 14 & 14 & -9 & 3 & -5 \\ 14 & 52 & -15 & 2 & -32 \\ -9 & -15 & 36 & -5 & 16 \\ 3 & 2 & -5 & 47 & 49 \\ -5 & 32 & 16 & 49 & 79 \end{pmatrix}$, $b = \begin{pmatrix} -15 \\ -100 \\ 106 \\ 329 \\ 463 \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$

7. *Resolver para* $A = \begin{pmatrix} \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & 1 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 9 \\ 8 \\ 8 \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$

8. *Resolver para* $A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$, $b = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{7} \\ \frac{1}{8} \\ \frac{1}{9} \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$

9. *Resolver el sistema*

$$\begin{aligned} 2x_1 + x_2 - x_3 + x_4 - 3x_5 &= 7 \\ x_1 + 2x_3 - x_4 + x_5 &= 2 \\ -2x_2 - x_3 + x_4 - x_5 &= -5 \\ 3x_1 + x_2 - 4x_3 + 5x_5 &= 6 \\ x_1 - x_2 - x_3 - x_4 + x_5 &= 3 \end{aligned}$$

10. *Resolver el sistema*

$$\begin{aligned} 3.333x_1 + 15920x_2 - 10.333x_3 &= 15913 \\ 2.222x_1 + 16.71x_2 9.612x_2 &= 28.544 \\ 1.5611x_1 + 5.1791x_2 + 1.6852x_3 &= 8.4254 \end{aligned}$$

**Ejercicio 5.** *Resolver por el método de Gauss-Jordan los siguientes sistemas de ecuaciones lineales*

1.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ -0.4x_1 + 2x_2 - x_3 &= 10 \\ 0.5x_1 - 3x_2 + x_3 &= 15 \end{aligned}$$

2.

$$\begin{aligned} 2x_1 - 0.9x_2 + 3x_3 &= -3.61 \\ -0.5x_1 + 0.1x_2 - x_3 &= 2.035 \\ x_1 - 6.35x_2 - 0.45x_3 &= 15.401 \end{aligned}$$

*3.*

$$\begin{aligned} 0.7x_1 + 2.7x_2 - 6x_3 + 0.7x_4 &= 1.6487 \\ 2x_1 - 0.8x_2 + 3x_3 - x_4 &= -2.342 \\ -x_1 - 1.5x_2 + 1.4x_3 + 3x_4 &= -4.189 \\ 7x_2 - 1.56x_3 + x_4 &= 15.792 \end{aligned}$$

*4.*

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*5.*

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 27 \\ -3x_1 - 6x_2 + 2x3 &= -61.5 \\ x_1 + x_2 + 5x_3 &= -21.5 \end{aligned}$$

*6.* $A = \begin{pmatrix} 1 & 3 & -2 & 1 \\ 1 & 3 & -1 & 2 \\ 0 & 1 & -1 & 4 \\ 2 & 6 & 1 & 2 \end{pmatrix}$ *y* $b = \begin{pmatrix} 4 \\ 1 \\ 5 \\ 2 \end{pmatrix}$

*7.*

$$\begin{aligned} 6x_1 - x_2 - x_3 + 4x_4 &= 17 \\ x_1 - 10x_2 + 2x_3 - x_4 &= -17 \\ 3x_1 - 2x_2 + 8x_3 - x_4 &= 19 \\ x_1 + x_2 + x_3 - 5x_4 &= -14 \end{aligned}$$

*8.*

$$\begin{aligned} x + 2y + 3z + 4w &= 1 \\ x - 4y + z + 11w &= 2 \\ -x + 8y + 7z + 6w &= -2 \\ 16x + 8y - 5z + 6w &= 11 \end{aligned}$$

*9.*

$$\begin{aligned} x_1 + x_2 &= 3 \\ x_1 + 2x_2 + x_3 &= -1 \\ x_2 + 3x_3 + x_4 &= 2 \\ x_3 + 4x_4 + x_5 &= 1 \\ x_4 + 5x_5 &= 3 \end{aligned}$$

*10.*

$$\begin{aligned} 15x_1 - 18x_2 + 15x_3 - 3x_4 &= 11 \\ -18x_1 + 24_2 - 18x_3 + 4x_4 &= 10 \\ 15x_1 - 18x_2 + 18x_3 - 3x_4 &= 11 \\ -3x_1 + 4x_2 - 3x_3 + x_4 &= 13 \end{aligned}$$

**Ejercicio 6.** *Resolver por el método de Gauss-Seidel los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} 3x_1 - 0.2x_2 - 0.5x_3 &= 8 \\ 0.1x_1 + 7x_2 + 0.4x_3 &= -19.5 \\ 0.4x_1 - 0.1x_2 + 10x_3 &= 72.4 \end{aligned}$$

*2.*

$$\begin{aligned} -5x_1 + 1.4x_2 - 2.7x_3 &= 94.2 \\ 0.7x_1 - 2.5x_2 + 15x_3 &= -6 \\ 3.3x_1 - 11x_2 + 4.4x_3 &= -27.5 \end{aligned}$$

*3.*

$$\begin{aligned} 3x_1 - 0.5x_2 + 0.6x_3 &= 5.24 \\ 0.3x_1 - 4x_2 - x_3 &= -0.387 \\ -0.7x_1 + 2x_2 + 7x_3 &= 14.803 \end{aligned}$$

*4.*

$$\begin{aligned} 5x_1 - 0.2x_2 + x_3 &= 1.5 \\ 0.1x_1 + 3x_2 - 0.5x_3 &= -2.7 \\ -0.3x_1 + x_2 - 7x_3 &= 9.5 \end{aligned}$$

*5.*

$$\begin{aligned} -3x_2 + 7x_3 &= 2 \\ x_1 + 2x_2 - x_3 &= 3 \\ 12x_1 + 2x_2 + 2x_3 &= 6 \end{aligned}$$

*6.*

$$\begin{aligned} 0.15_1 + 2.11x_2 + 30.75x_3 &= -26.38 \\ 0.64x_1 + 1.21x_2 + 2.05x_3 &= 1.01 \\ 3.21x_1 + 1.53x_2 + 1.04x_3 &= 5.23 \end{aligned}$$

*7.*

$$\begin{aligned} x_1 + x_2 - x_3 &= -3 \\ 6x_1 + 2x_2 + 2x_3 &= 2 \\ -3x_1 + 4x_2 + x_3 &= 1 \end{aligned}$$

*8.*

$$\begin{aligned} 2x_1 + x_2 - x3 &= 1 \\ 5x_1 + 2x_2 + 2x_3 &= -4 \\ 3x_1 + x_2 + x_3 &= 5 \end{aligned}$$

*9.*

$$\begin{aligned} 3x - 0.1y - 0.2z &= 7.85 \\ 0.1x + 7y - 0.3z &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*10.*

$$\begin{aligned} 17x_1 - 2x_2 - 3x_3 &= 500 \\ -5x_1 + 21x_2 - 2x_3 &= 200 \\ -5x_1 - 5x_2 + 22x_3 &= 30 \end{aligned}$$

**Ejercicio 7.** *Aplicar el método de Jacobi para resolver los siguientes sistemas de ecuaciones lineales*

*1.* $A = \left(\begin{array}{cccc|c} 10 & 2 & -1 & 0 & 26 \\ 1 & 20 & -2 & 3 & -15 \\ -2 & 1 & 30 & 0 & 53 \\ 1 & 2 & 3 & 20 & 47 \end{array}\right)$

*2.* $A = \left(\begin{array}{ccc|c} -1 & 2 & 10 & 11 \\ 11 & -1 & 2 & 12 \\ 1 & 5 & 2 & 8 \end{array}\right)$

*3.* $A = \left(\begin{array}{ccc|c} 8 & 2 & 3 & 51 \\ 2 & 5 & 1 & 23 \\ -3 & 1 & 6 & 20 \end{array}\right)$

*4.* $A = \left(\begin{array}{cccc|c} 2 & -1 & 1 & 3 & 10 \\ 2 & 2 & 2 & 2 & 1 \\ -1 & -1 & 2 & 2 & -5 \\ 3 & 1 & -1 & 4 & 6 \end{array}\right)$

5. $A = \left(\begin{array}{cccc|c} 3 & 1 & 1 & -1 & 5 \\ 0 & 2 & 1 & 4 & 0 \\ 1 & 1 & -1 & 9 & 1 \\ 2 & 4 & 6 & 3 & 0 \end{array}\right)$

6. $A = \left(\begin{array}{cccc|c} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 2 & -1 & 8 & 15 \end{array}\right)$

7.

$$\begin{aligned} x_1 + 2x_2 - 2x_3 &= 7 \\ x_1 + x_2 + x_3 &= 2 \\ 2x_1 + 2x_2 + x_3 &= 5 \end{aligned}$$

8.

$$\begin{aligned} -4x_1 + 14x_2 &= 10 \\ -5x_1 + 13x_2 &= 8 \\ -x_1 + 2x_3 &= 1 \end{aligned}$$

9.

$$\begin{aligned} x + y + 2z &= 1 \\ x + 2y + z &= 1 \\ 2x + y + z &= 1 \end{aligned}$$

10.

$$\begin{aligned} 6x_1 - 2x_2 + 2x_3 + 4x_4 &= 10 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 &= 20 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 &= 2 \\ -6x_1 + 4x_2 + x_3 - 18x_4 &= -19 \end{aligned}$$

**Ejercicio 8.** *1. Resuelva el sistema*

$$\begin{cases} 2x + y - z = 1, \\ -x + 3y + 2z = 12, \\ x + 2y + 3z = 7 \end{cases}$$

*mediante eliminación gaussiana simple (sin pivoteo).*

2. *Resuelva el mismo sistema anterior pero ahora aplicando eliminación gaussiana con pivoteo parcial. Compare los pasos con el ejercicio anterior.*

3. *Aplique eliminación gaussiana con pivoteo y escalamiento al sistema*

$$\begin{cases} 10x + 2y + z = 7, \\ 2x + 20y + 2z = 9, \\ x + 2y + 30z = 12 \end{cases}$$

*y analice la importancia del escalamiento.*

4. *Utilice el método de Gauss–Jordan para calcular la inversa de*

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

5. *Resuelva $Ax = b$ con $A$ y $b$ dados por*

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

*utilizando factorización $LU$ y sustitución hacia adelante y hacia atrás.*

6. *Calcule la factorización de Cholesky de*

$$A = \begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix}$$

*y resuelva $Ax = b$ con $b = (35, 33, 6)^\top$.*

7. *Resuelva mediante sustitución hacia atrás el sistema triangular superior:*

$$\begin{cases} 2x + 3y - z = 5, \\ -y + 2z = 4, \\ 3z = 6. \end{cases}$$

8. *Resuelva mediante sustitución hacia adelante el sistema triangular inferior:*

$$\begin{cases} x = 3, \\ 2y + x = 5, \\ z - y + 2x = 10. \end{cases}$$

10. *Aplique el método de Jacobi para resolver*

$$\begin{cases} 10x - y + 2z = 6, \\ -x + 11y - z + 3w = 25, \\ 2x - y + 10z - w = -11, \\ 3y - z + 8w = 15, \end{cases}$$

*realizando 3 iteraciones con* $x^{(0)} = \mathbf{0}$.

11. *Repita el ejercicio anterior con el método de Gauss–Seidel. Compare la velocidad de convergencia con Jacobi.*

12. *Aplique el método SOR con* $\omega = 1.25$ *al mismo sistema y compare las tres trayectorias de convergencia.*

13. *Escriba la matriz de iteración* $T_J$ *y el vector* $c_J$ *del método de Jacobi para el sistema*

$$\begin{cases} 4x + y = 9, \\ x + 3y = 7. \end{cases}$$

*Verifique si* $\rho(T_J) < 1$.

14. *Investigue experimentalmente en R cuál es el valor óptimo aproximado de* $\omega$ *para el método SOR en el sistema* $4 \times 4$ *del ejercicio 10.*

15. *Genere una matriz aleatoria simétrica definida positiva* $5 \times 5$ *en R y resuelva* $Ax = b$*:*

    (a) *Usando factorización LU.*

    (b) *Usando factorización de Cholesky.*

    (c) *Usando Gauss–Seidel con 20 iteraciones.*

    *Compare el tiempo de cómputo y la precisión de cada método.*

# Capítulo 4

# Raíces de Funciones

## 4.1. Método de Bisección

Dada una función continua $f : [a, b] \to \mathbb{R}$ con $f(a)\, f(b) < 0$, por el **Teorema de Bolzano** existe al menos una raíz en $(a, b)$. El **método de bisección** explota este cambio de signo: divide el intervalo a la mitad, selecciona la submitad donde persiste el cambio de signo, y repite. Es un método *robusto* (garantiza convergencia si se mantienen las hipótesis) y con *tasa lineal*.

**Algoritmo 2.** *Para encontrar la raíz de una función utilizando el método de la bisección es*

1. *Verificar continuidad de $f$ en $[a, b]$ (al menos de forma razonable) y que $f(a)\, f(b) < 0$.*

2. *Repetir para $n = 1, 2, \ldots$:*

   a) *$c = \dfrac{a+b}{2}$, evaluar $f(c)$.*

   b) *Si $f(c) = 0$ (o $|f(c)|$ pequeño), **parar**: $c$ es la raíz aproximada.*

   c) *Si $f(a)\, f(c) < 0$, poner $b \leftarrow c$; en caso contrario, $a \leftarrow c$.*

   d) *Criterio de paro: ancho $(b-a)/2 < \varepsilon$ o $|f(c)| < \varepsilon$, o alcanzar $n_{\text{máx}}$.*

**Cota del error**: Si $(a_0, b_0)$ es el intervalo inicial, tras $n$ iteraciones el punto medio $c_n$ satisface

$$|c_n - \alpha| \leq \frac{b_0 - a_0}{2^n}, \tag{4.1}$$

donde $\alpha$ es alguna raíz en $(a_0, b_0)$. Para asegurar $(b_n - a_n)/2 < \varepsilon$, basta con

$$n \geq \left\lceil \log_2\left(\frac{b_0 - a_0}{\varepsilon}\right) \right\rceil. \tag{4.2}$$

**Resultado 1.** *Si $f(x)$ es continua en $[a,b]$ y $f(a)\,f(b)<0$, por el* ***Teorema de Bolzano*** *existe al menos una raíz en $(a,b)$. El método de bisección consiste en:*

$$c_k=\frac{a_k+b_k}{2},\qquad \textit{y se actualiza el intervalo de búsqueda } \begin{cases} b_{k+1}=c_k & \textit{si } f(a_k)f(c_k)<0,\\ a_{k+1}=c_k & \textit{si } f(a_k)f(c_k)>0.\end{cases} \tag{4.3}$$

*La cota del error después de n iteraciones es*

$$|c_n-\alpha|\leq\frac{b_0-a_0}{2^n}. \tag{4.4}$$

**Ejemplo 22.**
```
# ==========================================
# M\'ETODO DE BISECCI\'oN PARA UNA C\'UBICA GENERAL
# ==========================================
# ---- Coeficientes del polinomio c\'ubico ----
A <- 1     # coeficiente de x^3
B <- -6    # coeficiente de x^2
C <- 11    # coeficiente de x
D <- -6    # t\'ermino independiente

# ---- Definici\'on de la funci\'on c\'ubica ----
f <- function(x) {
  A*x^3 + B*x^2 + C*x + D
}

# ---- Intervalo inicial ----
a1 <- 0
b1 <- 2.5   # aseg\'urate que f(a1)*f(b1) < 0

# ==== Iteraci\'on 1 ====
c1  <- (a1 + b1)/2
fa1 <- f(a1)
fb1 <- f(b1)
fc1 <- f(c1)
s1a <- sign(fa1 * fc1)
s1b <- sign(fb1 * fc1)
e1  <- (b1 - a1)/2
a2 <- if (fa1 * fc1 < 0) a1 else c1
b2 <- if (fa1 * fc1 < 0) c1 else b1

# ==== Iteraci\'on 2 ====
c2  <- (a2 + b2)/2
fa2 <- f(a2)
fb2 <- f(b2)
fc2 <- f(c2)
s2a <- sign(fa2 * fc2)
s2b <- sign(fb2 * fc2)
```

```
e2  <- (b2 - a2)/2
a3 <- if (fa2 * fc2 < 0) a2 else c2
b3 <- if (fa2 * fc2 < 0) c2 else b2

# ==== Iteraci\'on 3 ====
c3  <- (a3 + b3)/2
fa3 <- f(a3)
fb3 <- f(b3)
fc3 <- f(c3)
s3a <- sign(fa3 * fc3)
s3b <- sign(fb3 * fc3)
e3  <- (b3 - a3)/2
a4 <- if (fa3 * fc3 < 0) a3 else c3
b4 <- if (fa3 * fc3 < 0) c3 else b3

# ==== Iteraci\'on 4 ====
c4  <- (a4 + b4)/2
fa4 <- f(a4)
fb4 <- f(b4)
fc4 <- f(c4)
s4a <- sign(fa4 * fc4)
s4b <- sign(fb4 * fc4)
e4  <- (b4 - a4)/2
a5 <- if (fa4 * fc4 < 0) a4 else c4
b5 <- if (fa4 * fc4 < 0) c4 else b4

# ==== Iteraci\'on 5 ====
c5  <- (a5 + b5)/2
fa5 <- f(a5)
fb5 <- f(b5)
fc5 <- f(c5)
s5a <- sign(fa5 * fc5)
s5b <- sign(fb5 * fc5)
e5  <- (b5 - a5)/2
a6 <- if (fa5 * fc5 < 0) a5 else c5
b6 <- if (fa5 * fc5 < 0) c5 else b5

# ==== Iteraci\'on 6 ====
c6  <- (a6 + b6)/2
fa6 <- f(a6)
fb6 <- f(b6)
fc6 <- f(c6)
s6a <- sign(fa6 * fc6)
s6b <- sign(fb6 * fc6)
e6  <- (b6 - a6)/2
```

```
# ---- Aproximaci\'on final ----
raiz_aprox <- c6
error_cota <- e6

cat("Ra\'iz aproximada (6 iteraciones):", raiz_aprox, "\n")
cat("Cota m\'axima del error:", error_cota, "\n\n")

# ---- Tabla resumen ----
iter <- 1:6
Acol <- c(a1,a2,a3,a4,a5,a6)
Bcol <- c(b1,b2,b3,b4,b5,b6)
Ccol <- c(c1,c2,c3,c4,c5,c6)
FA   <- c(fa1,fa2,fa3,fa4,fa5,fa6)
FB   <- c(fb1,fb2,fb3,fb4,fb5,fb6)
FC   <- c(fc1,fc2,fc3,fc4,fc5,fc6)
signFAFC <- c(s1a,s2a,s3a,s4a,s5a,s6a)
signFBFC <- c(s1b,s2b,s3b,s4b,s5b,s6b)
ERR  <- c(e1,e2,e3,e4,e5,e6)

TAB <- data.frame(iter, a=Acol, b=Bcol, c=Ccol,
                  fa=FA, fb=FB, fc=FC,
                  "sign(fa*fc)"=signFAFC,
                  "sign(fb*fc)"=signFBFC,
                  error=ERR)

print(round(TAB, 6))
```

### 4.1.1. Ejemplos

**Ejemplo 23.** *Consideremos la función* $f(x) = x^2 - 5x + 6$. *para* $a = 1, b = -5, c = 6$.

- *Verificar cambio de signo. Evaluamos:* $f(1) = 1 - 5 + 6 = 2 > 0, \quad f(2.5) = 6.25 - 12.5 + 6 = -0.25 < 0$. *Existe cambio de signo entre* $x = 1$ *y* $x = 2.5$, *por lo tanto, hay al menos una raíz en* $[1, 2.5]$.
- *El punto medio inicial es:* $c_1 = \frac{1+2.5}{2} = 1.75, \quad f(1.75) = 3,0625 - 8,75 + 6 = 0,3125 > 0$. *Como* $f(1.75) > 0$ *y* $f(2.5) < 0$, *el nuevo intervalo es* $[1.75, 2.5]$.
- *Segunda iteración:* $c_2 = \frac{1.75+2.5}{2} = 2.125, \quad f(2.125) = 4,5156 - 10,625 + 6 = -0.1094 < 0$. *Nuevo intervalo:* $[1.75, 2.125]$.
- *Tercera iteración:* $c_3 = \frac{1.75+2.125}{2} = 1.9375, \quad f(1.9375) = 3,754 - 9,6875 + 6 = 0.0665 > 0$. *Nuevo intervalo:* $[1.9375, 2.125]$.
- *Cuarta iteración:*$c_4 = \frac{1.9375+2.125}{2} = 2.03125, \quad f(2.03125) = 4,126 - 10,156 + 6 = -0.030 < 0$. *Nuevo intervalo:* $[1.9375, 2.03125]$.

- *Después de unas pocas iteraciones, la raíz se aproxima a* $x \approx 2.0$.

| *Iteración* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.000* | *2.500* | *1.750* | $+0.3125$ |
| *2* | *1.750* | *2.500* | *2.125* | $-0.1094$ |
| *3* | *1.750* | *2.125* | *1.9375* | $+0.0665$ |
| *4* | *1.9375* | *2.125* | *2.0313* | $-0.030$ |
| *5* | *1.9375* | *2.0313* | *1.9844* | $+0.017$ |

**Ejemplo 24.** *Sea* $f(x) = x^2 - 2$ *con* $a = 1$, $b = -2$. *Entonces* $f(1) = -1 < 0$, $f(2) = 2 > 0$ $\Rightarrow$ *hay raíz en* $[1, 2]$.

| *Iter.* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.000000* | *2.000000* | *1.500000* | $+0.250000$ |
| *2* | *1.000000* | *1.500000* | *1.250000* | $-0.437500$ |
| *3* | *1.250000* | *1.500000* | *1.375000* | $-0.109375$ |
| *4* | *1.375000* | *1.500000* | *1.437500* | $+0.066406$ |
| *5* | *1.375000* | *1.437500* | *1.406250* | $-0.022461$ |
| *6* | *1.406250* | *1.437500* | *1.421875* | $+0.021729$ |
| *7* | *1.406250* | *1.421875* | *1.414063* | $-0.000427$ |
| *8* | *1.414063* | *1.421875* | *1.417969* | $+0.010635$ |

**Ejemplo 25.** *Sea la función* $f(x) = x^3 - 6x^2 + 11x - 6$, *con* $a = 1, b = -6, c = 11, d = -6$, *en el intervalo* $[1.5, 2.5]$.

- *Verificar cambio de signo.* $f(1.5) = 1.5^3 - 6(1.5)^2 + 11(1.5) - 6 = 3.375 - 13.5 + 16.5 - 6 = 0.375 > 0$, $f(2.5) = 15.625 - 37.5 + 27.5 - 6 = -0.375 < 0$. *Existe cambio de signo, por lo que hay al menos una raíz en el intervalo* $[1.5, 2.5]$.

- *Aplicar el método de bisección.* $c_1 = \frac{1.5+2.5}{2} = 2.0, \qquad f(2.0) = 8 - 24 + 22 - 6 = 0$. *Se obtiene exactamente la raíz en la primera iteración.*

- *Para ilustrar mejor, tomemos el intervalo más amplio* $[1, 3]$*:* $f(1) = 0, \quad f(3) = 0$. *Aquí no hay cambio de signo estricto (ya que ambos extremos son raíces), por tanto el método se aplica en un subintervalo, por ejemplo* $[1.2, 2.8]$.

- *Iteraciones del método:*

| *Iteración* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.200* | *2.800* | *2.000* | $0.000$ |

- *Nuevas evaluaciones:* $f(1.5) = 3.375 - 13.5 + 16.5 - 5.5 = 0.875 > 0$, $f(2.5) = 15.625 - 37.5 + 27.5 - 5.5 = 0.125 > 0$, $f(1.0) = 1 - 6 + 11 - 5.5 = 0.5 > 0$, $f(3.0) = 27 - 54 + 33 - 5.5 = 0.5 > 0$.

- *Busquemos ahora en el intervalo* $[1.5, 2.5]$ *ajustando un poco los valores hasta encontrar cambio de signo:* $f(1.7) \approx 0.28$, $f(1.9) \approx 0.03$, $f(2.0) \approx -0.5$. *Entonces hay cambio de signo en* $[1.8, 2.0]$.

| *Iteración* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.800* | *2.000* | *1.900* | $+0.030$ |
| *2* | *1.900* | *2.000* | *1.950* | $-0.210$ |
| *3* | *1.800* | *1.950* | *1.875* | $+0.130$ |
| *4* | *1.875* | *1.950* | *1.9125* | $-0.040$ |
| *5* | *1.875* | *1.9125* | *1.8937* | $+0.040$ |
| *6* | *1.8937* | *1.9125* | *1.9031* | $-0.005$ |

*La raíz aproximada se encuentra en* $x \approx 1.903$.

**Ejemplo 26.** *Consideremos la cúbica* $f(x) = x^3 - 6x^2 + 11x - 5.5$*, con intervalo inicial* $[a_0, b_0] = [0.8, 0.9]$ *donde hay cambio de signo. Las primeras 8 iteraciones (valores redondeados) son:*

| *Iter.* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *0.8000000* | *0.9000000* | *0.8500000* | $+1.29125 \times 10^{-1}$ |
| *2* | *0.8000000* | *0.8500000* | *0.8250000* | $+5.27656 \times 10^{-2}$ |
| *3* | *0.8000000* | *0.8250000* | *0.8125000* | $+1.29395 \times 10^{-2}$ |
| *4* | *0.8000000* | *0.8125000* | *0.8062500* | $-7.39038 \times 10^{-3}$ |
| *5* | *0.8062500* | *0.8125000* | *0.8093750* | $+2.80942 \times 10^{-3}$ |
| *6* | *0.8062500* | *0.8093750* | *0.8078125* | $-2.28175 \times 10^{-3}$ |
| *7* | *0.8078125* | *0.8093750* | *0.8085938* | $+2.66016 \times 10^{-4}$ |
| *8* | *0.8078125* | *0.8085938* | *0.8082031* | $-1.00732 \times 10^{-3}$ |

*Después de 8 iteraciones, la aproximación de la raíz es* $c_8 \approx 0{,}8082031$ *y la cota de error por ancho de intervalo es* $\frac{b_8 - a_8}{2} = \texttt{error\_cota} = \frac{0.8085938 - 0.8078125}{2} \approx 3{,}906 \times 10^{-4}$.

**Ejemplo 27.** *Sea* $f(x) = x^2 - 2x$ *con* $a = 1$ *y* $b = -3$ *pues* $ax^2 + bx + x = x^2 - 3x + x = x^2 - 2x$*. Usamos el intervalo* $[1.2, 3.0]$*:* $f(1.2) = 1.44 - 2.4 = -0.96 < 0$*,* $f(3) = 9 - 6 = 3 > 0$.

| *Iter.* | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.200000* | *3.000000* | *2.100000* | $+0.210000$ |
| *2* | *1.200000* | *2.100000* | *1.650000* | $-0.577500$ |
| *3* | *1.650000* | *2.100000* | *1.875000* | $-0.234375$ |
| *4* | *1.875000* | *2.100000* | *1.987500* | $-0.024844$ |
| *5* | *1.987500* | *2.100000* | *2.043750* | $+0.089414$ |
| *6* | *1.987500* | *2.043750* | *2.015625* | $+0.031494$ |
| *7* | *1.987500* | *2.015625* | *2.001563* | $+0.003127$ |
| *8* | *1.987500* | *2.001563* | *1.994531* | $-0.010908$ |

**Ejemplo 28.** *Sea la función* $f(x) = x^2 - 2x$ *en el intervalo elegido:* $[1.2, 3.0]$

- $f(1.2) = 1.44 - 2.4 = -0.96, \quad f(3) = 9 - 6 = 3.$
- *Iteración 1:* $c_1 = \frac{1.2+3}{2} = 2.1, \quad f(2.1) = 4.41 - 4.2 = 0.21 > 0$ *entonces* $[1.2, 2.1]$.
- *Iteración 2:* $c_2 = \frac{1.2+2.1}{2} = 1.65, \quad f(1.65) = 2.7225 - 3.3 = -0.5775 < 0$ *entonces* $[1.65, 2.1]$.

- *Iteración 3:* $c_3 = 1.875, \quad f(1.875) = 3.516 - 3.75 = -0.234 < 0$ *entonces* $[1.875, 2.1]$. *Y así sucesivamente hasta* $c_8 \approx 1.9945$.

| $n$ | $a_n$ | $b_n$ | $c_n$ | $f(c_n)$ |
|---|---|---|---|---|
| *1* | *1.200* | *3.000* | *2.100* | *+0.210* |
| *2* | *1.200* | *2.100* | *1.650* | *-0.577* |
| *3* | *1.650* | *2.100* | *1.875* | *-0.234* |
| *4* | *1.875* | *2.100* | *1.987* | *-0.024* |
| *5* | *1.987* | *2.100* | *2.043* | *+0.089* |
| *6* | *1.987* | *2.043* | *2.015* | *+0.031* |
| *7* | *1.987* | *2.015* | *2.001* | *+0.003* |
| *8* | *1.987* | *2.001* | *1.994* | *-0.011* |

**Ejemplo 29.** $f(x) = x^3 - x - 2$ *en* $[1, 2]$. *Se verifica* $f(1) = -2 < 0$ *y* $f(2) = 4 > 0$.

*Primeras iteraciones (hasta alcanzar* $10^{-6}$ *típicamente se requieren* $\sim 20$ *pasos desde* $[1, 2]$*):*

Cuadro 4.1: Método de bisección para $f(x) = x^3 - x - 2$, intervalo inicial $[-3, 2]$.

| Iter. | $a$ | $b$ | $c = \frac{a+b}{2}$ | $f(a)$ | $f(b)$ | $f(c)$ | sign$(f(a)f(c))$ | sign$(f(b)$ |
|---|---|---|---|---|---|---|---|---|
| 1 | -3.000000000 | 2.000000000 | -0.500000000 | -26.000000000 | 4.000000000 | -1.625000000 | + | − |
| 2 | -0.500000000 | 2.000000000 | 0.750000000 | -1.625000000 | 4.000000000 | -2.328125000 | + | − |
| 3 | 0.750000000 | 2.000000000 | 1.375000000 | -2.328125000 | 4.000000000 | -0.775390625 | + | − |
| 4 | 1.375000000 | 2.000000000 | 1.687500000 | -0.775390625 | 4.000000000 | 1.117919922 | − | + |
| 5 | 1.375000000 | 1.687500000 | 1.531250000 | -0.775390625 | 1.117919922 | 0.059112549 | − | + |
| 6 | 1.375000000 | 1.531250000 | 1.453125000 | -0.775390625 | 0.059112549 | -0.392456055 | + | − |
| 7 | 1.453125000 | 1.531250000 | 1.492187500 | -0.392456055 | 0.059112549 | -0.172897339 | + | − |
| 8 | 1.492187500 | 1.531250000 | 1.511718750 | -0.172897339 | 0.059112549 | -0.058979273 | + | − |
| 9 | 1.511718750 | 1.531250000 | 1.521484375 | -0.058979273 | 0.059112549 | 0.000622176 | − | + |
| 10 | 1.511718750 | 1.521484375 | 1.516601562 | -0.058979273 | 0.000622176 | -0.029292628 | + | − |
| 11 | 1.516601562 | 1.521484375 | 1.519042969 | -0.029292628 | 0.000622176 | -0.013864168 | + | − |
| 12 | 1.519042969 | 1.521484375 | 1.520263672 | -0.013864168 | 0.000622176 | -0.006627792 | + | − |

*Con tolerancia* $\varepsilon = 10^{-6}$*, el resultado se estabiliza alrededor de* $\alpha \approx 1{,}52138$.

**Ejemplo 30.** $f(x) = \cos x - x$ *en* $[0, 1]$. *Se verifica* $f(0) = 1 > 0$ *y* $f(1) \approx -0{,}4597 < 0$.

*Primeras iteraciones:*

Cuadro 4.2: Método de bisección para $f(x) = \cos x - x$ en $[0, 1]$.

| Iter. | $a$ | $b$ | $c = \frac{a+b}{2}$ | $f(a)$ | $f(b)$ | $f(c)$ | sign$(f(a)f(c))$ | sign$(f(b)f(c))$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 1.000000000 | 0.500000000 | 1.000000000 | -0.459697694 | 0.377582562 | − | + |
| 2 | 0.500000000 | 1.000000000 | 0.750000000 | 0.377582562 | -0.459697694 | -0.018311132 | + | − |
| 3 | 0.500000000 | 0.750000000 | 0.625000000 | 0.377582562 | -0.018311132 | 0.185963120 | − | + |
| 4 | 0.625000000 | 0.750000000 | 0.687500000 | 0.185963120 | -0.018311132 | 0.085334946 | − | + |
| 5 | 0.687500000 | 0.750000000 | 0.718750000 | 0.085334946 | -0.018311132 | 0.033879372 | − | + |
| 6 | 0.718750000 | 0.750000000 | 0.734375000 | 0.033879372 | -0.018311132 | 0.007874725 | − | + |
| 7 | 0.734375000 | 0.750000000 | 0.742187500 | 0.007874725 | -0.018311132 | -0.005195711 | + | − |
| 8 | 0.734375000 | 0.742187500 | 0.738281250 | 0.007874725 | -0.005195711 | 0.001345150 | − | + |
| 9 | 0.738281250 | 0.742187500 | 0.740234375 | 0.001345150 | -0.005195711 | -0.001923872 | + | − |
| 10 | 0.738281250 | 0.740234375 | 0.739257812 | 0.001345150 | -0.001923872 | -0.000289009 | + | − |
| 11 | 0.738281250 | 0.739257812 | 0.738769531 | 0.001345150 | -0.000289009 | 0.000528158 | − | + |
| 12 | 0.738769531 | 0.739257812 | 0.739013672 | 0.000528158 | -0.000289009 | 0.000119610 | − | + |

*Con* $\varepsilon = 10^{-6}$ *se obtiene aproximadamente* $\alpha \approx 0{,}739085$.

**Ejemplo 31.** *Aplica el método de la bisección para las siguientes funciones:*

1. *Encuentra una raíz de* $x^3 - 6x + 2$ *en* $[0, 2]$.

Cuadro 4.3: Método de la Bisección para $f(x) = x^3 - 6x + 2$ en $[0, 2]$ (12 iteraciones).

| Iter. | $a$ | $b$ | $c = \frac{a+b}{2}$ | $f(a)$ | $f(b)$ | $f(c)$ | sign$(f(a)f(c))$ | sign$(f(b)f(c))$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 2.000000000 | 1.000000000 | 2.000000000 | -2.000000000 | -3.000000000 | − | + |
| 2 | 0.000000000 | 1.000000000 | 0.500000000 | 2.000000000 | -3.000000000 | -0.875000000 | − | + |
| 3 | 0.000000000 | 0.500000000 | 0.250000000 | 2.000000000 | -0.875000000 | 0.515625000 | + | − |
| 4 | 0.250000000 | 0.500000000 | 0.375000000 | 0.515625000 | -0.875000000 | -0.197265625 | − | + |
| 5 | 0.250000000 | 0.375000000 | 0.312500000 | 0.515625000 | -0.197265625 | 0.155517578 | + | − |
| 6 | 0.312500000 | 0.375000000 | 0.343750000 | 0.155517578 | -0.197265625 | -0.021881104 | − | + |
| 7 | 0.312500000 | 0.343750000 | 0.328125000 | 0.155517578 | -0.021881104 | 0.066577911 | + | − |
| 8 | 0.328125000 | 0.343750000 | 0.335937500 | 0.066577911 | -0.021881104 | 0.022286892 | + | − |
| 9 | 0.335937500 | 0.343750000 | 0.339843750 | 0.022286892 | -0.021881104 | 0.000187337 | + | − |
| 10 | 0.339843750 | 0.343750000 | 0.341796875 | 0.000187337 | -0.021881104 | -0.010850795 | − | + |
| 11 | 0.339843750 | 0.341796875 | 0.340820312 | 0.000187337 | -0.010850795 | -0.005332704 | − | + |
| 12 | 0.339843750 | 0.340820312 | 0.340332031 | 0.000187337 | -0.005332704 | -0.002572927 | − | + |

2. *Estudia* $f(x) = e^{-x} - x$ *en* $[0, 1]$.

Cuadro 4.4: Método de Bisección para $f(x) = e^{-x} - x$ en $[0, 1]$ (12 iteraciones).

| Iter. | $a$ | $b$ | $c = \frac{a+b}{2}$ | $f(a)$ | $f(b)$ | $f(c)$ | sign$(f(a)f(c))$ | sign$(f(b)f(c))$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 1.000000000 | 0.500000000 | 1.000000000 | -0.632120559 | 0.106530660 | + | − |
| 2 | 0.500000000 | 1.000000000 | 0.750000000 | 0.106530660 | -0.632120559 | -0.277633447 | − | + |
| 3 | 0.500000000 | 0.750000000 | 0.625000000 | 0.106530660 | -0.277633447 | -0.089738571 | − | + |
| 4 | 0.500000000 | 0.625000000 | 0.562500000 | 0.106530660 | -0.089738571 | 0.007282825 | + | − |
| 5 | 0.562500000 | 0.625000000 | 0.593750000 | 0.007282825 | -0.089738571 | -0.041497550 | − | + |
| 6 | 0.562500000 | 0.593750000 | 0.578125000 | 0.007282825 | -0.041497550 | -0.017175839 | − | + |
| 7 | 0.562500000 | 0.578125000 | 0.570312500 | 0.007282825 | -0.017175839 | -0.004963760 | − | + |
| 8 | 0.562500000 | 0.570312500 | 0.566406250 | 0.007282825 | -0.004963760 | 0.001155202 | + | − |
| 9 | 0.566406250 | 0.570312500 | 0.568359375 | 0.001155202 | -0.004963760 | -0.001905360 | − | + |
| 10 | 0.566406250 | 0.568359375 | 0.567382812 | 0.001155202 | -0.001905360 | -0.000375349 | − | + |
| 11 | 0.566406250 | 0.567382812 | 0.566894531 | 0.001155202 | -0.000375349 | 0.000389859 | + | − |
| 12 | 0.566894531 | 0.567382812 | 0.567138672 | 0.000389859 | -0.000375349 | 0.000007238 | + | − |

3. *Sea la función cuadrática* $f(x) = x^2 - 4$ *en* $(-1, 4)$.

Cuadro 4.5: Método de Bisección para $f(x) = x^2 - 4$ en $(-1, 4)$ (12 iteraciones).

| Iter. | $a$ | $b$ | $c = \frac{a+b}{2}$ | $f(a)$ | $f(b)$ | $f(c)$ | sign$(f(a)f(c))$ | sign$(f(b)f(c))$ |
|---|---|---|---|---|---|---|---|---|
| 1 | -1.000000000 | 4.000000000 | 1.500000000 | -3.000000000 | 12.000000000 | -1.750000000 | + | − |
| 2 | 1.500000000 | 4.000000000 | 2.750000000 | -1.750000000 | 12.000000000 | 3.562500000 | − | + |
| 3 | 1.500000000 | 2.750000000 | 2.125000000 | -1.750000000 | 3.562500000 | 0.515625000 | − | + |
| 4 | 1.500000000 | 2.125000000 | 1.812500000 | -1.750000000 | 0.515625000 | -0.714843750 | + | − |
| 5 | 1.812500000 | 2.125000000 | 1.968750000 | -0.714843750 | 0.515625000 | -0.122558594 | + | − |
| 6 | 1.968750000 | 2.125000000 | 2.046875000 | -0.122558594 | 0.515625000 | 0.189208984 | − | + |
| 7 | 1.968750000 | 2.046875000 | 2.007812500 | -0.122558594 | 0.189208984 | 0.031494141 | − | + |
| 8 | 1.968750000 | 2.007812500 | 1.988281250 | -0.122558594 | 0.031494141 | -0.063354492 | + | − |
| 9 | 1.988281250 | 2.007812500 | 1.998046875 | -0.063354492 | 0.031494141 | -0.007873535 | + | − |
| 10 | 1.998046875 | 2.007812500 | 2.002929688 | -0.007873535 | 0.031494141 | 0.011726379 | − | + |
| 11 | 1.998046875 | 2.002929688 | 2.000488281 | -0.007873535 | 0.011726379 | 0.001953602 | − | + |
| 12 | 1.998046875 | 2.000488281 | 1.999267578 | -0.007873535 | 0.001953602 | -0.002960747 | + | − |

### 4.1.2. Implementaciones numéricas

**Algoritmo 3.** *Pseudocódigo*

```
Inicio
    Definir f(x)
    Leer a, b, tolerancia, iter_max

    Si f(a)*f(b) > 0 Entonces
        Imprimir "No hay cambio de signo en [a,b]"
        Terminar
    FinSi

    iter <-- 0
    error <-- |b - a|

    Mientras (error > tolerancia) Y (iter < iter_max) Hacer
        c <- (a + b)/2

        Si f(a)*f(c) < 0 Entonces
            b <-- c
        Sino
            a <-- c
        FinSi

        error <-- |b - a|
        iter <-- iter + 1
    FinMientras

    Imprimir "Ra\'iz aproximada:", c
    Imprimir "Iteraciones:", iter
Fin
```

*Implementación numérica*

```
# M\'etodo de Bisecci\'on
biseccion <- function(f, a, b, tol = 1e-6, iter_max = 100){
  if(f(a)*f(b) > 0){
    cat("Error: no hay cambio de signo en el intervalo [a,b].\n")
    return(NA)
  }

  iter <- 0
  error <- abs(b - a)

  while(error > tol && iter < iter_max){
```

```
    c <- (a + b)/2

    if(f(a)*f(c) < 0){
      b <- c
    } else {
      a <- c
    }

    error <- abs(b - a)
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", c, "\n")
  cat("Iteraciones:", iter, "\n")
  return(c)
}

# Ejemplo de uso:
# f(x) = x^3 - x - 2
f <- function(x) x^3 - x - 2
biseccion(f, a = 1, b = 2)
```

**Algoritmo 4.** `# Definir f(x) (la funci\'on matem\'atica a evaluar)`

```
f <- function(x) x^3 - 6*x^2 + 11*x - 5.5

# Intervalo inicial con cambio de signo
a <- 0.8;  b <- 0.9
fa <- f(a); fb <- f(b)
if (fa * fb >= 0) stop("El intervalo inicial no tiene cambio de signo.")

# Estructura para guardar el historial (n, a, b, c, f(c), ancho)
hist <- data.frame(
  n = integer(), a = numeric(), b = numeric(),
  c = numeric(), fc = numeric(), ancho = numeric(),
  stringsAsFactors = FALSE
)

# Ejecutar exactamente 8 iteraciones, sin usar ninguna funcion/recursividad
for (n in 1:8) {
  c  <- (a + b)/2
  fc <- f(c)
  ancho <- (b - a)/2

  # Guardar fila
  hist[n,] <- list(n, a, b, c, fc, ancho)
```

```
  # Mostrar en consola (opcional)
  cat(sprintf("Iter %2d | a=%.7f b=%.7f c=%.7f f(c)=%.7e ancho=%.7e\n",
              n, a, b, c, fc, ancho))

  # Actualizar el intervalo seg\'un el signo
  if (fa * fc < 0) {
    b <- c; fb <- fc
  } else {
    a <- c; fa <- fc
  }
}

# Resultado aproximado tras 8 iteraciones:
c_aprox <- hist$c[8]
error_cota <- hist$ancho[8]  # (b - a)/2 despu\'es del paso 8
c_aprox; error_cota
hist
```

**Algoritmo 5.**

```
f <- function(x) x^2 - 2
a <- 1.0; b <- 2.0
fa <- f(a); fb <- f(b)
if (fa*fb >= 0) stop("Sin cambio de signo en [a,b].")
histA <- data.frame(n=integer(), a=numeric(), b=numeric(),
                    c=numeric(), fc=numeric(), ancho=numeric())
for (n in 1:8) {
  c  <- (a + b)/2
  fc <- f(c)
  histA[n,] <- list(n, a, b, c, fc, (b-a)/2)
  cat(sprintf("A-%2d | a=%.6f b=%.6f c=%.6f f(c)=%.6e ancho=%.6e\n",
              n, a, b, c, fc, (b-a)/2))
  if (fa * fc < 0) { b <- c; fb <- fc } else { a <- c; fa <- fc }
}
```

**Algoritmo 6.**

```
f <- function(x) x^2 - 2*x
a <- 1.2; b <- 3.0
fa <- f(a); fb <- f(b)
if (fa*fb >= 0) stop("Sin cambio de signo en [a,b].")
histB <- data.frame(n=integer(), a=numeric(), b=numeric(),
                    c=numeric(), fc=numeric(), ancho=numeric())
for (n in 1:8) {
  c  <- (a + b)/2
  fc <- f(c)
  histB[n,] <- list(n, a, b, c, fc, (b-a)/2)
  cat(sprintf("B-%2d | a=%.6f b=%.6f c=%.6f f(c)=%.6e ancho=%.6e\n",
              n, a, b, c, fc, (b-a)/2))
  if (fa * fc < 0) { b <- c; fb <- fc } else { a <- c; fa <- fc }
```

```
}
```

## 4.2. Método de la Secante

El método de la secante aproxima la derivada mediante la pendiente de la recta que une dos puntos de la curva, evitando calcular $f'(x)$.

**Algoritmo 7.** *Dado $x_{k-1}$ y $x_k$, definimos*

$$x_{k+1} \;=\; x_k \;-\; f(x_k)\,\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}.$$

- ***Criterios de paro:*** $|x_{k+1} - x_k| < \varepsilon$ *o* $|f(x_{k+1})| < \varepsilon$.
- ***Ventajas:*** *no requiere derivada; suele ser más rápido que bisección.*
- ***Sugerencia:*** *elegir dos semillas razonables; evitar $f(x_k) \approx f(x_{k-1})$ (división por número muy pequeño).*

**Ejemplo 32.** *Función Cuadrática: $f(x) = x^2 - 4$. Semillas: $x_0 = 1$, $x_1 = 3$. Buscamos la raíz positiva (2). Consideremos la aproximación inicial $x_0, x_1$ y el nuevo punto $x_2$ es el punto donde se intersecta la secante con el eje $x$.*

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

- *Iteración 1.*

$$x_2 = 3 - 5\frac{(3-1)}{5-(-3)} = 1.75, \qquad |x_2 - x_1| = 1.25.$$

- *Iteración 2.* $x_0 = 3$, $x_1 = 1.75$, $f(1.75) = -0.9375$,

$$x_3 = 1.75 - (-0.9375)\frac{(1.75-3)}{-0.9375-5} = 1.947368421.$$

- *Iteración 3.* $f(1.947368421) = -0.207756233$,

$$x_4 = 1.947368421 - (-0.207756233)\frac{(1.947368421 - 1.75)}{-0.207756233-(-0.9375)} = 2.003558719.$$

- *Iteración 4.* $f(2.003558719) = 0.014247540$,

$$x_5 = 2.003558719 - (0.014247540)\frac{(2.003558719 - 1.947368421)}{0.014247540-(-0.207756233)} = 1.999952593.$$

- *Iteración 5.* $f(1.999952593) = -1.89625 \times 10^{-4}$, *y* $x_6 = 1.999999958$, *por lo tanto* $x \approx 2.000000000$.

*Solución en R:*

```
# f(x) = x^2 - 4   (x0=1, x1=3)
x0 <- 1.0; x1 <- 3.0
# Iteraci\'on 1
f0 <- x0^2 - 4; f1 <- x1^2 - 4
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 2
f0 <- x0^2 - 4; f1 <- x1^2 - 4
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 3
f0 <- x0^2 - 4; f1 <- x1^2 - 4
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 4
f0 <- x0^2 - 4; f1 <- x1^2 - 4
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 5
f0 <- x0^2 - 4; f1 <- x1^2 - 4
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
root_aprox <- x2
print(root_aprox)
```

**Ejemplo 33.** $f(x) = x^3 - x - 2$. *Aproximación inicial:* $x_0 = 1$, $x_1 = 2$, *entonces*

$$f(1) = -2, \qquad f(2) = 4.$$

- *Iteración 1.*

$$x_2 = 2 - (4)\frac{(2-1)}{4-(-2)} = 1.333333333.$$

- *Iteración 2.* $f(1.333333333) = -0.962962963$*;* $x_3 = 1.462686567$.
- *Iteración 3.* $f(1.462686567) = -0.333338875$*;* $x_4 = 1.531169432$.
- *Iteración 4.* $f(1.531169432) = 0.058626418$*;* $x_5 = 1.520926421$.
- *Iteración 5.*$f(1.520926421) = -0.002693300$*;* $x_6 = 1.521376317$*;*

*Solución en R*

```
# f(x) = x^3 - x - 2   (x0=1, x1=2)
x0 <- 1.0; x1 <- 2.0
# Iteraci\'on 1
f0 <- x0^3 - x0 - 2; f1 <- x1^3 - x1 - 2
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
```

```
x0 <- x1; x1 <- x2
# Iteraci\'on 2
f0 <- x0^3 - x0 - 2; f1 <- x1^3 - x1 - 2
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 3
f0 <- x0^3 - x0 - 2; f1 <- x1^3 - x1 - 2
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 4
f0 <- x0^3 - x0 - 2; f1 <- x1^3 - x1 - 2
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 5
f0 <- x0^3 - x0 - 2; f1 <- x1^3 - x1 - 2
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
root_aprox <- x2
print(root_aprox)
```

**Ejemplo 34.** $f(x) = \cos x - x$. *Aproximación inicial:* $x_0 = 0.5$, $x_1 = 1.0$, *por tanto* $f(0.5) = 0.37758256$ *y* $f(1.0) = -0.45969769$.

- *Iteración 1.* $x_2 = 0.725481587$, $f(x_2) \approx 0.022698391$.
- *Iteración 2.* $x_3 = 0.738398620$, $f(x_3) \approx 0.001148782$.
- *Iteración 3.* $x_4 = 0.739087211$, $f(x_4) \approx -3.4771 \times 10^{-6}$.
- *Iteración 4.* $x_5 = 0.739085133$*; entonces* $x \approx 0.739085133$.

*Solución en R*

```
# f(x) = cos(x) - x   (x0=0.5, x1=1.0)
x0 <- 0.5; x1 <- 1.0
# Iteraci\'on 1
f0 <- cos(x0) - x0; f1 <- cos(x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 2
f0 <- cos(x0) - x0; f1 <- cos(x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 3
f0 <- cos(x0) - x0; f1 <- cos(x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 4
f0 <- cos(x0) - x0; f1 <- cos(x1) - x1
```

```
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
root_aprox <- x2
print(root_aprox)
```

**Ejemplo 35.** $f(x) = e^{-x} - x$. *Aproximación inicial:* $x_0 = 0$, $x_1 = 1$, *por tanto* $f(0) = 1$, $f(1) = e^{-1} - 1 \approx -0.632120559$.

*Iteración 1.* $x_2 = 0.612699837$, $f(x_2) \approx -0.070813948$.

*Iteración 2.* $x_3 = 0.563838389$, $f(x_3) \approx 0.005182355$.

*Iteración 3.* $x_4 = 0.567170358$, $f(x_4) \approx -4.2419 \times 10^{-5}$.

*Iteración 4.* $x_5 = 0.567143307$, $f(x_5) \approx -2.5380 \times 10^{-8}$.

*Iteración 5.* $x_6 = 0.567143290$, $x \approx 0.567143290$.

*Solución con R*

```
# f(x) = exp(-x) - x   (x0=0, x1=1)
x0 <- 0.0; x1 <- 1.0
# Iteraci\'on 1
f0 <- exp(-x0) - x0; f1 <- exp(-x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 2
f0 <- exp(-x0) - x0; f1 <- exp(-x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 3
f0 <- exp(-x0) - x0; f1 <- exp(-x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 4
f0 <- exp(-x0) - x0; f1 <- exp(-x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
x0 <- x1; x1 <- x2
# Iteraci\'on 5
f0 <- exp(-x0) - x0; f1 <- exp(-x1) - x1
x2 <- x1 - f1*(x1 - x0)/(f1 - f0)
root_aprox <- x2
print(root_aprox)
```

**Algoritmo 8.** *Pseudocódigo*

```
Algoritmo: M\'etodo de la Secante

Entrada:
    f(x)        --> funci\'on
    x0, x1      --> valores iniciales
    tol         --> tolerancia (por ejemplo, 1e-6)
```

```
    iter_max    --> n\'umero m\'aximo de iteraciones

Proceso:
    iter <-- 0
    error <-- 1

    Mientras (error > tol) Y (iter < iter_max) Hacer
        fx0 <-- f(x0)
        fx1 <-- f(x1)

        Si |fx1 - fx0| < 1e-12 Entonces
            Imprimir "Error: divisi\'on por cero"
            Salir
        FinSi

        x2 <-- x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        error <-- |x2 - x1|

        x0 <-- x1
        x1 <-- x2
        iter <-- iter + 1
    FinMientras

Salida:
    Imprimir "Ra\'iz aproximada:", x2
    Imprimir "Iteraciones:", iter
FinAlgoritmo
```

*Implememtación numérica*

```
# M\'etodo de la Secante
secante <- function(f, x0, x1, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1

  while(error > tol && iter < iter_max){
    fx0 <- f(x0)
    fx1 <- f(x1)

    # Evitar divisi\'on por cero
    if(abs(fx1 - fx0) < 1e-12){
      cat("Error: Divisi\'on por cero (f(x1) - f(x0) approx 0)\n")
      return(NA)
    }
```

```
    x2 <- x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    error <- abs(x2 - x1)

    # Actualizar valores
    x0 <- x1
    x1 <- x2
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x2, "\n")
  cat("Iteraciones:", iter, "\n")
  return(x2)
}

# Ejemplo de uso:
# f(x) = x^3 - 2x^2 + 3x - 5
f <- function(x) x^3 - 2*x^2 + 3*x - 5
secante(f, x0 = 1, x1 = 2)
```

## 4.3. Método de Newton-Raphson

Se busca una raíz de $f(x) = 0$ usando la recta *tangente* en $x_k$. El siguiente punto está dado por

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{4.5}$$

El criterio de paro es $|x_{k+1} - x_k| < \varepsilon$ o $|f(x_{k+1})| < \varepsilon$.

**Algoritmo 9.** `Entrada: f(x), f'(x), x0, tolerancia eps, Nmax`

```
x = x0
Para k = 1..Nmax:
    fx = f(x); dfx = f'(x)
    si dfx = 0: detener (tangente horizontal)
    x1 = x - fx/dfx
    si |x1 - x| < eps o |f(x1)| < eps: retornar x1
    x = x1
Fin
```

**Ejemplo 36.** $f(x) = x^2 - 4$, $f'(x) = 2x$. *Aproximación inicial* $x_0 = 3$

$$
\begin{aligned}
x_1 &= 3 - \frac{9-4}{2\cdot 3} = 3 - \frac{5}{6} = 2.166666667.\\
x_2 &= 2.166666667 - \frac{(2.166666667)^2 - 4}{2\cdot 2.166666667} = 2.166666667 - \frac{0.694444444}{4.333333334} = 2.006410256.\\
x_3 &= 2.006410256 - \frac{(2.006410256)^2 - 4}{2\cdot 2.006410256} \approx 2.000006410.\\
x_4 &\approx 2.000000000.
\end{aligned}
$$

*Raíz aproximada* $x \approx 2$.

*Solución con R:*

```
# f(x) = x^2 - 4,  f'(x) = 2*x,  x0 = 3; x <- 3.0
# k=1
fx <- x^2 - 4; dfx <- 2*x
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1
# k=2
fx <- x^2 - 4; dfx <- 2*x
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1
# k=3
fx <- x^2 - 4; dfx <- 2*x
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1
root_aprox <- x
print(root_aprox)
```

**Ejemplo 37.** $f(x) = x^3 - x - 2$, $f'(x) = 3x^2 - 1$. *Valor inicial* $x_0 = 1.5$

$$
\begin{aligned}
f(1.5) &= -0.125; f'(1.5) = 5.75,\\
x_1 &= 1.5 - \frac{-0.125}{5.75} = 1.521739130.\\
f(1.521739130) &\approx 0.002137; f'(1.521739130) \approx 5.947070,\\
x_2 &= 1.521739130 - \frac{0.002137}{5.947070} \approx 1.521380215.\\
f(1.521380215) &\approx 6.03 \times 10^{-7},\quad f'(1.521380215) \approx 5.943790,\\
x_3 &\approx 1.521380005,\\
x_4 &\approx 1.521379706.
\end{aligned}
$$

*por tanto* $x \approx 1.521379706$. *Solución con R*

```
# f(x) = x^3 - x - 2,  f'(x) = 3*x^2 - 1,  x0 = 1.5
x <- 1.5

# k=1
fx <- x^3 - x - 2; dfx <- 3*x^2 - 1
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1
```

```
# k=2
fx <- x^3 - x - 2; dfx <- 3*x^2 - 1
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1

# k=3
fx <- x^3 - x - 2; dfx <- 3*x^2 - 1
x1 <- x - fx/dfx; err <- abs(x1 - x); x <- x1

root_aprox <- x
print(root_aprox)
```

**Ejemplo 38.** $f(x) = \cos x - x$, $f'(x) = -\sin x - 1$. *Aproximación inicial* $x_0 = 0.5$

$$\begin{aligned} f(0.5) &= 0.37758256, \quad f'(0.5) = -\sin(0.5) - 1 \approx -1.47942554, \\ x_1 &= 0.5 - \frac{0.37758256}{-1.47942554} = 0.755222. \\ f(0.755222) &\approx -0.027103, \quad f'(0.755222) \approx -1.685451, \\ x_2 &= 0.755222 - \frac{-0.027103}{-1.685451} \approx 0.739142. \\ f(0.739142) &\approx -9.46 \times 10^{-5}, \quad f'(0.739142) \approx -1.673654, \\ x_3 &\approx 0.739085; \\ x_4 &\approx 0.739085133. \end{aligned}$$

*por lo tanto* $x \approx 0.739085133$.

**Ejemplo 39.** $f(x) = e^{-x} - x$, $f'(x) = -e^{-x} - 1$. *Aproximación inicial* $x_0 = 0$

$$\begin{aligned} f(0) &= 1, \quad f'(0) = -2 \Rightarrow x_1 = 0 - \frac{1}{-2} = 0.5. \\ f(0.5) &= e^{-0.5} - 0.5 \approx 0.10653066, \quad f'(0.5) = -e^{-0.5} - 1 \approx -1.60653066, \\ x_2 &= 0.5 - \frac{0.10653066}{-1.60653066} \approx 0.566311. \\ f(0.566311) &\approx 0.001157, \quad f'(0.566311) \approx -1.567468, \\ x_3 &\approx 0.567049, \quad x_4 \approx 0.56714329. \end{aligned}$$

*por lo tanto la raíz se encuentra en* $x \approx 0.56714329$.

**Algoritmo 10.** *Pseudocódigo*

```
Inicio
    Definir f(x) y f'(x)
    Leer x0, tolerancia, iter_max
    iter <-- 0
    error <-- 1
```

```
    Mientras (error > tolerancia) Y (iter < iter_max) Hacer
        x1 <-- x0 - f(x0)/f'(x0)
        error <-- |x1 - x0|
        x0 <-- x1
        iter <-- iter + 1
    FinMientras

    Imprimir "Ra\'iz aproximada:", x1
    Imprimir "Iteraciones:", iter
Fin
```

*Implementación numérica*

```
# M\'etodo de Newton-Raphson
newton <- function(f, df, x0, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1

  while(error > tol && iter < iter_max){
    x1 <- x0 - f(x0)/df(x0)
    error <- abs(x1 - x0)
    x0 <- x1
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x1, "\n")
  cat("Iteraciones:", iter, "\n")
  return(x1)
}

# Ejemplo de uso:
# f(x) = x^2 - 2
f  <- function(x) x^2 - 2
df <- function(x) 2*x
newton(f, df, x0 = 1)
```

## 4.4. Método del punto fijo

Dada una ecuación $f(x) = 0$, elegimos una función $g$ tal que

$$f(x) = 0 \quad \text{sí y sólo sí } x = g(x).$$

El método de **punto fijo** construye la sucesión

$$x_{k+1} = g(x_k), \qquad k = 0, 1, 2, \ldots$$

**Teorema 15.** *Si $g$ es continua en un intervalo $I$ que contiene a la raíz $r$, $g(I) \subset I$ y $\max_{x\in I} |g'(x)| = L < 1$, entonces existe un único punto fijo $r \in I$ y para $x_0 \in I$ se cumple $x_k \to r$ con convergencia al menos lineal (factor $\leq L$).*

**Algoritmo 11.**
```
Entrada: g(x), x0, tolerancia eps, Nmax
x = x0
Para k = 1..Nmax:
    x1 = g(x)
    si |x1 - x| < eps  o  |f(x1)| < eps: retornar x1
    x = x1
Fin
```

**Ejemplo 40.** $f(x) = x^2 - 4 = 0$

$$g(x) = \frac{1}{2}\Big(x + \frac{4}{x}\Big) \quad \textit{(iteración de Herón para } \sqrt{4}\textit{)}.$$

*Entonces $x_{k+1} = g(x_k)$ y el punto fijo es $r = 2$. Tomamos $x_0 = 3$.*

$$\begin{aligned}
x_1 &= \frac{1}{2}(3 + \tfrac{4}{3}) = 2.166666667.\\
x_2 &= \frac{1}{2}\Big(2.166666667 + \tfrac{4}{2.166666667}\Big) = 2.006410256.\\
x_3 &= \frac{1}{2}\Big(2.006410256 + \tfrac{4}{2.006410256}\Big) = 2.000003626.\\
x_4 &= \frac{1}{2}\Big(2.000003626 + \tfrac{4}{2.000003626}\Big) = 2.000000000.\\
x_5 &\approx 2.000000000.
\end{aligned}$$

*por lo tanto $r \approx 2$. Aquí $|g'(r)| = \frac{1}{2}(1 - \frac{4}{r^2}) = 0$, por eso converge muy rápido. Solución con R*

```
# g(x) = 0.5*(x + 4/x)   (x0 = 3)
x <- 3.0

# Iteraci\'on 1
x1 <- 0.5*(x + 4/x); x <- x1

# Iteraci\'on 2
x1 <- 0.5*(x + 4/x); x <- x1

# Iteraci\'on 3
x1 <- 0.5*(x + 4/x); x <- x1

# Iteraci\'on 4
x1 <- 0.5*(x + 4/x); x <- x1
```

```
# Iteraci\'on 5
x1 <- 0.5*(x + 4/x); x <- x1

root_aprox <- x
print(root_aprox)
```

**Ejemplo 41.** $f(x) = x^3 - x - 2 = 0$. *Reescribimos* $x = \sqrt[3]{x+2}$

$$g(x) = (x+2)^{1/3}.$$

*Cerca de la raíz* $r \approx 1.52138$, $|g'(r)| = \frac{1}{3}(r+2)^{-2/3} < 1$. *Tomamos* $x_0 = 1$.

$$\begin{aligned} x_1 &= (1+2)^{1/3} = 1.44224957. \\ x_2 &= (1.44224957+2)^{1/3} = 1.51571657. \\ x_3 &= (1.51571657+2)^{1/3} = 1.52137900. \\ x_4 &= (1.52137900+2)^{1/3} = 1.52137966. \\ x_5 &= (1.52137966+2)^{1/3} = 1.52137971. \end{aligned}$$

*por lo tanto* $r \approx 1.52137971$. *Solución con R*

```
# g(x) = (x + 2)^(1/3)   (x0 = 1)
x <- 1.0

# Iteraci\'on 1
x1 <- (x + 2)^(1/3); x <- x1

# Iteraci\'on 2
x1 <- (x + 2)^(1/3); x <- x1

# Iteraci\'on 3
x1 <- (x + 2)^(1/3); x <- x1

# Iteraci\'on 4
x1 <- (x + 2)^(1/3); x <- x1

# Iteraci\'on 5
x1 <- (x + 2)^(1/3); x <- x1

root_aprox <- x
print(root_aprox)
```

**Ejemplo 42.** $f(x) = \cos x - x = 0$ *Se selecciona* $x = \cos x$*:*

$$g(x) = \cos x.$$

$|g'(r)| = |-\sin r| \approx 0.673 < 1$. *Tomamos* $x_0 = 0.5$.

$$\begin{aligned} x_1 &= \cos(0.5) = 0.87758256. \\ x_2 &= \cos(0.87758256) = 0.63901249. \\ x_3 &= \cos(0.63901249) = 0.80268510. \\ x_4 &= \cos(0.80268510) = 0.69477803. \\ x_5 &= \cos(0.69477803) = 0.76819583. \end{aligned}$$

*Solución con R*

```
# g(x) = cos(x)   (x0 = 0.5)
x <- 0.5

# Iteraci\'on 1
x1 <- cos(x); x <- x1

# Iteraci\'on 2
x1 <- cos(x); x <- x1

# Iteraci\'on 3
x1 <- cos(x); x <- x1

# Iteraci\'on 4
x1 <- cos(x); x <- x1

# Iteraci\'on 5
x1 <- cos(x); x <- x1

root_aprox <- x
print(root_aprox)
```

**Ejemplo 43.** $f(x) = e^{-x} - x = 0$, *se toma*

$$g(x) = e^{-x}.$$

*En la raíz* $r \approx 0.567143$, $|g'(r)| = e^{-r} \approx 0.567 < 1$. *Tomamos* $x_0 = 1$.

$$\begin{aligned} x_1 &= e^{-1} = 0.36787944. \\ x_2 &= e^{-0.36787944} = 0.69220063. \\ x_3 &= e^{-0.69220063} = 0.50047350. \\ x_4 &= e^{-0.50047350} = 0.60624354. \\ x_5 &= e^{-0.60624354} = 0.54539579. \end{aligned}$$

*Solución con R*

```
# g(x) = exp(-x)   (x0 = 1)
x <- 1.0

# Iteraci\'on 1
x1 <- exp(-x); x <- x1

# Iteraci\'on 2
x1 <- exp(-x); x <- x1

# Iteraci\'on 3
x1 <- exp(-x); x <- x1

# Iteraci\'on 4
x1 <- exp(-x); x <- x1

# Iteraci\'on 5
x1 <- exp(-x); x <- x1

root_aprox <- x
print(root_aprox)
```

**Algoritmo 12.** *Pseudocódigo*

```
Inicio
    Definir f(x) y g(x)
    Leer x0, tolerancia, iter_max
    iter <-- 0
    error <-- 1

    Mientras (error > tolerancia) Y (iter < iter_max) Hacer
        x1 <-- g(x0)
        error <-- |x1 - x0|
        x0 <-- x1
        iter <-- iter + 1
    FinMientras

    Imprimir "Ra\'iz aproximada:", x1
    Imprimir "Iteraciones:", iter
Fin
```

*Implementación numérica:*

```
# M\'etodo del Punto Fijo
fijo <- function(g, x0, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1
```

```
  while(error > tol && iter < iter_max){
    x1 <- g(x0)
    error <- abs(x1 - x0)
    x0 <- x1
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x1, "\n")
  cat("Iteraciones:", iter, "\n")
  return(x1)
}

# Ejemplo de uso:
# f(x) = cos(x) - x  -->  g(x) = cos(x)
g <- function(x) cos(x)
fijo(g, x0 = 0.5)
```

## 4.5. Ejercicios

### Método de Bisección

**Instrucciones:**

1. Para cada función $f(x)$, encuentre un intervalo $[a, b]$ tal que $f(a) \cdot f(b) < 0$.
2. Realice al menos 8 iteraciones del método de bisección.
3. Registre los valores $a_k, b_k, c_k, f(a_k), f(b_k), f(c_k)$, signo$(f(a_k)f(c_k))$ y el error $|b_k - a_k|$.
4. Grafique la función y los intervalos de reducción en cada paso.

**Ejercicio 9.** *Resuelve los siguientes ejercicioes aplicando el método de la bisección*

1. $f(x) = x^2 - 5$ *en* $[2, 3]$
2. $f(x) = x^3 - 4x + 1$ *en* $[0, 1]$
3. $f(x) = x^4 - 8x + 2$ *en* $[0, 3]$
4. $f(x) = e^{-x} - \frac{x}{2}$ *en* $[0, 2]$
5. $f(x) = \cos(x) - x^2$ *en* $[0, 1]$
6. $f(x) = 3x^2 - 7x - 2$. $[a, b] = [-1, 0]$.
7. $f(x) = -4x^2 + 5x + 6$.$[a, b] = [-1, 0]$.
8. $f(x) = 2x^2 + 3x - 5$. $[a, b] = [0, 2]$.
9. $f(x) = x^3 - 5x + 1$. $[a, b] = [0, 1]$.
10. $f(x) = x^4 - 6x^3 + 5x^2 + 8x - 4$. $[a, b] = [0, 1]$.
11. $f(x) = -2x^4 + 4x^3 + 6x^2 - 3x + 2$. $[a, b] = [-2, -1]$.
12. $f(x) = 2x^4 - x^3 - 7x^2 + 4x + 3$. $[a, b] = [-1, 0]$.

*13.* $f(x) = x^4 + 2x^3 - 10x^2 - x + 5$. $[a, b]$ ~~*16.*~~ $[0,\ 1]$.

*14.* $f(x) = 5\,e^{0.6x-1} - 3$. $[a, b] = [0,\ 2]$.

*15.* $f(x) = 7 - 4\,e^{-0.9x+0.2}$. $[a, b] = [-2,\ 0]$.

*16.* $f(x) = \sin(2x) - \dfrac{x}{2}$. $[a, b] = [1,\ 2]$.

*17.* $f(x) = (2x + 1.4)\,\cos\!\left(3x + \frac{\pi}{4}\right)$. $[a, b] = [0.1,\ 0.6]$.

*18.* $f(x) = (-1.5x + 0.9)\,\sin\!\left(4x - \frac{\pi}{6}\right)$. $[a, b] = [0.2,\ 0.9]$.

**Formato sugerido de tabla:**

| $k$ | $a$ | $b$ | $c$ | $f(a)$ | $f(b)$ | $f(c)$ | $signo(f(a) * f(c))$ | $error$ |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| ... | | | | | | | | |

Cuadro 4.6: Iteraciones del método de bisección para $f(x)$.

### 4.5.1. Método de la Secante

**Instrucciones:**

1. Para cada función $f(x)$, elija dos puntos iniciales $x_0, x_1$ cercanos a la raíz.
2. Aplique iterativamente:

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

3. Realice al menos 6 iteraciones o hasta alcanzar una tolerancia de $10^{-5}$.
4. Compare el número de iteraciones con el método de bisección.
5. Grafique las secantes de cada iteración y el punto de intersección con el eje $x$.

**Ejercicio 10.** *Resuelva los siguientes ejercicios aplicando el método de la secante:*

*1.* $f(x) = x^2 - 2x - 3$, *con* $x_0 = 0$, $x_1 = 3$

*2.* $f(x) = x^4 - 10x^2 + 9$, $x_0 = 0$, $x_1 = 2$

*3.* $f(x) = 2x^2 - 5 * x + 3$, $x_0 = -1$, $x_1 = 2$

*4.* $f(x) = -3x^2 + 4 * x + 2$, $x_0 = -1$, $x_1 = 3$

5. $f(x) = x^3 + 4x^2 - x - 1$, *con* $x_0 = -3$, $x_1 = 1.5$
   $x_1 = 1.5$
6. $f(x) = x^4 - 5x^3 + 6x^2 + 4x - 8$, *con* $x_0 = 0$, $x_1 = 1.1$
   $x_1 = 3$
7. $f(x) = x^4 - 5$, *con* $x_0 = 1$, $x_1 = 2$
8. $f(x) = -2x^4 + 3x^3 + 7x^2 - 5x + 1$, *con* $x_0 = 2$, $x_1 = 4$
9. $f(x) = 4e^{-0.8x-0.5} - 7$, *con* $x_0 = 0$, $x_0 = 0.05$, $x_1 = 0.35$
10. $f(x) = -3e^{-1.2x+0.4} + 2$, *con* $x_0 = 0$, $x_1 = 0.3$, $x_1 = 0.8$
11. $f(x) = \sin(x) - \frac{x}{3}$, *con* $x_0 = 0$, $x_1 = 1$
12. $f(x) = (3x + 3.1)\cos(2x + \pi/6)$, *con*
13. $f(x) = (-2x + 1.1)\sin(5x - \pi/3)$, *con*

**Formato sugerido de tabla:**

| $k$ | $x_{k-1}$ | $x_k$ | $f(x_k)$ | $x_{k+1}$ | $e_k = \|x_{k+1} - x_k\|$ |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| ... | | | | | |

Cuadro 4.7: Iteraciones del método de la secante para $f(x)$.

### Método de Newton

**Instrucciones:** Resuelve los siguientes ejercicios aplicando el **método de Newton**. Para cada función $f(x)$:

1. Verifica que exista un intervalo $[a, b]$ donde $f(a) \cdot f(b) < 0$.
2. Realiza al menos 8 iteraciones del método.
3. Registra los valores $x_k, f(x_k), f'(x_k)$ y el error $|x_{k+1} - x_k|$.
4. Grafica la función y marca el punto donde converge la raíz.

**Ejercicio 11.**

1. $f(x) = x^3 - 7x + 6$, $[a, b] = [0, 1]$
2. $f(x) = x^3 + 2x^2 - 5$, $[a, b] = [1, 2]$
3. $f(x) = x^4 - 3x^3 + 2$, $[a, b] = [0, 1]$
4. $f(x) = \sin(x) - \dfrac{x}{3}$, $[a, b] = [0, 2]$
5. $f(x) = e^{-x} - x$, $[a, b] = [0, 1]$
6. $f(x) = x^3 - 2x - 5$, $[a, b] = [2, 3]$
7. $f(x) = x^5 - 3x + 1$, $[a, b] = [0, 1]$

8. $f(x) = \cos(x) - 2x, \quad [a,b] = [0,1]$
9. $f(x) = \ln(x+2) + x - 1, \quad [a,b] = [-1,0]$
10. $f(x) = x\,e^{-x} - 0.1, \quad [a,b] = [0,1]$
11. $f(x) = x^3 - 4\sin(x), \quad [a,b] = [1,2]$
12. $f(x) = e^x - 3x^2, \quad [a,b] = [0,1]$
13. $f(x) = x^2 - \cos(x), \quad [a,b] = [0,1]$
14. $f(x) = x^3 + 4x^2 - 10, \quad [a,b] = [1,2]$
15. $f(x) = 2x\,\sin(x) - 1, \quad [a,b] = [0,1]$

**Formato sugerido de tabla para Newton:**

| $k$ | $x_k$ | $f(x_k)$ | $f'(x_k)$ | $\lvert x_{k+1} - x_k \rvert$ |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |
| 9 | | | | |
| 10 | | | | |

Cuadro 4.8: Iteraciones del método de Newton para $f(x)$.

### Método del Punto Fijo

**Instrucciones:** Resuelve los siguientes ejercicios aplicando el **método del punto fijo**.

1. Reescriba la ecuación $f(x) = 0$ en forma $x = g(x)$.
2. Verifique que la función $g(x)$ cumpla la condición de convergencia local $|g'(x)| < 1$ en el intervalo considerado.
3. Realice al menos 8 iteraciones partiendo de un valor inicial $x_0$.
4. Registre los valores $x_k, g(x_k), |x_{k+1} - x_k|$.
5. Grafique $y = g(x)$ y $y = x$ para visualizar el punto de intersección.

**Ejercicio 12.**

1. $f(x) = \cos(x) - x$, *Sugerencia:* $g(x) = \cos(x), \quad x_0 = 0.5$
2. $f(x) = x^3 + x - 1$, *Sugerencia:* $g(x) = 1 - x^3, \quad x_0 = 0.6$
3. $f(x) = e^{-x} - x$, *Sugerencia:* $g(x) = e^{-x}, \quad x_0 = 0.7$
4. $f(x) = x^2 - 4x + 1$, *Sugerencia:* $g(x) = \dfrac{x^2+1}{4}, \quad x_0 = 1$
5. $f(x) = x^3 - 2x - 5$ , *Sugerencia:* $g(x) = \sqrt[3]{2x+5}, \quad x_0 = 2$

6. $f(x) = x^2 - e^x + 2$, *Sugerencia:* $g(x) = \sqrt{e^x - 2}$, $x_0 = 0.5$
7. $f(x) = \ln(x+1) + x - 2$, *Sugerencia:* $g(x) = 2 - \ln(x+1)$, $x_0 = 0.5$
8. $f(x) = x^3 - 3x + 1$, *Sugerencia:* $g(x) = \sqrt[3]{3x-1}$, $x_0 = 0.8$
9. $f(x) = x - \sin(x) - 1$, *Sugerencia:* $g(x) = \sin(x) + 1$, $x_0 = 1$
10. $f(x) = x^2 - 3$, *Sugerencia:* $g(x) = \dfrac{3}{x}$, $x_0 = 1.5$
11. $f(x) = 2x - e^{-x}$, *Sugerencia:* $g(x) = \dfrac{e^{-x}}{2}$, $x_0 = 0$
12. $f(x) = x^3 + 4x^2 - 10$, *Sugerencia:* $g(x) = \sqrt{\dfrac{10 - x^3}{4}}$, $x_0 = 1.5$
13. $f(x) = x^2 - \cos(x)$, *Sugerencia:* $g(x) = \sqrt{\cos(x)}$, $x_0 = 0.5$
14. $f(x) = x - e^{-x^2}$, *Sugerencia:* $g(x) = e^{-x^2}$, $x_0 = 0.5$
15. $f(x) = x^3 - 2$, *Sugerencia:* $g(x) = \sqrt[3]{2}$, $x_0 = 1$

**Formato sugerido de tabla para el método del punto fijo:**

| $k$ | $x_k$ | $g(x_k)$ | $\lvert x_{k+1} - x_k \rvert$ |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |

Cuadro 4.9: Iteraciones del método del punto fijo para $f(x) = 0$.

## 4.6. Para impresión

### 4.6.1. Punto fijo

Pseudocódigo

```
Algoritmo: M\'etodo del Punto Fijo

Entrada:
    g(x)        --> funci\'on de iteraci\'on
    x0          --> valor inicial
    tol         --> tolerancia (por ejemplo, 1e-6)
    iter_max    --> n\'umero m\'aximo de iteraciones

Proceso:
    iter <-- 0
    error <-- 1

    Mientras (error > tol) Y (iter < iter_max) Hacer
        x1 <-- g(x0)
        error <-- |x1 - x0|
        x0 <-- x1
        iter <-- iter + 1
    FinMientras

Salida:
    Imprimir "Ra\'iz aproximada:", x1
    Imprimir "Iteraciones:", iter
FinAlgoritmo
```

Implementación numérica

```
\begin{lstlisting}[language=R, caption={M\'etodo del Punto Fijo en R}]
fijo <- function(g, x0, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1

  while(error > tol && iter < iter_max){
    x1 <- g(x0)
    error <- abs(x1 - x0)
    x0 <- x1
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x1, "\n")
  cat("Iteraciones:", iter, "\n")
```

```
  return(x1)
}

# Ejemplo de uso:
g <- function(x) cos(x)
fijo(g, x0 = 0.5)
\end{lstlisting}
```

### 4.6.2. Newton-Raphson

Pseudocódigo

```
Algoritmo: M\'etodo de Newton{Raphson

Entrada:
    f(x)        --> funci\'on
    f'(x)       --> derivada de f(x)
    x0          --> valor inicial
    tol         --> tolerancia
    iter_max    --> n\'umero m\'aximo de iteraciones

Proceso:
    iter <-- 0
    error <-- 1

    Mientras (error > tol) Y (iter < iter_max) Hacer
        x1 <-- x0 - f(x0)/f'(x0)
        error <-- |x1 - x0|
        x0 <-- x1
        iter <-- iter + 1
    FinMientras

Salida:
    Imprimir "Ra\'iz aproximada:", x1
    Imprimir "Iteraciones:", iter
FinAlgoritmo
```

Implementación numérica

```
newton <- function(f, df, x0, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1
```

```
  while(error > tol && iter < iter_max){
    x1 <- x0 - f(x0)/df(x0)
    error <- abs(x1 - x0)
    x0 <- x1
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x1, "\n")
  cat("Iteraciones:", iter, "\n")
  return(x1)
}

# Ejemplo:
f  <- function(x) x^2 - 2
df <- function(x) 2*x
newton(f, df, x0 = 1)
```

### 4.6.3. Bisección

Pseudocódigo

```
Algoritmo: M\'etodo de Bisecci\'on

Entrada:
    f(x)        --> funci\'on continua
    a, b        --> intervalo [a,b] con f(a)*f(b) < 0
    tol         --> tolerancia
    iter_max    --> n\'umero m\'aximo de iteraciones

Proceso:
    Si f(a)*f(b) > 0 Entonces
        Imprimir "No hay cambio de signo"
        Salir
    FinSi

    iter <-- 0
    error <-- |b - a|

    Mientras (error > tol) Y (iter < iter_max) Hacer
        c <-- (a + b)/2

        Si f(a)*f(c) < 0 Entonces
            b <-- c
```

```
        Sino
            a <-- c
        FinSi

        error <-- |b - a|
        iter <-- iter + 1
    FinMientras

Salida:
    Imprimir "Ra\'iz aproximada:", c
    Imprimir "Iteraciones:", iter
FinAlgoritmo
```

Implementación numérica

```
biseccion <- function(f, a, b, tol = 1e-6, iter_max = 100){
  if(f(a)*f(b) > 0){
    cat("Error: no hay cambio de signo en [a,b].\n")
    return(NA)
  }

  iter <- 0
  error <- abs(b - a)

  while(error > tol && iter < iter_max){
    c <- (a + b)/2
    if(f(a)*f(c) < 0){
      b <- c
    } else {
      a <- c
    }
    error <- abs(b - a)
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", c, "\n")
  cat("Iteraciones:", iter, "\n")
  return(c)
}

# Ejemplo:
f <- function(x) x^3 - x - 2
biseccion(f, a = 1, b = 2)
```

### 4.6.4. Secante

Pseudocódigo

```
Algoritmo: M\'etodo de la Secante

Entrada:
    f(x)        --> funci\'on
    x0, x1      --> valores iniciales
    tol         --> tolerancia
    iter_max    --> n\'umero m\'aximo de iteraciones

Proceso:
    iter <-- 0
    error <-- 1

    Mientras (error > tol) Y (iter < iter_max) Hacer
        fx0 <-- f(x0)
        fx1 <-- f(x1)

        Si |fx1 - fx0| < 1e-12 Entonces
            Imprimir "Error: divisi\'on por cero"
            Salir
        FinSi

        x2 <-- x1 - fx1 * (x1 - x0) / (fx1 - fx0)
        error <-- |x2 - x1|

        x0 <-- x1
        x1 <-- x2
        iter <-- iter + 1
    FinMientras

Salida:
    Imprimir "Ra\'iz aproximada:", x2
    Imprimir "Iteraciones:", iter
FinAlgoritmo
```

Implementación numérica

```
secante <- function(f, x0, x1, tol = 1e-6, iter_max = 100){
  iter <- 0
  error <- 1

  while(error > tol && iter < iter_max){
    fx0 <- f(x0)
    fx1 <- f(x1)
```

```
    if(abs(fx1 - fx0) < 1e-12){
      cat("Error: Divisi\'on por cero (f(x1) - f(x0) approxx 0)\n")
      return(NA)
    }

    x2 <- x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    error <- abs(x2 - x1)

    x0 <- x1
    x1 <- x2
    iter <- iter + 1
  }

  cat("Ra\'iz aproximada:", x2, "\n")
  cat("Iteraciones:", iter, "\n")
  return(x2)
}

# Ejemplo:
f <- function(x) x^3 - 2*x^2 + 3*x - 5
secante(f, x0 = 1, x1 = 2)
```

## 4.7. Tarea para el portafolio

**A continuación se presentan las indicaciones sobre los ejercicios que se van a a incluir en el portafolio del curso de Métodos Numéricos**

1. Del método de Bisección, (subsección 4.5) del ejercicio 1, resolver de los numerales impares: 3 ejercicios *a mano* y 5 ejercicios con el *código automatizado*.

2. Del método de la Secante (subsección 4.5.1), del ejercicio 2, resolver de los ejercicios impares 3: *a mano* y de los ejercicios pares 5: con el *código automatizado*.

3. De la subsección 4.5.1, Del método de Newton (subsección 4.5.1), del ejercicio 3, resolver de los ejercicios pares: 3 *a mano* y de los ejercicios impares 5: con el *código automatizado*.

4. De la subsección 4.5.1, Del método del Punto Fijo (subsección 4.5.1), del ejercicio 4, resolver de los: 3 *a mano* y 5 con el *código automatizado*.

# Capítulo 5

# Interpolación Numérica

## 5.1. Conceptos Fundamentales

**Definición 11.** *La interpolación es una técnica numérica utilizada para estimar valores desconocidos de una función a partir de un conjunto de puntos conocidos. A diferencia del ajuste de curvas, la interpolación busca una función que pase exactamente por los puntos dados. La interpolación consiste en estimar el valor de una función dentro del intervalo cubierto por un conjunto de datos conocidos.*

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n) \tag{5.1}$$

*donde se asume que $y_i = f(x_i)$ para $i = 0, 1, \ldots, n$. El objetivo es construir una función $P(x)$, generalmente un polinomio, tal que:*

$$P(x_i) = f(x_i) = y_i, \quad i = 0, 1, \ldots, n. \tag{5.2}$$

*Dado que existen $n + 1$ puntos distintos $(x_i, y_i)$, se puede demostrar que existe un único polinomio de grado $n$ que los interpola:*

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

*La determinación de los coeficientes $a_i$ puede hacerse de diversas maneras, dando origen a los distintos métodos de interpolación.*

**Nota 22.** *En la interpolación, la función construida pasa exactamente por los puntos dados; en el ajuste de curvas, se busca una función que aproxime los datos minimizando un error global. Si el polinomio interpolante se usa para estimar valores de $f(x)$ fuera del intervalo cubierto por los puntos dados, el proceso se denomina* ***extrapolación****. La extrapolación extiende esta idea a valores fuera del intervalo, con menor confiabilidad, tiende a generar errores grandes, por lo que debe usarse con precaución.*

**Comparación entre interpolación y ajuste de curvas:**

| **Interpolación** | **Ajuste de Curvas** |
|---|---|
| La curva pasa exactamente por todos los puntos. | La curva se aproxima a los puntos, pero no necesariamente pasa por todos. |
| Se usa cuando los datos son exactos o tabulados sin error experimental. | Se usa cuando los datos contienen ruido o errores de medición. |
| Ejemplo: tablas de logaritmos o funciones trigonométricas. | Ejemplo: datos experimentales en laboratorio. |

**Nota 23.** *El error de interpolación mide la diferencia entre el valor real de la función y el valor estimado por el polinomio interpolante. Se expresa mediante la fórmula:*

$$R_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \tag{5.3}$$

*para algún $\xi$ en el intervalo de interpolación.*

*El error cometido al aproximar $f(x)$ mediante el polinomio interpolante $P_n(x)$ se expresa como:*

$$R_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i),$$

*donde $\xi$ es algún valor dentro del intervalo $(x_0, x_n)$. El error depende de la derivada de orden $(n+1)$ de la función $f(x)$ y de la distribución de los nodos $x_i$.*

**Nota 24.** ***Problemas mal condicionados (efecto Runge):*** *El uso de polinomios de alto grado con nodos equiespaciados puede producir oscilaciones extremas cerca de los extremos del intervalo. Cuando los nodos $x_i$ están equiespaciados y se incrementa el número de puntos, el polinomio interpolante puede oscilar significativamente, especialmente cerca de los extremos del intervalo. Este fenómeno se conoce como* ***efecto Runge****. Una forma de mitigarlo es usar nodos distribuidos según los* ***polinomios de Chebyshev*** *o recurrir a métodos por tramos como los*

## 5.2. Forma General del Polinomio Interpolante

Se define el polinomio de grado $n$ que pasa por $n+1$ puntos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

$$P_n(x_i) = y_i, \quad i = 0, 1, 2, \ldots, n$$

donde se conoce el valor de la función $f(x)$ en cada punto, es decir, $y_i = f(x_i)$. El **problema de interpolación polinómica** consiste en determinar un polinomio $P_n(x)$ de grado menor o igual a $n$, la forma general del polinomio interpolante es:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

donde los coeficientes $a_0, a_1, \ldots, a_n$ se determinan imponiendo las condiciones de interpolación:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1 \\ \quad \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n \end{cases}$$

En forma matricial:

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}}_{V} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}}_{\vec{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\vec{y}}$$

donde $V$ es la **matriz de Vandermonde**. La existencia de una solución única depende de que $V$ sea invertible, lo cual se cumple siempre que todos los $x_i$ sean distintos. Para demostrar que el polinomio interpolante es único, consideremos dos polinomios $P_n(x)$ y $Q_n(x)$ que ambos interpola los mismos puntos:

$$P_n(x_i) = Q_n(x_i) = y_i, \quad i = 0, 1, \ldots, n.$$

se define la función:

$$R(x) = P_n(x) - Q_n(x).$$

Entonces $R(x)$ es también un polinomio de grado a lo sumo $n$, y satisface:

$$R(x_i) = P_n(x_i) - Q_n(x_i) = 0, \quad \text{para todos } i = 0, 1, \ldots, n.$$

Por lo tanto, $R(x)$ tiene $n+1$ raíces distintas $x_0, x_1, \ldots, x_n$. Sin embargo, un polinomio de grado $n$ no puede tener más de $n$ raíces distintas, a menos que todos sus coeficientes sean cero. Así,

$$R(x) \equiv 0 \quad \Longrightarrow \quad P_n(x) = Q_n(x),$$

lo que demuestra que el polinomio interpolante es **único**.

**Nota 25.**

- *Para $n+1$ puntos distintos, siempre existe un polinomio de grado $n$ que pasa exactamente por ellos.*
- *El cálculo directo mediante la matriz de Vandermonde es exacto teóricamente, pero numéricamente inestable para grandes $n$ debido a la mala condición de la matriz.*
- *Por esta razón, en la práctica se prefieren formas más estables del polinomio, como las fórmulas de **Lagrange** o **Newton**.*

**Ejemplo 44.** *Supóngase que se tienen los puntos:*

$$(1,2), \quad (2,3), \quad (4,7).$$

*El polinomio de grado 2 tendrá la forma:*

$$P_2(x) = a_0 + a_1 x + a_2 x^2.$$

*Sustituyendo los puntos:*

$$\begin{cases} a_0 + a_1(1) + a_2(1)^2 = 2 \\ a_0 + a_1(2) + a_2(2)^2 = 3 \\ a_0 + a_1(4) + a_2(4)^2 = 7 \end{cases}$$

*Al resolver el sistema, se obtiene:*

$$a_0 = 1, \quad a_1 = 0.5, \quad a_2 = 0.25.$$

*Por tanto,*

$$P_2(x) = 1 + 0.5x + 0.25x^2.$$

*Verificando:*

$$P_2(1) = 2, \quad P_2(2) = 3, \quad P_2(4) = 7,$$

*lo que confirma que el polinomio interpola correctamente los puntos.*

**Ejemplo 45.** *Interpolar los puntos:*

$$(0,1), \quad (1,0), \quad (2,-1), \quad (3,2).$$

*Buscamos un polinomio cúbico:*

$$P_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

*Sustituyendo:*

$$\begin{cases} a_0 = 1, \\ a_0 + a_1 + a_2 + a_3 = 0, \\ a_0 + 2a_1 + 4a_2 + 8a_3 = -1, \\ a_0 + 3a_1 + 9a_2 + 27a_3 = 2. \end{cases}$$

*De la primera ecuación,* $a_0 = 1$. *Sustituyendo en las demás:*

$$\begin{cases} a_1 + a_2 + a_3 = -1, \\ 2a_1 + 4a_2 + 8a_3 = -2, \\ 3a_1 + 9a_2 + 27a_3 = 1. \end{cases}$$

*Resolviendo el sistema:*

$$a_1 = -\tfrac{5}{2}, \quad a_2 = \tfrac{9}{4}, \quad a_3 = -\tfrac{3}{4}.$$

*El polinomio interpolante es:*

$$P_3(x) = 1 - \frac{5}{2}x + \frac{9}{4}x^2 - \frac{3}{4}x^3.$$

*Verificando:*

$$P_3(0) = 1, \quad P_3(1) = 0, \quad P_3(2) = -1, \quad P_3(3) = 2.$$

***Interpretación:*** *este polinomio de grado tres conecta los cuatro puntos de manera exacta. Si se grafican los datos y la curva, se observa un comportamiento ondulado típico de polinomios de orden alto, mostrando cómo la interpolación exacta puede oscilar entre los nodos.*

**Nota 26.**
- *Para $n+1$ puntos distintos, existe un único polinomio de grado $n$ que interpola los datos.*
- *El cálculo directo de los coeficientes mediante la matriz de Vandermonde es teóricamente exacto, pero numéricamente inestable para valores grandes de $n$.*
- *Por eficiencia y estabilidad se prefieren métodos como los de **Lagrange** y **Newton**, que se estudiarán a continuación.*

## 5.3. Interpolación de Lagrange

Dado un conjunto de $n+1$ puntos con abscisas distintas

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n), \qquad y_i = f(x_i),$$

el polinomio interpolante de Lagrange $P_n(x)$ se escribe como

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

Cada base $L_i$ satisface $L_i(x_k) = \delta_{ik}$, por lo que $P_n(x_i) = y_i$ para todo $i$.

Si $f$ es $(n+1)$ veces derivable en un intervalo que contiene a los nodos $x_i$ y al punto $x$, entonces

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,\omega_{n+1}(x), \text{ donde } \omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i),$$

para algún $\xi$ en el intervalo convexo de $\{x_0, \ldots, x_n, x\}$.

**Nota 27.**
- *Construir $P_n(x)$ y evaluarlo directamente con la fórmula de Lagrange cuesta $O(n^2)$ por punto de evaluación.*

- *Para muchos nodos o evaluaciones repetidas, se prefiere la forma* baricéntrica *por estabilidad y costo $O(n)$ por evaluación (se comenta al final).*
- *Con nodos equiespaciados y n grande puede aparecer el* efecto Runge. *Una mitigación es usar nodos de Chebyshev o splines.*

**Ejemplo 46.** *Interpolar $(0,0)$, $(1,1)$, $(2,4)$ (muestran $f(x)=x^2$). Se obtiene*

$$\begin{aligned} L_0(x) &= \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{(x-1)(x-2)}{2}, \\ L_1(x) &= \frac{(x-0)(x-2)}{(1-0)(1-2)} = -(x)(x-2), \\ L_2(x) &= \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{x(x-1)}{2}. \end{aligned}$$

*Entonces*

$$P_2(x) = 0 \cdot L_0(x) + 1 \cdot L_1(x) + 4 \cdot L_2(x) = x^2,$$

*que recupera exactamente la función cuadrática.*

## 5.4. Interpolación de Newton–Diferencias Finitas

El método de Newton construye el polinomio interpolante de manera incremental, usando el concepto de **diferencias divididas**. Esto permite agregar nuevos puntos sin recalcular todo el polinomio.

Sea un conjunto de $n+1$ puntos distintos:

$$(x_0, y_0),\ (x_1, y_1),\ \ldots,\ (x_n, y_n).$$

El polinomio de Newton tiene la forma:

$$\begin{aligned} P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ &+ f[x_0, x_1, \ldots, x_n] \prod_{j=0}^{n-1} (x - x_j), \end{aligned}$$

donde los coeficientes $f[x_i, \ldots, x_j]$ son las *diferencias divididas.*

- De primer orden:

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}.$$

- De segundo orden:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}.$$

- En general:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

Estas diferencias se organizan en una tabla triangular.

**Ejemplo 47.** *Interpolar los datos:*

$$(1, 2),\ (2, 3),\ (4, 7).$$

***Paso 1:*** *construir la tabla de diferencias divididas:*

| $x_i$ | $f[x_i]$ | $f[x_i, x_{i+1}]$ | $f[x_i, x_{i+1}, x_{i+2}]$ |
|---|---|---|---|
| *1* | *2* | $\frac{3-2}{2-1} = 1$ | $\frac{(7-3)/(4-2)-1}{4-1} = \frac{1-1}{3} = 0$ |
| *2* | *3* | $\frac{7-3}{4-2} = 2$ | |
| *4* | *7* | | |

***Paso 2:*** *polinomio de Newton:*

$$P_2(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1),$$

*sustituyendo:*

$$P_2(x) = 2 + 1(x - 1) + 0(x - 1)(x - 2) = 2 + (x - 1) = x + 1.$$

*En este caso el término cuadrático se anuló (el polinomio resultante es lineal porque los puntos están casi alineados).*

**Ejemplo 48.** *Sea:*

$$(0, 1),\ (1, 0),\ (2, -1),\ (3, 2).$$

***Tabla de diferencias divididas:***

| $x_i$ | $f[x_i]$ | $f[x_i, x_{i+1}]$ | $f[x_i, x_{i+1}, x_{i+2}]$ | $f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$ |
|---|---|---|---|---|
| *0* | *1* | $\frac{0-1}{1-0} = -1$ | $\frac{(-1)-(-1)}{2-0} = 0$ | $\frac{(1.5)-0}{3-0} = 0.5$ |
| *1* | *0* | $\frac{-1-0}{2-1} = -1$ | $\frac{2-(-1)}{3-1} = 1.5$ | |
| *2* | *-1* | $\frac{2-(-1)}{3-2} = 3$ | | |
| *3* | *2* | | | |

***Polinomio:***

$$\begin{aligned} P_3(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 1 - 1(x - 0) + 0(x)(x - 1) + 0.5(x)(x - 1)(x - 2). \end{aligned}$$

*Simplificando:*

$$P_3(x) = 1 - x + 0.5x(x - 1)(x - 2) = 1 - x + 0.5x^3 - 1.5x^2 + x = 1 - 1.5x^2 + 0.5x^3.$$

*Por tanto:*

$$P_3(x) = 1 - \frac{3}{2}x^2 + \frac{1}{2}x^3.$$

*Verificando:*

$$P_3(0) = 1, \quad P_3(1) = 0, \quad P_3(2) = -1, \quad P_3(3) = 2.$$

**Nota 28.**
- *Las diferencias divididas se pueden calcular una sola vez y reutilizar para múltiples evaluaciones.*
- *Si se añade un nuevo punto $(x_{n+1}, y_{n+1})$, basta con agregar una nueva columna a la tabla sin recomputar todo.*
- *El polinomio de Newton es más estable que el de Lagrange para grandes $n$ y se adapta mejor a la incorporación incremental de datos.*

**Ejemplo 49.** *Dados los puntos*

$$(1, 2), \quad (2, 3), \quad (4, 7),$$

*construyamos la tabla de diferencias divididas y el polinomio de Newton.*

**Paso 1: Primera columna (valores de la funcion).**

$$f[x_0] = 2 \ (en\ x_0 = 1), \qquad f[x_1] = 3 \ (en\ x_1 = 2), \qquad f[x_2] = 7 \ (en\ x_2 = 4).$$

**Paso 2: Diferencias divididas de primer orden.**

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{3-2}{2-1} = 1,$$
$$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{7-3}{4-2} = 2.$$

**Paso 3: Diferencia dividida de segundo orden.**

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{2-1}{4-1} = \frac{1}{3}.$$

***Atencion:*** *En el ejemplo del texto principal se obtuvo cero por un redondeo al mostrar pasos; aquí dejamos el valor exacto $\frac{1}{3}$. Para verificar coherencia, construyamos el polinomio y validemos en los nodos.*

**Paso 4: Polinomio de Newton.**

$$\begin{aligned} P_2(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &= 2 + 1(x-1) + \frac{1}{3}(x-1)(x-2). \end{aligned}$$

**Paso 5: Verificacion en los nodos.**

$$P_2(1) = 2 + 1(0) + \tfrac{1}{3}(0)(-1) = 2,$$
$$P_2(2) = 2 + 1(1) + \tfrac{1}{3}(1)(0) = 3,$$
$$P_2(4) = 2 + 1(3) + \tfrac{1}{3}(3)(2) = 2 + 3 + 2 = 7.$$

**Paso 6: Evaluacion en un punto intermedio (por ejemplo, $x = 3$).**

$$P_2(3) = 2 + 1(2) + \tfrac{1}{3}(2)(1) = 2 + 2 + \tfrac{2}{3} = \tfrac{14}{3}.$$

**Tabla triangular resumida.**

| $x_i$ | $f[x_i]$ | $f[x_i, x_{i+1}]$ | $f[x_i, x_{i+1}, x_{i+2}]$ |
|---|---|---|---|
| 1 | 2 | 1 | $\frac{1}{3}$ |
| 2 | 3 | 2 | |
| 4 | 7 | | |

## 5.5. Polinomio de Newton

Sea una función $f$ conocida en los puntos

$$(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n), \qquad f_k = f(x_k).$$

El polinomio interpolante de Newton en términos de diferencias divididas se escribe como

$$P_n(x) = f[x_0] + (x - x_0) f[x_0, x_1] + (x - x_0)(x - x_1) f[x_0, x_1, x_2] + \cdots + (x - x_0) \cdots (x - x_{n-1}) f[x_0, \ldots, x_n]. \tag{5.4}$$

Recordemos que las *diferencias divididas* se definen recursivamente:

$$\begin{aligned} f[x_k] &= f(x_k), \\ f[x_k, x_{k+1}] &= \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}, \\ f[x_k, x_{k+1}, x_{k+2}] &= \frac{f[x_{k+1}, x_{k+2}] - f[x_k, x_{k+1}]}{x_{k+2} - x_k}, \\ &\vdots \end{aligned}$$

En esta sección supondremos que los nodos están *equispaciados*, es decir:

$$x_k = x_0 + kh, \qquad k = 0, 1, \ldots, n,$$

para cierto paso $h > 0$ constante. Definimos primero las *diferencias progresivas* de $f$ en los nodos $x_k$:

$$\begin{aligned} \Delta f_k &= f_{k+1} - f_k, \\ \Delta^2 f_k &= \Delta f_{k+1} - \Delta f_k, \\ \Delta^3 f_k &= \Delta^2 f_{k+1} - \Delta^2 f_k, \\ &\vdots \end{aligned}$$

es decir:

$$\begin{aligned}
\Delta f_0 &= f_1 - f_0, \\
\Delta^2 f_0 &= (f_2 - f_1) - (f_1 - f_0) = f_2 - 2f_1 + f_0, \\
\Delta^3 f_0 &= (f_3 - 2f_2 + f_1) - (f_2 - 2f_1 + f_0) \\
&= f_3 - 3f_2 + 3f_1 - f_0, \\
&\vdots
\end{aligned}$$

Para el primer orden tenemos:

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}.$$

Como $x_1 - x_0 = h$ y $f_1 - f_0 = \Delta f_0$, se sigue que

$$f[x_0, x_1] = \frac{\Delta f_0}{h}. \tag{5.5}$$

Por definición de diferencias divididas de segundo orden:

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}.$$

Usamos el caso de primer orden para $f[x_1, x_2]$ y $f[x_0, x_1]$:

$$f[x_1, x_2] = \frac{f_2 - f_1}{h}, \qquad f[x_0, x_1] = \frac{f_1 - f_0}{h}.$$

Entonces:

$$\begin{aligned}
f[x_0, x_1, x_2] &= \frac{\frac{f_2 - f_1}{h} - \frac{f_1 - f_0}{h}}{x_2 - x_0} \\
&= \frac{\frac{f_2 - 2f_1 + f_0}{h}}{2h} = \frac{f_2 - 2f_1 + f_0}{2h^2}.
\end{aligned}$$

Observamos que $f_2 - 2f_1 + f_0 = \Delta^2 f_0$, por lo que

$$f[x_0, x_1, x_2] = \frac{\Delta^2 f_0}{2!\, h^2}. \tag{5.6}$$

Procediendo de manera análoga llegamos a la expresión general:

$$f[x_0, x_1, \ldots, x_k] = \frac{\Delta^k f_0}{k!\, h^k}, \qquad k = 0, 1, \ldots, n. \tag{5.7}$$

Para $k = 0$ se tiene $f[x_0] = f_0$, y como $\Delta^0 f_0 = f_0$, la igualdad se cumple. Para $k = 1$ y $k = 2$ la fórmula se verificó explícitamente en 5.5 y 5.6.

Supongamos que

$$f[x_0, \ldots, x_k] = \frac{\Delta^k f_0}{k!\, h^k}, \qquad f[x_1, \ldots, x_{k+1}] = \frac{\Delta^k f_1}{k!\, h^k}.$$

entonces consideremos

$$f[x_0, x_1, \ldots, x_{k+1}] = \frac{f[x_1, \ldots, x_{k+1}] - f[x_0, \ldots, x_k]}{x_{k+1} - x_0}.$$

Como $x_{k+1} - x_0 = (k+1)h$, usando la hipótesis:

$$f[x_1, \ldots, x_{k+1}] = \frac{\Delta^k f_1}{k!\, h^k}, \qquad f[x_0, \ldots, x_k] = \frac{\Delta^k f_0}{k!\, h^k},$$

por tanto se obtiene

$$f[x_0, \ldots, x_{k+1}] \;=\; \frac{\frac{\Delta^k f_1}{k!\, h^k} - \frac{\Delta^k f_0}{k!\, h^k}}{(k+1)h} = \frac{\Delta^k f_1 - \Delta^k f_0}{k!\, h^k\, (k+1)h}.$$

Por definición de diferencia progresiva de orden $k+1$:

$$\Delta^{k+1} f_0 = \Delta^k f_1 - \Delta^k f_0.$$

es decir

$$f[x_0, \ldots, x_{k+1}] = \frac{\Delta^{k+1} f_0}{(k+1)!\, h^{k+1}},$$

que es precisamente la fórmula (5.7) para $k+1$, y por lo tanto queda demostrada la relación

$$f[x_0, \ldots, x_k] = \frac{\Delta^k f_0}{k!\, h^k}, \quad k = 0, 1, \ldots, n,$$

cuando los nodos son equiespaciados. Ahora, a partir de la expresión general 5.4 y sustituimos las diferencias divididas usando 5.7:

$$\begin{aligned} P_n(x) &= f_0 + (x - x_0)\frac{\Delta f_0}{1!\, h} + (x - x_0)(x - x_1)\frac{\Delta^2 f_0}{2!\, h^2} + \cdots \\ &+ (x - x_0)(x - x_1)\cdots(x - x_{n-1})\frac{\Delta^n f_0}{n!\, h^n}. \end{aligned}$$

donde

$$x_k = x_0 + kh, \text{ por lo tanto } x - x_k = x - x_0 - kh.$$

haciendo

$$s = \frac{x - x_0}{h} \text{ se tiene que } x - x_0 = sh,$$

y entonces

$$\begin{aligned} x - x_1 &= x - (x_0 + h) = (x - x_0) - h = (s-1)h, \\ x - x_2 &= x - (x_0 + 2h) = (x - x_0) - 2h = (s-2)h, \\ &\vdots \\ x - x_{k-1} &= (s - (k-1))h. \end{aligned}$$

Por lo tanto, el producto

$$(x-x_0)(x-x_1)\cdots(x-x_{k-1}) = h^k\, s(s-1)\cdots(s-k+1).$$

Sustituyendo en $P_n(x)$:

$$\begin{aligned} P_n(x) &= f_0 + \frac{hs}{h}\Delta f_0 + \frac{h^2 s(s-1)}{2!\,h^2}\Delta^2 f_0 + \cdots + \frac{h^n s(s-1)\cdots(s-n+1)}{n!\,h^n}\Delta^n f_0 \\ &= f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\,\Delta^2 f_0 + \cdots + \frac{s(s-1)\cdots(s-n+1)}{n!}\,\Delta^n f_0. \end{aligned}$$

Es decir, se tiene la fórmula de Newton con diferencias progresivas:

$$P_n(x) = f_0 + s\,\Delta f_0 + \frac{s(s-1)}{2!}\,\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\,\Delta^3 f_0 + \cdots + \frac{s(s-1)\cdots(s-n+1)}{n!}\,\Delta^n f_0, \tag{5.8}$$

donde

$$s = \frac{x-x_0}{h}.$$

De forma análoga definimos las *diferencias regresivas*:

$$\begin{aligned} \nabla f_k &= f_k - f_{k-1}, \\ \nabla^2 f_k &= \nabla f_k - \nabla f_{k-1}, \\ \nabla^3 f_k &= \nabla^2 f_k - \nabla^2 f_{k-1}, \\ &\vdots \end{aligned}$$

En particular, las primeras diferencias regresivas en el nodo final $x_n$ son:

$$\begin{aligned} \nabla f_n &= f_n - f_{n-1}, \text{ luego} \\ \nabla^2 f_n &= (f_n - f_{n-1}) - (f_{n-1} - f_{n-2}) = f_n - 2f_{n-1} + f_{n-2}, \\ &\vdots \end{aligned}$$

Ahora consideramos las diferencias divididas de atrás hacia adelante:

$$f[x_n],\ f[x_n, x_{n-1}],\ f[x_n, x_{n-1}, x_{n-2}], \ldots$$

Se tiene que

$$f[x_n, x_{n-1}] = \frac{f_n - f_{n-1}}{x_n - x_{n-1}} = \frac{\nabla f_n}{h}.$$

luego,

$$f[x_n, x_{n-1}, x_{n-2}] = \frac{f[x_{n-1}, x_{n-2}] - f[x_n, x_{n-1}]}{x_{n-2} - x_n}.$$

donde

$$f[x_{n-1}, x_{n-2}] = \frac{f_{n-1} - f_{n-2}}{h}, \text{ y } f[x_n, x_{n-1}] = \frac{f_n - f_{n-1}}{h},$$

y $x_{n-2} - x_n = -2h$. Entonces:

$$\begin{aligned} f[x_n, x_{n-1}, x_{n-2}] &= \frac{\frac{f_{n-1}-f_{n-2}}{h} - \frac{f_n - f_{n-1}}{h}}{-2h} = \frac{(f_{n-1} - f_{n-2}) - (f_n - f_{n-1})}{-2h^2} \\ &= \frac{-f_n + 2f_{n-1} - f_{n-2}}{-2h^2} = \frac{f_n - 2f_{n-1} + f_{n-2}}{2h^2} = \frac{\nabla^2 f_n}{2!\, h^2}. \end{aligned}$$

De forma análoga al caso progresivo, se puede probar por inducción que:

$$f[x_n, x_{n-1}, \ldots, x_{n-k}] = \frac{\nabla^k f_n}{k!\, h^k}, \qquad k = 0, 1, \ldots, n. \tag{5.9}$$

Procediendo de manera análoga escribimos ahora el polinomio de Newton de atrás hacia adelante:

$$\begin{aligned} P_n(x) &= f[x_n] + (x - x_n) f[x_n, x_{n-1}] + (x - x_n)(x - x_{n-1}) f[x_n, x_{n-1}, x_{n-2}] \\ &+ \cdots + (x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1}) f[x_n, \ldots, x_{n-k}] + \cdots . \end{aligned}$$

Sustituimos la ecuación (5.9):

$$\begin{aligned} P_n(x) &= f_n + (x - x_n)\frac{\nabla f_n}{1!\, h} + (x - x_n)(x - x_{n-1})\frac{\nabla^2 f_n}{2!\, h^2} + \cdots \\ &+ (x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1})\frac{\nabla^k f_n}{k!\, h^k} + \cdots . \end{aligned}$$

Definimos ahora

$$s = \frac{x - x_n}{h} \text{ entonces } x - x_n = sh.$$

Además,

$$\begin{aligned} x - x_{n-1} &= x - (x_n - h) = (x - x_n) + h = (s+1)h, \\ x - x_{n-2} &= x - (x_n - 2h) = (x - x_n) + 2h = (s+2)h, \\ &\vdots \\ x - x_{n-k+1} &= (s + k - 1)h. \end{aligned}$$

Por lo tanto

$$(x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1}) = h^k\, s(s+1) \cdots (s+k-1).$$

Sustituyendo en $P_n(x)$:

$$\begin{aligned} P_n(x) &= f_n + \frac{hs}{h}\nabla f_n + \frac{h^2 s(s+1)}{2!\, h^2}\nabla^2 f_n + \cdots + \frac{h^k s(s+1)\cdots(s+k-1)}{k!\, h^k}\nabla^k f_n + \cdots \\ &= f_n + s\,\nabla f_n + \frac{s(s+1)}{2!}\,\nabla^2 f_n + \cdots + \frac{s(s+1)\cdots(s+k-1)}{k!}\,\nabla^k f_n + \cdots . \end{aligned}$$

con lo cual se obtiene la fórmula de Newton con diferencias regresivas:

$$P_n(x) = f_n + s\,\nabla f_n + \frac{s(s+1)}{2!}\,\nabla^2 f_n + \frac{s(s+1)(s+2)}{3!}\,\nabla^3 f_n + \cdots + \frac{s(s+1)\cdots(s+n-1)}{n!}\,\nabla^n f_n, \tag{5.10}$$

donde

$$s = \frac{x - x_n}{h}.$$

### 5.5.1. Ejemplos del método de Newton con diferencias finitas

**Ejemplo 50.** *Se quiere aproximar $f(x) = x^2$ en un punto cercano a $x_0 = 0$ usando diferencias progresivas de Newton.*

*Considérense los puntos equiespaciados (con $h = 1$):*

$$x_0 = 0, \quad x_1 = 1, x_2 = 2,$$

*y*

$$f_0 = f(0) = 0, f_1 = f(1) = 1, f_2 = f(2) = 4.$$

*Entonces se tienen la siguiente tabla de diferencias divididas:*

| $k$ | $x_k$ | $f_k$ | *Diferencias* |
|---|---|---|---|
| 0 | 0 | 0 | |
| 1 | 1 | 1 | $\Delta f_0 = f_1 - f_0 = 1 - 0 = 1$ |
| 2 | 2 | 4 | $\Delta f_1 = f_2 - f_1 = 4 - 1 = 3$ |
| | | | $\Delta^2 f_0 = \Delta f_1 - \Delta f_0 = 3 - 1 = 2$ |

*entonces el Polinomio de Newton progresivo se calcula a partir de la fórmula de Newton progresiva:*

$$\begin{aligned} P_2(x) &= f_0 + s\,\Delta f_0 + \frac{s(s-1)}{2!}\,\Delta^2 f_0, \\ s &= \frac{x - x_0}{h}. \end{aligned}$$

*Con $x_0 = 0$ y $h = 1$, resulta $s = x$, sustituyendo:*

$$P_2(x) = 0 + s(1) + \frac{s(s-1)}{2}(2) = s + s(s-1) = s + s^2 - s = s^2.$$

*Como $s = x$, se tiene $P_2(x) = x^2$, es decir, recuperamos la función exacta (como era de esperar al interpolar un polinomio de grado 2 con tres puntos). Evaluando en $x = 1.5$: $s = \frac{x - x_0}{h} = \frac{1.5 - 0}{1} = 1.5$, $P_2(1.5) = (1.5)^2 = 2.25$. El valor exacto de $f(1.5)$ es también 2.25, de modo que no hay error de interpolación en este caso.*

**Ejemplo 51.** *Aproximar* $f(x) = \ln(1+x)$ *cerca de* $x_0 = 0$ *usando diferencias progresivas con* $h = 0.5$*, y luego se evaluará en* $x = 0.75$*.*

*Considérense los puntos equiespaciados:*

$$x_0 = 0, \quad x_1 = 0.5, \quad x_2 = 1.0, \qquad h = 0.5.$$

*Evaluando la función:*

$$\begin{aligned} f_0 &= \ln(1+0) = 0, \\ f_1 &= \ln(1+0.5) = \ln(1.5) \approx 0.405465, \\ f_2 &= \ln(1+1.0) = \ln(2) \approx 0.693147. \end{aligned}$$

*entonces la Tabla de diferencias progresivas quedaría como:*

| $k$ | $x_k$ | $f_k$ | *Diferencias* |
|---|---|---|---|
| 0 | 0.0 | 0.000000 | |
| 1 | 0.5 | 0.405465 | $\Delta f_0 = 0.405465 - 0.000000 = 0.405465$ |
| 2 | 1.0 | 0.693147 | $\Delta f_1 = 0.693147 - 0.405465 = 0.287682$ |
| | | | $\Delta^2 f_0 = \Delta f_1 - \Delta f_0 = 0.287682 - 0.405465 = -0.117783$ |

*Entonces el polinomio de Newton progresivo se obtiene con la fórmula:*

$$\begin{aligned} P_2(x) &= f_0 + s\,\Delta f_0 + \frac{s(s-1)}{2!}\,\Delta^2 f_0, \\ s &= \frac{x - x_0}{h} = \frac{x}{0.5} = 2x. \end{aligned}$$

*Sustituimos* $f_0 = 0$*,* $\Delta f_0 = 0.405465$ *y* $\Delta^2 f_0 = -0.117783$*:*

$$P_2(x) = 0 + s(0.405465) + \frac{s(s-1)}{2}(-0.117783).$$

*Evaluando en* $x = 0.75$

$$s = \frac{0.75 - 0}{0.5} = 1.5.$$

*Entonces*

$$P_2(0.75) = 1.5(0.405465) + \frac{1.5(1.5-1)}{2}(-0.117783),$$

*donde* $1.5(0.405465) \approx 0.6081975$*,* $\frac{1.5(0.5)}{2} = \frac{0.75}{2} = 0.375$*, y* $0.375(-0.117783) \approx -0.044168$*, por lo tanto,* $P_2(0.75) \approx 0.6081975 - 0.044168 \approx 0.564029$*, donde el valor real es* $f(0.75) = \ln(1.75) \approx 0.559616$*. El error de interpolación es pequeño:* $|f(0.75) - P_2(0.75)| \approx 0.0044$*.*

**Ejemplo 52.** *Ahora usaremos la forma* regresiva *para aproximar* $f(x) = \sqrt{x}$ *cerca del nodo final* $x_n = 3$*, con* $h = 1$*, y se estimará* $f(2.6)$*. Considérense los puntos* $x_0 = 1, \quad x_1 = 2, \quad x_2 = 3, \qquad h = 1.$

*Entonces*

$$\begin{aligned} f_0 &= \sqrt{1} = 1.000000, \\ f_1 &= \sqrt{2} \approx 1.414214, \\ f_2 &= \sqrt{3} \approx 1.732051. \end{aligned}$$

*la correspondiente tabla de diferencias regresivas (centradas en $x_2$)*

| $k$ | $x_k$ | $f_k$ | *Diferencias* |
|---|---|---|---|
| 0 | 1 | 1.000000 | |
| 1 | 2 | 1.414214 | $\nabla f_1 = f_1 - f_0 = 1.414214 - 1.000000 = 0.414214$ |
| 2 | 3 | 1.732051 | $\nabla f_2 = f_2 - f_1 = 1.732051 - 1.414214 = 0.317837$ |
| | | | $\nabla^2 f_2 = \nabla f_2 - \nabla f_1 = 0.317837 - 0.414214 = -0.096376$ |

*Nos interesan especialmente $\nabla f_2$ y $\nabla^2 f_2$ para el polinomio regresivo. La fórmula de Newton regresivo de grado 2 es*

$$P_2(x) = f_n + s\,\nabla f_n + \frac{s(s+1)}{2!}\,\nabla^2 f_n, \qquad s = \frac{x - x_n}{h}.$$

*donde $n = 2$, $x_n = x_2 = 3$, $h = 1$, por lo que $s = x - 3$. Sustituyendo se tiene*

$$P_2(x) = f_2 + s\,\nabla f_2 + \frac{s(s+1)}{2}\,\nabla^2 f_2.$$

*Evaluando en $x = 2.6$: $s = \frac{2.6-3}{1} = -0.4$. Entonces $P_2(2.6) = 1.732051 + (-0.4)(0.317837) + \frac{(-0.4)(-0.4+1)}{2}(-0.096376)$. Calculamos cada término:*

$$\begin{aligned} (-0.4)(0.317837) &\approx -0.127135, \\ -0.4 + 1 = 0.6, \quad (-0.4)(0.6) = -0.24, \quad \frac{-0.24}{2} &= -0.12, \\ -0.12(-0.096376) &\approx 0.011570. \end{aligned}$$

*Por lo tanto $P_2(2.6) \approx 1.732051 - 0.127135 + 0.011570 \approx 1.616486$.*

*El valor real es $f(2.6) = \sqrt{2.6} \approx 1.612452$. El error es $|f(2.6) - P_2(2.6)| \approx 0.0040$, lo cual muestra una buena aproximación usando la forma regresivacerca del extremo derecho.*

**Ejemplo 53.** *Ahora ilustraremos un caso de grado 3 usando $f(x) = \sin(x)$ con paso $h = 0.5$, cerca de $x_0 = 0$, y evaluaremos en $x = 0.7$. Considérese*

$$x_0 = 0.0, \quad x_1 = 0.5, \quad x_2 = 1.0, \quad x_3 = 1.5, \qquad h = 0.5.$$

*Evaluamos la función en los puntos dados*

$$\begin{aligned} f_0 &= \sin(0.0) = 0.000000, \\ f_1 &= \sin(0.5) \approx 0.479426, \\ f_2 &= \sin(1.0) \approx 0.841471, \\ f_3 &= \sin(1.5) \approx 0.997495 \end{aligned}$$

*con lo que la tabla de diferencias progresivas*

| $k$ | $x_k$ | $f_k$ | *Diferencias* |
|---|---|---|---|
| 0 | 0.0 | 0.000000 | |
| 1 | 0.5 | 0.479426 | $\Delta f_0 = 0.479426 - 0.000000 = 0.479426$ |
| 2 | 1.0 | 0.841471 | $\Delta f_1 = 0.841471 - 0.479426 = 0.362045$ |
| 3 | 1.5 | 0.997495 | $\Delta f_2 = 0.997495 - 0.841471 = 0.156024$ |
| | | | $\Delta^2 f_0 = \Delta f_1 - \Delta f_0 = 0.362045 - 0.479426 = -0.117380$ |
| | | | $\Delta^2 f_1 = \Delta f_2 - \Delta f_1 = 0.156024 - 0.362045 = -0.206021$ |
| | | | $\Delta^3 f_0 = \Delta^2 f_1 - \Delta^2 f_0 = -0.206021 - (-0.117380) = -0.088641$ |

*y por tanto el polinomio de Newton progresivo de grado 3 tiene por ecuación*

$$P_3(x) = f_0 + s\,\Delta f_0 + \frac{s(s-1)}{2!}\,\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\,\Delta^3 f_0, \qquad s = \frac{x - x_0}{h} = \frac{x}{0.5} = 2x.$$

*Sustituimos* $f_0 = 0$, $\Delta f_0 = 0.479426$, $\Delta^2 f_0 = -0.117380$, $\Delta^3 f_0 = -0.088641$*:*

$$P_3(x) = s(0.479426) + \frac{s(s-1)}{2}(-0.117380) + \frac{s(s-1)(s-2)}{6}(-0.088641).$$

*Evaluando en* $x = 0.7$*:* $s = \frac{0.7-0}{0.5} = 1.4$*, entonces.*

$$P_3(0.7) = 1.4(0.479426) + \frac{1.4(1.4-1)}{2}(-0.117380) + \frac{1.4(1.4-1)(1.4-2)}{6}(-0.088641).$$

*Evaluamos cada término:* $1.4(0.479426) \approx 0.671196$, $1.4 - 1 = 0.4$, $\frac{1.4(0.4)}{2} = \frac{0.56}{2} = 0.28$, $0.28(-0.117380) \approx -0.032866$, *luego* $1.4 - 2 = -0.6$, $1.4(0.4)(-0.6) = 1.4 \cdot 0.4 \cdot (-0.6) = -0.336$, $\frac{-0.336}{6} \approx -0.056$, $-0.056(-0.088641) \approx 0.004963$. *Por tanto* $P_3(0.7) \approx 0.671196 - 0.032866 + 0.004963 \approx 0.643293$. *Donde el valor real es* $\sin(0.7) \approx 0.644218$. *El error de interpolación es* $|\sin(0.7) - P_3(0.7)| \approx 0.0009$*, lo cual muestra que un polinomio de grado 3 con cuatro nodos cercanos puede aproximar muy bien a* $\sin(x)$ *en el intervalo considerado.*

**Ejemplo 54.** *Dados los puntos*

$$(0,1), \quad (1,0), \quad (2,-1), \quad (3,2),$$

*construyamos la tabla y el polinomio* $P_3(x)$*.*

$$f[x_0] = 1, \quad f[x_1] = 0, \quad f[x_2] = -1, \quad f[x_3] = 2.$$

$$f[x_0, x_1] = \frac{0-1}{1-0} = -1,$$
$$f[x_1, x_2] = \frac{-1-0}{2-1} = -1,$$
$$f[x_2, x_3] = \frac{2-(-1)}{3-2} = 3.$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{(-1) - (-1)}{2 - 0} = 0,$$
$$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} = \frac{3 - (-1)}{3 - 1} = \frac{4}{2} = 2.$$

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} = \frac{2 - 0}{3 - 0} = \frac{2}{3}.$$

*Entonces el Polinomio de Newton.*

$$\begin{aligned} P_3(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &+ f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 1 + (-1)(x - 0) + 0 \cdot (x)(x - 1) + \frac{2}{3}(x)(x - 1)(x - 2). \end{aligned}$$

*Verificando:*

$$\begin{aligned} &P_3(0) = 1 - 0 + 0 + 0 = 1, \\ &P_3(1) = 1 - 1 + 0 + 0 = 0, \\ &P_3(2) = 1 - 2 + 0 + 0 = -1, \\ &P_3(3) = 1 - 3 + 0 + \tfrac{2}{3}(3)(2)(1) = -2 + 4 = 2. \end{aligned}$$

*La tabla triangular completa quedaría en la forma*

| $x_i$ | $f[x_i]$ | $f[x_i, x_{i+1}]$ | $f[x_i, x_{i+1}, x_{i+2}]$ | $f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$ |
|---|---|---|---|---|
| 0 | 1 | $-1$ | 0 | $\frac{2}{3}$ |
| 1 | 0 | $-1$ | 2 | |
| 2 | $-1$ | 3 | | |
| 3 | 2 | | | |

*Evaluando en un punto interno (por ejemplo, $x = 1.5$).*

$$\begin{aligned} P_3(1.5) &= 1 + (-1)(1.5) + 0 \cdot (1.5)(0.5) + \tfrac{2}{3}(1.5)(0.5)(-0.5) \\ &= 1 - 1.5 + \tfrac{2}{3} \cdot (1.5 \cdot 0.5 \cdot -0.5) \\ &= -0.5 + \tfrac{2}{3} \cdot (-0.375) \\ &= -0.5 - 0.25 = -0.75. \end{aligned}$$

**Ejemplo 55.** *$f(x) = x^2$ en $x_0 = 0$, $h = 1$, estimar $f(0.5)$ Datos en $x = 0, 1, 2, 3$:*

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $y = f(x) = x^2$ | 0 | 1 | 4 | 9 |

*Tabla de diferencias progresivas (en la primera fila):*

$$\Delta y_0 = 1 - 0 = 1 \mid \Delta^2 y_0 = 3 - 1 = 2 \quad \Delta^3 y_0 = 2 - 2 = 0$$

*Aquí* $\Delta y_1 = 4 - 1 = 3$, $\Delta^2 y_1 = 5 - 3 = 2$ *(mostradas sólo para cálculo de* $\Delta^2 y_0$*).*

*Parámetro* $p = \dfrac{x - x_0}{h} = \dfrac{0.5 - 0}{1} = 0.5.$

$$\begin{aligned} P(0.5) &= y_0 + p\,\Delta y_0 + \frac{p(p-1)}{2}\,\Delta^2 y_0 \\ &= 0 + (0.5)(1) + \frac{0.5(-0.5)}{2}\,(2) \\ &= 0.5 + \left(-\frac{0.25}{2}\right)(2) = 0.5 - 0.25 = 0.25. \end{aligned}$$

*Valor exacto:* $f(0.5) = 0.25$*. Coincide (el término de tercer orden es nulo).*

**Ejemplo 56.** $f(x) = x^3$ *en* $x_0 = 1$, $h = 1$, *estimar* $f(1.5)$ *Datos en* $x = 1, 2, 3, 4$*:*

| $x$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $y = f(x) = x^3$ | 1 | 8 | 27 | 64 |

*Diferencias progresivas (primera fila en* $x_0 = 1$*):*

$$\Delta y_0 = 8 - 1 = 7, \quad \Delta y_1 = 27 - 8 = 19, \quad \Delta y_2 = 64 - 27 = 37.$$

$$\Delta^2 y_0 = 19 - 7 = 12, \quad \Delta^2 y_1 = 37 - 19 = 18, \qquad \Delta^3 y_0 = 18 - 12 = 6.$$

*Parámetro* $p = \dfrac{x - x_0}{h} = \dfrac{1.5 - 1}{1} = 0.5.$

$$\begin{aligned} P(1.5) &= y_0 + p\,\Delta y_0 + \frac{p(p-1)}{2}\,\Delta^2 y_0 + \frac{p(p-1)(p-2)}{6}\,\Delta^3 y_0 \\ &= 1 + (0.5)(7) + \frac{0.5(-0.5)}{2}(12) + \frac{0.5(-0.5)(-1.5)}{6}(6). \end{aligned}$$

*Cálculo término a término:*

$$1 + 3.5 + \left(\frac{-0.25}{2}\right)12 + \left(\frac{0.375}{6}\right)6 = 1 + 3.5 - 1.5 + 0.375 = 3.375.$$

*Exacto:* $f(1.5) = (1.5)^3 = 3.375$*.*

**Ejemplo 57.** $f(x) = x^2$ *en* $x_n = 3$, $h = 1$, *estimar* $f(2.6)$ *Datos en* $x = 0, 1, 2, 3$ *(como en el Ejemplo 1). Diferencias regresivas en el extremo* $x_3 = 3$*:*

$$\nabla y_3 = y_3 - y_2 = 9 - 4 = 5, \quad \nabla^2 y_3 = \nabla y_3 - \nabla y_2 = 5 - 3 = 2, \quad \nabla^3 y_3 = 2 - 2 = 0.$$

*Aquí* $\nabla y_2 = y_2 - y_1 = 3$, $\nabla^2 y_2 = \nabla y_2 - \nabla y_1 = 3 - 2 = 1$ *(solo de apoyo).*

*Parámetro* $q = \dfrac{x - x_n}{h} = \dfrac{2.6 - 3}{1} = -0.4.$

$$\begin{aligned} P(2.6) &= y_3 + q\,\nabla y_3 + \frac{q(q+1)}{2}\,\nabla^2 y_3 \\ &= 9 + (-0.4)(5) + \frac{(-0.4)(0.6)}{2}(2) \\ &= 9 - 2 - 0.24 = 6.76. \end{aligned}$$

*Exacto:* $f(2.6) = (2.6)^2 = 6.76$*.*

**Ejemplo 58.** *$f(x) = x^3$ en $x_n = 4$, $h = 1$, estimar $f(3.2)$ Datos en $x = 1, 2, 3, 4$ . Diferencias regresivas en el extremo $x_4 = 4$:*

$$\begin{aligned}\nabla y_4 &= y_4 - y_3 = 64 - 27 = 37,\\ \nabla^2 y_4 &= \nabla y_4 - \nabla y_3 = 37 - 19 = 18,\\ \nabla^3 y_4 &= \nabla^2 y_4 - \nabla^2 y_3 = 18 - 12 = 6.\end{aligned}$$

*Parámetro* $q = \dfrac{x - x_n}{h} = \dfrac{3.2 - 4}{1} = -0.8.$

$$\begin{aligned}P(3.2) &= y_4 + q\,\nabla y_4 + \frac{q(q+1)}{2}\,\nabla^2 y_4 + \frac{q(q+1)(q+2)}{6}\,\nabla^3 y_4\\ &= 64 + (-0.8)(37) + \frac{(-0.8)(0.2)}{2}(18) + \frac{(-0.8)(0.2)(1.2)}{6}(6).\end{aligned}$$

*Cálculo:*

$$64 - 29.6 + (-0.08)(18) + \left(\frac{-0.192}{6}\right)6 = 64 - 29.6 - 1.44 - 0.192 = 32.768.$$

*Exacto:* $f(3.2) = (3.2)^3 = 32.768$.

**Nota 29.**

- *Use Newton hacia adelante si $x$ está próximo a $x_0$; use Newton hacia atrás si $x$ está próximo a $x_n$.*
- *Para funciones polinómicas de grado $m$, las diferencias de orden $> m$ son cero y la fórmula se vuelve exacta con $m + 1$ nodos.*
- *En datos reales con ruido, truncar la serie a pocas diferencias suele mejorar estabilidad.*

**Ejemplo 59.**

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $f(x)$ | 0 | 1 | 4 | 9 |

| $x$ | $y$ | $\Delta y$ | $\Delta^2 y$ | $\Delta^3 y$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 0 |
| 1 | 1 | 3 | 2 | |
| 2 | 4 | 5 | | |
| 3 | 9 | | | |

*Se quiere estimar:*$f(0.5)$, *con* $x_0 = 0$, $h = 1$ *y* $p = 0.5$.
***Sustitución:***

$$\begin{aligned}P(0.5) &= y_0 + p\,\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0\\ &= 0 + (0.5)(1) + \frac{(0.5)(-0.5)}{2}(2)\\ &= 0.5 - 0.25 = 0.25.\end{aligned}$$

*Verificando:* $f(0.5) = (0.5)^2 = 0.25$. $\Rightarrow$ *Coincide exactamente.*

**Ejemplo 60.**

| $x$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $f(x)$ | 1 | 8 | 27 | 64 |

*La tabla de diferencias queda:*

| $x$ | $y$ | $\Delta y$ | $\Delta^2 y$ | $\Delta^3 y$ |
|---|---|---|---|---|
| 1 | 1 | 7 | 12 | 6 |
| 2 | 8 | 19 | 18 | |
| 3 | 27 | 37 | | |
| 4 | 64 | | | |

*Estimemos* $f(1.5)$. $x_0 = 1,\ h = 1,\ p = 0.5$.

$$\begin{aligned} P(1.5) &= y_0 + p\,\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 \\ &= 1 + (0.5)(7) + \frac{(0.5)(-0.5)}{2}(12) + \frac{(0.5)(-0.5)(-1.5)}{6}(6) \\ &= 1 + 3.5 - 1.5 + 0.375 = 3.375 \end{aligned}$$

*verificando* $f(1.5) = (1.5)^3 = 3.375$, *es decir el método reproduce perfectamente el valor porque* $f$ *es un polinomio de grado 3 y se usaron 4 nodos.*

**Ejemplo 61.**

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $f(x)$ | 0 | 1 | 4 | 9 |

*Las Diferencias regresivas (desde el final):*

| $x$ | $y$ | $\nabla y$ | $\nabla^2 y$ | $\nabla^3 y$ |
|---|---|---|---|---|
| 0 | 0 | | | |
| 1 | 1 | 1 | | |
| 2 | 4 | 3 | 2 | |
| 3 | 9 | 5 | 2 | 0 |

*Estimando* $f(2.6)$: $x_n = 3,\ h = 1,\ q = \dfrac{2.6 - 3}{1} = -0.4$.

$$\begin{aligned} P(2.6) &= y_3 + q\,\nabla y_3 + \frac{q(q+1)}{2!}\nabla^2 y_3 \\ &= 9 + (-0.4)(5) + \frac{(-0.4)(0.6)}{2}(2) \\ &= 9 - 2 - 0.24 = 6.76. \end{aligned}$$

*donde* $f(2.6) = (2.6)^2 = 6.76$.

**Ejemplo 62.**

| $x$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $f(x)$ | 1 | 8 | 27 | 64 |

*donde las Diferencias regresivas:*

| $x$ | $y$ | $\nabla y$ | $\nabla^2 y$ | $\nabla^3 y$ |
|---|---|---|---|---|
| 1 | 1 | | | |
| 2 | 8 | 7 | | |
| 3 | 27 | 19 | 12 | |
| 4 | 64 | 37 | 18 | 6 |

*Estimando a* $f(3.2)$*:* $x_n = 4,\ h = 1,\ q = \dfrac{3.2-4}{1} = -0.8.$

$$\begin{aligned} P(3.2) &= y_4 + q\,\nabla y_4 + \frac{q(q+1)}{2!}\nabla^2 y_4 + \frac{q(q+1)(q+2)}{3!}\nabla^3 y_4 \\ &= 64 + (-0.8)(37) + \frac{(-0.8)(0.2)}{2}(18) + \frac{(-0.8)(0.2)(1.2)}{6}(6). \end{aligned}$$

*donde* $64 - 29.6 - 1.44 - 0.192 = 32.768$*, vefrificando* $f(3.2) = (3.2)^3 = 32.768$.

**Ejemplo 63.** *Implementación numérica:*

```
is_equispaced <- function(x, tol = 1e-9) {
  if (length(x) < 2) return(TRUE)
  h <- diff(x)
  max(abs(h - h[1])) < tol
}

falling_prod <- function(p, k) {
  if (k == 0) return(1)
  prod(p - 0:(k-1))
}

rising_prod <- function(q, k) {
  if (k == 0) return(1)
  prod(q + 0:(k-1))
}

# ============================================================
# Tablas de diferencias
# ============================================================
forward_diff_table <- function(x, y, check_equispaced = TRUE) {
  n <- length(x)
  if (length(y) != n) stop("x e y deben tener la misma longitud.")
  if (check_equispaced && !is_equispaced(x)) stop("Los nodos no son equiespaciados.")
```

```
  T <- matrix(0, nrow = n, ncol = n)
  T[,1] <- y
  if (n >= 2) {
    for (j in 2:n) {
      for (i in 1:(n - j + 1)) {
        T[i,j] <- T[i+1,j-1] - T[i,j-1]   # Delta^j y_i
      }
    }
  }
  T
}

backward_diff_table <- function(x, y, check_equispaced = TRUE) {
  n <- length(x)
  if (length(y) != n) stop("x e y deben tener la misma longitud.")
  if (check_equispaced && !is_equispaced(x)) stop("Los nodos no son equiespaciados.
  T <- matrix(0, nrow = n, ncol = n)
  T[,1] <- y
  if (n >= 2) {
    for (j in 2:n) {
      for (i in n:j) {
        T[i,j] <- T[i,j-1] - T[i-1,j-1]   # Nabla^j y_i
      }
    }
  }
  T
}

# ============================================================
# Evaluadores Newton
# - max_order: orden maximo a usar (por defecto, n-1). Truncar ayuda con ruido.
# ============================================================
newton_forward_eval <- function(x, T, xq, max_order = NULL) {
  n <- length(x)
  h <- x[2] - x[1]
  p <- (xq - x[1]) / h
  if (is.null(max_order)) max_order <- n - 1
  max_order <- max(0, min(max_order, n - 1))
  P <- T[1,1]
  for (k in 1:max_order) {
    coef <- falling_prod(p, k) / factorial(k)
    P <- P + coef * T[1, k + 1]
  }
  # Estimacion de error de truncamiento (termino siguiente) si existe:
  err <- NA_real_
```

```
  next_k <- max_order + 1
  if (next_k <= n - 1) {
    coef_next <- falling_prod(p, next_k) / factorial(next_k)
    err <- abs(coef_next * T[1, next_k + 1])
  }
  list(value = P, est_trunc_error = err, p = p)
}

newton_backward_eval <- function(x, T, xq, max_order = NULL) {
  n <- length(x)
  h <- x[2] - x[1]
  q <- (xq - x[n]) / h
  if (is.null(max_order)) max_order <- n - 1
  max_order <- max(0, min(max_order, n - 1))
  P <- T[n,1]
  for (k in 1:max_order) {
    coef <- rising_prod(q, k) / factorial(k)
    P <- P + coef * T[n, k + 1]
  }
  # Estimacion de error (termino siguiente) si existe:
  err <- NA_real_
  next_k <- max_order + 1
  if (next_k <= n - 1) {
    coef_next <- rising_prod(q, next_k) / factorial(next_k)
    err <- abs(coef_next * T[n, next_k + 1])
  }
  list(value = P, est_trunc_error = err, q = q)
}

# ============================================================
# Seleccion automatica de metodo y evaluacion vectorizada
# - method = "auto" | "forward" | "backward"
# - max_order: truncamiento del orden
# Devuelve data.frame con resultados por cada xq
# ============================================================
newton_interp_equispaced <- function(x, y, xq, method = "auto", max_order = NULL) {
  if (!is_equispaced(x)) stop("Los nodos x deben ser equiespaciados.")
  n <- length(x)
  if (length(y) != n) stop("x e y deben tener la misma longitud.")
  # Precomputo de tablas
  Tforw <- forward_diff_table(x, y, check_equispaced = FALSE)
  Tback <- backward_diff_table(x, y, check_equispaced = FALSE)
  # Funcion auxiliar para elegir metodo segun cercania
  choose_method <- function(xq_single) {
    if (method == "forward") return("forward")
```

```
    if (method == "backward") return("backward")
    # auto: decide por cercania
    d0 <- abs(xq_single - x[1])
    dn <- abs(xq_single - x[n])
    if (d0 <= dn) "forward" else "backward"
  }
  # Evaluacion (vectorizada)
  vals <- numeric(length(xq))
  errs <- rep(NA_real_, length(xq))
  meth <- character(length(xq))
  par1 <- numeric(length(xq))  # p o q usado
  for (i in seq_along(xq)) {
    m <- choose_method(xq[i])
    meth[i] <- m
    if (m == "forward") {
      res <- newton_forward_eval(x, Tforw, xq[i], max_order = max_order)
      vals[i] <- res$value
      errs[i] <- res$est_trunc_error
      par1[i] <- res$p
    } else {
      res <- newton_backward_eval(x, Tback, xq[i], max_order = max_order)
      vals[i] <- res$value
      errs[i] <- res$est_trunc_error
      par1[i] <- res$q
    }
  }\newtheorem{Ses}{Sesi\'on}[section]
  data.frame(
    xq = xq,
    estimate = vals,
    est_trunc_error = errs,
    method_used = meth,
    p_or_q = par1
  )
}

# ============================================================
# Ejemplos rapidos
# ============================================================
# Ejemplo 1 (adelante, auto): f(x)=x^2 en x=0:3, evaluar en xq=seq(0,2,by=0.5)
# x <- 0:3; y <- x^2
# newton_interp_equispaced(x, y, seq(0, 2, by = 0.5))

# Ejemplo 2 (atras, auto): f(x)=x^3 en x=1:4, evaluar en xq=c(3.2, 3.6)
# x <- 1:4; y <- x^3
# newton_interp_equispaced(x, y, c(3.2, 3.6))
```

## 5.6. Splines

Los **splines** son funciones polinómicas por tramos utilizadas para interpolar un conjunto de puntos de forma suave y estable. A diferencia de un único polinomio de grado alto (que puede oscilar violentamente entre nodos, efecto de Runge), los splines utilizan polinomios de bajo grado en cada intervalo, garantizando continuidad en los nodos y sus derivadas. Le idea es que en cada subintervalo $[x_i, x_{i+1}]$ se construye un polinomio cúbico $S_i(x)$, y se imponen condiciones para que toda la función $S(x)$ sea continua, con derivadas suaves.

**Definición 12.** *Sea un conjunto de nodos:* $x_0 < x_1 < \cdots < x_n$*, y valores conocidos:* $y_i = f(x_i), \quad i = 0, 1, \ldots, n$.

*El* ***spline cúbico*** $S(x)$ *se define por tramos:*

$$S(x) = \begin{cases} S_0(x), & x_0 \le x < x_1, \\ S_1(x), & x_1 \le x < x_2, \\ \vdots & \\ S_{n-1}(x), & x_{n-1} \le x \le x_n, \end{cases} \tag{5.11}$$

*donde cada tramo es un polinomio cúbico:*

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3. \tag{5.12}$$

El spline cúbico debe cumplir:

1. Interpolación: $S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$.

2. Continuidad de la primera derivada: $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$.

3. Continuidad de la segunda derivada: $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$.

Estas condiciones producen $4(n-1)$ ecuaciones, que junto con dos condiciones adicionales en los extremos definen completamente los $4n$ coeficientes.

- **Spline natural:** Se impone $S''(x_0) = S''(x_n) = 0$. Esto genera una forma "suave" en los extremos.

- **Spline completo (o clamped):** Se imponen las derivadas primeras conocidas: $S'(x_0) = f'(x_0)$, $S'(x_n) = f'(x_n)$.

- **Spline "not-a-knot":** Obliga continuidad de la tercera derivada en los nodos interiores $x_1$ y $x_{n-1}$.

**Sistema tridiagonal de las segundas derivadas**

Definimos:

$$h_i = x_{i+1} - x_i, \qquad m_i = \frac{y_{i+1} - y_i}{h_i}, \quad i = 0, \ldots, n-1. \tag{5.13}$$

El sistema para las segundas derivadas $M_i = S''(x_i)$ es:

$$h_{i-1}M_{i-1} + 2(h_{i-1} + h_i)M_i + h_i M_{i+1} = 6(m_i - m_{i-1}), \quad i = 1, \ldots, n-1, \tag{5.14}$$

con las condiciones de frontera:

$$\text{Spline natural: } M_0 = M_n = 0. \tag{5.15}$$

**Una vez calculadas las** $M_i$**,** cada tramo $S_i(x)$ se obtiene como:

$$S_i(x) = \frac{M_{i+1}(x - x_i)^3}{6h_i} - \frac{M_i(x_{i+1} - x)^3}{6h_i} + \left(\frac{y_{i+1}}{h_i} - \frac{M_{i+1}h_i}{6}\right)(x - x_i) + \left(\frac{y_i}{h_i} - \frac{M_i h_i}{6}\right)(x_{i+1} - x) \tag{5.16}$$

**Algoritmo 13.** *El sistema tridiagonal se resuelve eficientemente con el* ***método de Thomas*** *(algoritmo especializado para matrices tridiagonales). El proceso computacional sigue estos pasos:*

1. *Calcular los* $h_i$ *y* $m_i$*.*
2. *Formar el sistema tridiagonal en términos de* $M_i$*.*
3. *Resolver para obtener* $M_1, \ldots, M_{n-1}$*.*
4. *Construir los polinomios* $S_i(x)$ *en cada subintervalo.*

**Ejemplo 64.** *Interpolar los puntos:*

$$(0, 0), \quad (1, 1), \quad (2, 0).$$

*Se tiene* $n = 2$*,* $h_0 = h_1 = 1$*,* $m_0 = 1$*,* $m_1 = -1$*.*
*Ecuación central:*

$$h_0 M_0 + 2(h_0 + h_1)M_1 + h_1 M_2 = 6(m_1 - m_0),$$

$$1(0) + 4M_1 + 1(0) = 6(-1 - 1) = -12 \quad \Rightarrow \quad M_1 = -3.$$

*Y* $M_0 = M_2 = 0$*.*
*Sustituyendo en la fórmula del tramo* $[0, 1]$*:*

$$S_0(x) = -\frac{M_1}{6}(x - 1)^3 + x.$$

*En* $[1, 2]$*:*

$$S_1(x) = -\frac{M_1}{6}(2 - x)^3 + (2 - x).$$

*Graficar ambos tramos muestra una curva suave en forma de "loma".*

**Ejemplo 65.** *Puntos:*

$$(0,0), \quad (1,1), \quad (2,0), \quad (3,1).$$

*Datos:* $h_i = 1$, $m_0 = 1$, $m_1 = -1$, $m_2 = 1$.

*Sistema:*

$$\begin{cases} 4M_1 + M_2 = 6(-1-1) = -12, \\ M_1 + 4M_2 = 6(1-(-1)) = 12. \end{cases}$$

*Resolviendo:*

$$M_1 = -4, \quad M_2 = 4, \quad M_0 = M_3 = 0.$$

*Cada tramo se obtiene sustituyendo los valores de $M_i$ en la fórmula general.*

**Ejemplo 66.** *Puntos:*

$$(0,1), \quad (1,2), \quad (2,0),$$

*con derivadas conocidas $S'(0) = 0$ y $S'(2) = -1$.*

*Se aplican las condiciones en los extremos:*

$$2h_0M_0 + h_0M_1 = 6\left(\frac{y_1 - y_0}{h_0} - S'(0)\right),$$

$$h_0M_0 + 2h_0M_1 = 6\left(S'(2) - \frac{y_2 - y_1}{h_0}\right).$$

*Resolviendo se obtienen $M_0$ y $M_1$, y luego los coeficientes $a_i, b_i, c_i, d_i$ de cada tramo.*

**Ejemplo 67.** *Mediciones de temperatura ($^\circ C$) a distintas horas:*

| *Hora (h)* | 0 | 3 | 6 | 9 |
|---|---|---|---|---|
| *Temperatura* | 15 | 18 | 14 | 10 |

*Interpolar con un spline cúbico natural y estimar $T(4.5)$.*

**Algoritmo 14.**

1. *Calcular $h_i = 3$, $m_i = (y_{i+1} - y_i)/h_i$.*
2. *Formar el sistema para $M_1, M_2$.*
3. *Resolverlo con el método de Thomas.*
4. *Evaluar $S_1(4.5)$ usando la fórmula de $S_i(x)$ en el intervalo $[3,6]$.*

***Resultado esperado:*** *$T(4.5) \approx 15.5^\circ C$. La curva obtenida describe de forma suave la variación de temperatura entre mediciones.*

**Nota 30.**

- *Los splines cúbicos son continuos en $S$, $S'$, y $S''$, lo que garantiza suavidad.*
- *Reducen el error sin introducir oscilaciones (efecto Runge).*
- *El error máximo en un spline cúbico natural es proporcional a $O(h^4)$.*
- *Son ampliamente usados en gráficos por computadora, CAD, animación y análisis de datos experimentales.*

## 5.7. Error de Interpolación

La interpolación polinómica busca aproximar una función $f(x)$ mediante un polinomio $P_n(x)$ que coincide con $f(x)$ en $n+1$ puntos conocidos. Sin embargo, entre los nodos de interpolación puede existir una diferencia o **error de interpolación**, denotado por $R_n(x)$.

### Fórmula general del error

Sea $f \in C^{(n+1)}[a,b]$ y $P_n(x)$ el polinomio interpolante de grado $n$ para los puntos $(x_0, y_0), \ldots, (x_n, y_n)$ con $y_i = f(x_i)$. Entonces, para todo $x \in [a,b]$, existe un número $\xi \in (a,b)$ tal que:

$$R_n(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$$

**Interpretación:** El error depende de dos factores:

- El término $\dfrac{f^{(n+1)}(\xi)}{(n+1)!}$, relacionado con la suavidad de $f(x)$.
- El producto $\displaystyle\prod_{i=0}^{n}(x - x_i)$, que refleja la posición de los nodos de interpolación.

### Cota del error

Si se conoce una cota $|f^{(n+1)}(x)| \leq M$ para $x \in [a,b]$, entonces:

$$|R_n(x)| \leq \frac{M}{(n+1)!} \max_{x \in [a,b]} \left| \prod_{i=0}^{n} (x - x_i) \right|.$$

Esta expresión indica que el error aumenta:

- Si $f^{(n+1)}(x)$ es grande (función muy curvada o con derivadas altas).
- Si los nodos $x_i$ están mal distribuidos (especialmente equiespaciados para $n$ grande).

### Comportamiento del error y efecto de Runge

Cuando los nodos son equiespaciados, el término

$$\prod_{i=0}^{n} (x - x_i)$$

crece rápidamente en los extremos del intervalo, provocando oscilaciones en el polinomio interpolante — fenómeno conocido como **efecto de Runge**. Este efecto se agrava con $n$ grande y funciones con alta curvatura, como $f(x) = \dfrac{1}{1+x^2}$.

*Conclusión: aumentar el grado $n$ no garantiza una mejor aproximación.*

**Nodos de Chebyshev (minimizan el error máximo)**

Para reducir las oscilaciones, se eligen nodos no equiespaciados, conocidos como **nodos de Chebyshev**. En el intervalo $[-1, 1]$ se definen como:

$$x_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0, 1, \ldots, n.$$

Estos nodos distribuyen más puntos cerca de los extremos del intervalo, donde el error tiende a crecer más.

**Propiedad fundamental:** Los nodos de Chebyshev minimizan la cantidad

$$\max_{x\in[-1,1]} \left|\prod_{i=0}^{n}(x - x_i)\right|,$$

y, por tanto, minimizan el error máximo de interpolación en el sentido de la norma infinita.

**Ejemplo 68.** *: comparación cualitativa*
*Sea* $f(x) = \dfrac{1}{1+x^2}$ *en* $[-5, 5]$.

- *Con* $n = 10$ *y nodos* ***equiespaciados****: el polinomio presenta grandes oscilaciones en los extremos (efecto de Runge).*
- *Con* $n = 10$ *y nodos* ***de Chebyshev****: la interpolación es estable y el error máximo es mucho menor.*

**Ejemplo 69.** *Estimación del error Sea* $f(x) = e^x$ *en* $[0, 1]$*, con nodos equiespaciados* $x_0 = 0$*,* $x_1 = 0.5$*,* $x_2 = 1$*. Entonces* $f^{(3)}(x) = e^x \leq e$.
*Usando la fórmula del error para* $x = 0.25$*:*

$$|R_2(0.25)| \leq \frac{e}{3!}|(0.25 - 0)(0.25 - 0.5)(0.25 - 1)| = \frac{e}{6}(0.25 \cdot 0.25 \cdot 0.75) \approx 0.0085.$$

*La cota muestra que el error es pequeño, y decrece proporcionalmente a* $h^{n+1}$.

**Nota 31.**

- *La magnitud de* $R_n(x)$ *depende tanto de la función como de la elección de nodos.*
- *Nodos de Chebyshev distribuyen mejor el error, especialmente para grados altos.*
- *En la práctica, cuando los datos son experimentales o* $f(x)$ *es desconocida, se prefiere reducir* $n$ *y usar* ***splines****, que mantienen continuidad y baja oscilación.*

## 5.8. Ejercicios

Esta sección presenta ejercicios orientados a consolidar la comprensión de los métodos de interpolación estudiados: Lagrange, Newton (diferencias divididas) y Newton hacia adelante/atrás (diferencias finitas).

Se recomienda resolver los primeros ejercicios de forma manual y posteriormente verificar los resultados con el código en R proporcionado.

**Ejercicio 13.** *1. Dado el conjunto de puntos*

$$(0, 1),\ (1, 3),\ (2, 11),$$

a) *Determine el polinomio interpolante mediante el* ***método de Lagrange****.*

b) *Verifique el resultado construyendo la* ***tabla de diferencias divididas*** *y aplicando el* ***método de Newton****.*

c) *Evalúe el polinomio en $x = 1.5$ y compare con la función cuadrática ajustada visualmente a los puntos.*

*2. Con los puntos*

$$(1, 2),\ (2, 5),\ (3, 10),\ (4, 17),$$

a) *Obtenga el polinomio interpolante de grado 3 usando el* ***método de Newton****.*

b) *Muestre la tabla completa de diferencias divididas.*

c) *Evalúe el polinomio en $x = 2.5$.*

d) *Verifique el resultado calculando directamente $f(x) = x^2 + 1$ y compare.*

*3. Dados los puntos*

$$(-1, 4),\ (0, 1),\ (2, 3),$$

a) *Construya el polinomio de Lagrange.*

b) *Expanda y simplifique el resultado hasta la forma general $P_2(x) = a_0 + a_1x + a_2x^2$.*

c) *Verifique los valores de $P_2(-1)$, $P_2(0)$ y $P_2(2)$.*

**Ejercicio 14.** *1. Para la función tabulada*

| $x$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $f(x)$ | 1 | 2 | 4 | 8 | 16 |

a) *Construya la* ***tabla de diferencias progresivas*** $\Delta y_i$.

b) *Determine el polinomio interpolante con Newton hacia adelante.*

c) *Calcule $f(2.5)$ y compárelo con $2^{2.5}$.*

2. *Para los valores:*

| $x$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $f(x)$ | 1 | 8 | 27 | 64 |

   a) *Construya la tabla de* ***diferencias regresivas*** $\nabla y_i$.

   b) *Aplique la fórmula de Newton hacia atrás para estimar* $f(3.4)$.

   c) *Compare con el valor exacto de* $f(x) = x^3$.

3. *Para los datos experimentales:*

| $x$ | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| $y$ | 20.5 | 24.0 | 32.0 | 42.5 | 51.0 |

   a) *Verifique que los nodos sean equiespaciados.*

   b) *Calcule las diferencias progresivas hasta el tercer orden.*

   c) *Estime* $y(25)$ *usando Newton hacia adelante.*

   d) *Comente si el uso de un polinomio de orden más alto podría mejorar o empeorar la estabilidad numérica.*

**Ejercicio 15.** *Implemente los siguientes ejercicios utilizando las funciones* `newton_interp_equispaced()`, `forward_diff_table()` *y* `backward_diff_table()`.

1. *Replique los ejemplos manuales anteriores en R y compare los resultados obtenidos manualmente con los valores calculados por el script. Incluya en su reporte:*

   - *La tabla de diferencias generada.*
   - *El valor estimado y el error de truncamiento.*
   - *Una gráfica de los puntos originales y el polinomio interpolante.*

2. *Genere aleatoriamente 5 puntos de una función cuadrática y aplique:*

   - *Interpolación de Lagrange (con la fórmula clásica).*
   - *Interpolación de Newton (diferencias divididas).*
   - *Interpolación hacia adelante (diferencias finitas).*

   *Compare los tres resultados gráficamente y analice diferencias numéricas.*

3. *Usando los datos:*

$$x = [0, 1, 2, 3, 4], \quad y = [2.0, 2.7, 4.8, 8.9, 16.0],$$

   a) *Aplique el método automático* `newton_interp_equispaced()` *para evaluar la función en* $x = 1.5, 2.5, 3.5$.

b) *Reporte el método seleccionado (adelante o atrás) y el error estimado.*

c) *Compare con una función de referencia* $f(x) = 2^x$.

**Ejercicio 16.** *Puntos:* $(0,1),(1,3),(2,11)$.

- *Lagrange / Forma general: Sea* $P_2(x) = a_0 + a_1x + a_2x^2$. *Con* $x = 0 \Rightarrow a_0 = 1$; $x = 1 \Rightarrow 1 + a_1 + a_2 = 3$; $x = 2 \Rightarrow 1 + 2a_1 + 4a_2 = 11$. *De* $a_1 + a_2 = 2$ *y* $a_1 + 2a_2 = 5 \Rightarrow a_2 = 3,\ a_1 = -1$.

$$P_2(x) = 1 - x + 3x^2.$$

- *Evaluación:* $P_2(1.5) = 1 - 1.5 + 3(2.25) = 6.25$.
- *Newton: tabla de diferencias divididas produce los mismos coeficientes.*

**Ejercicio 17.** *Puntos:* $(1,2),(2,5),(3,10),(4,17)$.

- *Observa que* $y = x^2 + 1$ *calza exactamente con los datos.*
- ***Newton**: diferencias*

$$\Delta y = \{3,5,7\}, \quad \Delta^2 y = \{2,2\}, \quad \Delta^3 y = \{0\}.$$

- ***Polinomio**: de grado 2 (término cúbico nulo), equivale a* $P(x) = x^2 + 1$.
- ***Evaluación**:* $P(2.5) = 2.5^2 + 1 = 7.25$.

**Ejercicio 18.** *Puntos:* $(-1,4),(0,1),(2,3)$.

- *Forma general* $P_2(x) = a_0 + a_1x + a_2x^2$. *De* $x = 0 \Rightarrow a_0 = 1$.
- *Sistema:* $-a_1 + a_2 = 3$ *y* $a_1 + 2a_2 = 1$. *Solución:* $a_2 = \frac{4}{3},\ a_1 = -\frac{5}{3}$.

$$P_2(x) = 1 - \frac{5}{3}x + \frac{4}{3}x^2.$$

- *Verifica:* $P_2(-1) = 4,\ P_2(0) = 1,\ P_2(2) = 3$.

**Ejercicio 19.** *Tabla* $2^x$ *en* $x = 0,1,2,3,4$*:* $y = \{1,2,4,8,16\}$.

- *Diferencias progresivas:*$\Delta y_0 = 1,\ \Delta^2 y_0 = 1,\ \Delta^3 y_0 = 1,\ \Delta^4 y_0 = 1$.
- *Newton adelante (grado 4),* $x_0 = 0, h = 1$, $p = 2.5$*:*

$$\begin{aligned} P(2.5) &= y_0 + p\Delta y_0 + \frac{p(p-1)}{2}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{6}\Delta^3 y_0 + \frac{p(p-1)(p-2)(p-3)}{24}\Delta^4 y_0 \\ &= 1 + 2.5 + 1.875 + 0.3125 - 0.0390625 \\ &= 5.6484375. \end{aligned}$$

- *Comparación:* $2^{2.5} = \sqrt{32} \approx 5.65685425$. *Error* $\approx -8.4168 \times 10^{-3}$ *(el grado 4 no puede reproducir una función exponencial).*

**Ejercicio 20.** $x = 1, 2, 3, 4$ *con* $f(x) = x^3 = \{1, 8, 27, 64\}$*; estimar* $f(3.4)$ *por* ***atrás****.*

- ***Regresivas en*** $x_4 = 4$***:*** $\nabla y_4 = 37,\ \nabla^2 y_4 = 18,\ \nabla^3 y_4 = 6.$
- $h = 1,\ q = (3.4 - 4) = -0.6.$

$$\begin{aligned} P(3.4) &= y_4 + q\nabla y_4 + \frac{q(q+1)}{2}\nabla^2 y_4 + \frac{q(q+1)(q+2)}{6}\nabla^3 y_4 \\ &= 64 - 22.2 - 2.16 - 0.336 \\ &= 39.304. \end{aligned}$$

- ***Exacto****:* $3.4^3 = 39.304$*. (Para polinomio cúbico con 4 nodos, el método reproduce exactamente.)*

**Ejercicio 21.** *Datos experimentales equiespaciados (*$h = 10$*):*

| $x$ | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|
| $y$ | 20.5 | 24.0 | 32.0 | 42.5 | 51.0 |

- ***Progresivas en*** $x_0 = 10$***:***

$$\Delta y_0 = 3.5, \quad \Delta^2 y_0 = 4.5, \quad \Delta^3 y_0 = -2.0, \quad \Delta^4 y_0 = -2.5.$$

- ***Estimar*** $y(25)$ *(adelante, hasta orden 3):* $p = \frac{25-10}{10} = 1.5.$

$$\begin{aligned} P(25) &= y_0 + p\Delta y_0 + \frac{p(p-1)}{2}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{6}\Delta^3 y_0 \\ &= 20.5 + (1.5)(3.5) + \frac{1.5 \cdot 0.5}{2}(4.5) + \frac{1.5 \cdot 0.5 \cdot (-0.5)}{6}(-2.0) \\ &= 27.5625. \end{aligned}$$

- ***Término siguiente (cota pragmática)****:*

$$\frac{p(p-1)(p-2)(p-3)}{24}\Delta^4 y_0 = \frac{1.5 \cdot 0.5 \cdot (-0.5) \cdot (-1.5)}{24}(-2.5) \approx \boxed{-0.0586}.$$

*(Magnitud* $\approx 5.86 \times 10^{-2}$*: sugiere el orden del error de truncamiento.)*

**Ejercicio 22.** *Para 5 puntos de una cuadrática simulada* $y = ax^2 + bx + c$ *con ruido* $N(0, \sigma)$*:*

- *Lagrange (clásico) y Newton (divididas) deben coincidir en precisión con datos sin ruido.*

- *Con ruido, truncar el orden (p.ej. usar 3–4 nodos o limitar* `max_order`*) reduce oscilaciones y mejora estabilidad.*

**Ejercicio 23.** *Con* $x = \{0, 1, 2, 3, 4\}$*,* $y = \{2.0, 2.7, 4.8, 8.9, 16.0\}$*:*

- `newton_interp_equispaced` *seleccionará* adelante *cerca de* $x_0$ *y* atrás *cerca de* $x_4$ *(o el que quede más cercano).*
- *La columna* `est_trunc_error` *reportará el tamaño del término siguiente (guía de error).*
- *Al comparar con* $2^x$*, se observarán discrepancias crecientes fuera del rango central (interpolación no reproduce exponenciales con grado bajo de forma exacta).*

**Ejemplo 70.** *Datos:*

$$(x_i, y_i) = (0, 1),\ (1, 2.2),\ (2, 2.8),\ (3, 3.6).$$

*Pendientes por tramo*

$$b_0 = \frac{2.2 - 1}{1 - 0} = 1.2, \quad b_1 = \frac{2.8 - 2.2}{2 - 1} = 0.6, \quad b_2 = \frac{3.6 - 2.8}{3 - 2} = 0.8.$$

*Por tramos:*

$$S(x) = \begin{cases} 1 + 1.2(x - 0), & 0 \le x \le 1, \\ 2.2 + 0.6(x - 1), & 1 \le x \le 2, \\ 2.8 + 0.8(x - 2), & 2 \le x \le 3. \end{cases}$$

*Evaluaciones:*

$$S(1.7) = 2.2 + 0.6(0.7) = 2.62, \qquad S(2.4) = 2.8 + 0.8(0.4) = 3.12.$$

*Verificaciones:*

$$S(1^-) = 2.2,\ S(1^+) = 2.2; \quad S(2^-) = 2.8,\ S(2^+) = 2.8.$$

*(Sólo continuidad de la función; derivadas cambian en los nudos.)*

**Ejemplo 71.** *Datos:*

$$(x_0, y_0) = (0, 0),\ (x_1, y_1) = (1, 1),\ (x_2, y_2) = (2, 0),\ (x_3, y_3) = (3, 1).$$

*Paso* $h_i = 1$*. Pendientes locales:*

$$m_0 = \frac{1 - 0}{1} = 1, \quad m_1 = \frac{0 - 1}{1} = -1, \quad m_2 = \frac{1 - 0}{1} = 1.$$

*Spline* natural*:* $M_0 = M_3 = 0$*. Sistema para* $M_1, M_2$ *(tridiagonal clásico):*

$$\begin{cases} M_0 + 4M_1 + M_2 = 6(m_1 - m_0) = 6(-1 - 1) = -12, \\ M_1 + 4M_2 + M_3 = 6(m_2 - m_1) = 6(1 - (-1)) = 12. \end{cases}$$

*Con $M_0 = M_3 = 0$:*

$$4M_1 + M_2 = -12, \qquad M_1 + 4M_2 = 12.$$

*Resolviendo:*

$$M_2 = 4, \quad M_1 = -4, \quad (M_0 = 0,\ M_3 = 0).$$

*Coeficientes por tramo ($h_i = 1$):*

$$\begin{aligned} a_i &= y_i, \\ b_i &= m_i - \frac{(2M_i + M_{i+1})h_i}{6}, \\ c_i &= \frac{M_i}{2}, \\ d_i &= \frac{M_{i+1} - M_i}{6h_i}. \end{aligned}$$

*Tramo $[0,1]$ ($i = 0$): $m_0 = 1,\ M_0 = 0,\ M_1 = -4$:*

$$a_0 = 0,\ b_0 = 1 - \tfrac{(0-4)}{6} = \tfrac{5}{3},\ c_0 = 0,\ d_0 = \tfrac{-4-0}{6} = -\tfrac{2}{3}.$$

$$S_0(x) = \tfrac{5}{3}x - \tfrac{2}{3}x^3.$$

*Tramo $[1,2]$ ($i = 1$): $m_1 = -1,\ M_1 = -4,\ M_2 = 4$:*

$$a_1 = 1,\ b_1 = -1 - \tfrac{(2(-4)+4)}{6} = -\tfrac{4}{3},\ c_1 = -2,\ d_1 = \tfrac{4-(-4)}{6} = \tfrac{4}{3},$$

$$S_1(x) = 1 - \tfrac{4}{3}(x-1) - 2(x-1)^2 + \tfrac{4}{3}(x-1)^3.$$

*Tramo $[2,3]$ ($i = 2$): $m_2 = 1,\ M_2 = 4,\ M_3 = 0$:*

$$a_2 = 0,\ b_2 = 1 - \tfrac{(8+0)}{6} = -\tfrac{1}{3},\ c_2 = 2,\ d_2 = \tfrac{0-4}{6} = -\tfrac{2}{3},$$

$$S_2(x) = -\tfrac{1}{3}(x-2) + 2(x-2)^2 - \tfrac{2}{3}(x-2)^3 + 0 \ (y \text{ desplazar por } y_2 = 0).$$

*Evaluaciones:*

$$S(0) = 0,\ S(1) = 1,\ S(2) = 0,\ S(3) = 1, \quad S'(0) = \tfrac{5}{3},\ S''(0) = 0,\ S''(3) = 0.$$

*(Estructura suave: continuidad de $S, S', S''$ en $x = 1, 2$.)*

**Ejemplo 72.** *Datos:*

$$(0,0),\ (1,2),\ (2,3), \qquad S'(0) = 1,\ S'(2) = 0.$$

$h_0 = h_1 = 1$, $m_0 = 2$, $m_1 = 1$. *Frontera clamped y ecuaciones interiores:*

$$\begin{cases} 2h_0 M_0 + h_0 M_1 = 6(m_0 - S'(0)) = 6, \\ h_0 M_0 + 2(h_0 + h_1) M_1 + h_1 M_2 = 6(m_1 - m_0) = -6, \\ h_1 M_1 + 2h_1 M_2 = 6(S'(2) - m_1) = -6. \end{cases}$$

*Resolución:*

$$M_0 = 4, \quad M_1 = -2, \quad M_2 = -2.$$

*Coeficientes (con $h = 1$):*

$$\begin{aligned} a_0 = 0, \quad b_0 = m_0 - \tfrac{2M_0 + M_1}{6} = 2 - \tfrac{8-2}{6} = 1, \\ c_0 = \tfrac{M_0}{2} = 2, \quad d_0 = \tfrac{M_1 - M_0}{6} = -1, \\ begineqnarray*4pt] a_1 = 2, \quad b_1 = m_1 - \tfrac{2M_1 + M_2}{6} = 1 - \tfrac{-4-2}{6} = 2, \\ c_1 = \tfrac{M_1}{2} = -1, \quad d_1 = \tfrac{M_2 - M_1}{6} = 0. \end{aligned}$$

*Por tramos:*

$$S(x) = \begin{cases} x + 2x^2 - x^3, & 0 \le x \le 1, \\ 2 + 2(x-1) - (x-1)^2, & 1 \le x \le 2. \end{cases}$$

*Verif.:* $S(0) = 0$, $S(1) = 2$, $S(2) = 3$; $S'(0) = 1$ *y* $S'(2) = 0$ *(clamped cumplido).*

**Ejemplo 73.** *1.* ***Spline lineal:*** *Con* $(0, 1.2), (1, 2.0), (3, 3.1), (4, 4.0)$*:*

a*) Escriba* $S_i(x)$ *por tramos y evalúe* $S(2.5)$.

b*) Grafique a mano los segmentos y comente la falta de suavidad en los nudos.*

*2.* ***Spline cúbico natural:*** *Con* $(0, 0), (1, 1), (2, 0), (3, 1)$*:*

a*) Arme el sistema tridiagonal para* $M_1, M_2$ *y resuélvalo.*

b*) Obtenga los coeficientes* $(a_i, b_i, c_i, d_i)$ *y evalúe* $S(1.5)$ *y* $S(2.3)$.

*3.* ***Spline cúbico completo:*** *Con* $(0, 0), (1, 2), (2, 3)$ *y* $S'(0) = 1$, $S'(2) = 0$*:*

a*) Arme el sistema para* $M_0, M_1, M_2$.

b*) Calcule* $S(0.5)$ *y* $S(1.8)$.

## 5.9. Implementaciones Computacionales

### Pseudocódigo (forma clásica)

Sea `x[0..n]`, `y[0..n]` los datos, y `xq` el punto a evaluar.

```
ALGORITMO LagrangeEval(x[0..n], y[0..n], xq):
    p := 0
    para i := 0..n hacer
        Li := 1
        para j := 0..n hacer
            si j != i entonces
                Li := Li * (xq - x[j]) / (x[i] - x[j])
            fin si
        fin para
        p := p + y[i] * Li
    fin para
    retornar p
FIN
```

### Pseudocódigo (baricéntrico, estable)

Precompute los pesos baricéntricos $w_i = \dfrac{1}{\prod_{j \neq i}(x_i - x_j)}$ una sola vez.

```
ALGORITMO BarycentricPrecompute(x[0..n]):
    para i := 0..n hacer
        wi := 1
        para j := 0..n hacer
            si j != i entonces
                wi := wi * 1.0 / (x[i] - x[j])
            fin si
        fin para
        w[i] := wi
    fin para
    retornar w
FIN

ALGORITMO BarycentricEval(x[0..n], y[0..n], w[0..n], xq):
    // Si xq coincide con un nodo, devuelve su y exacta
    para i := 0..n hacer
        si xq == x[i] entonces retornar y[i]
    fin para
    num := 0 ; den := 0
    para i := 0..n hacer
        term := w[i] / (xq - x[i])
        num := num + term * y[i]
        den := den + term
    fin para
    retornar num / den
FIN
```

### Implementación en R (forma clásica y baricéntrica)

```
# --- Forma clasica O(n^2) por evaluacion ---
lagrange_eval <- function(x, y, xq) {
  # x, y: vectores de longitud n+1
  # xq: escalar o vector
  sapply(xq, function(xx) {
    n <- length(x) - 1
    p <- 0
    for (i in 0:n) {
      Li <- 1
      for (j in 0:n) {
        if (j != i) {
          Li <- Li * (xx - x[j+1]) / (x[i+1] - x[j+1])
        }
      }
      p <- p + y[i+1] * Li
    }
    p
  })
}

# --- Pesos baricentricos (precomputo O(n^2)) ---
barycentric_weights <- function(x) {
  n <- length(x) - 1
  w <- rep(1, n+1)
  for (i in 0:n) {
    for (j in 0:n) {
      if (j != i) w[i+1] <- w[i+1] / (x[i+1] - x[j+1])
    }
  }
  w
}

# --- Evaluacion baricentrica O(n) por punto ---
barycentric_eval <- function(x, y, w, xq) {
  sapply(xq, function(xx) {
    # Coincidencia exacta con un nodo
    idx <- which(xx == x)
    if (length(idx) > 0) return(y[idx[1]])
    terms <- w / (xx - x)
    sum(terms * y) / sum(terms)
  })
}

# --- Ejemplo de uso ---
```

```
# Datos: f(x)=x^2 en nodos 0,1,2
x <- c(0,1,2); y <- x^2

# Evaluar en una malla
xx <- seq(0,2,length.out=21)
yy_lagr <- lagrange_eval(x,y,xx)

w  <- barycentric_weights(x)
yy_bar <- barycentric_eval(x,y,w,xx)

# yy_lagr y yy_bar deben coincidir (salvo error redondeo) con xx^2
# plot(xx, yy_lagr, type="l"); points(x,y)
```

### Pseudocódigo

```
ALGORITMO NewtonDiffDiv(x[0..n], y[0..n]):
    para i := 0..n hacer
        F[i,0] := y[i]
    fin para
    para j := 1..n hacer
        para i := 0..n-j hacer
            F[i,j] := (F[i+1,j-1] - F[i,j-1]) / (x[i+j] - x[i])
        fin para
    fin para
    retornar F[0,0..n] // Coeficientes f[x0], f[x0,x1], ...
FIN

ALGORITMO NewtonEval(x[0..n], F[0,0..n], xq):
    p := F[0,0]
    prod := 1
    para j := 1..n hacer
        prod := prod * (xq - x[j-1])
        p := p + F[0,j]*prod
    fin para
    retornar p
FIN
```

### Implementación en R

```
# --- Tabla de diferencias divididas ---
newton_diffdiv <- function(x, y) {
  n <- length(x)
  F <- matrix(0, n, n)
  F[,1] <- y
  for (j in 2:n) {
    for (i in 1:(n-j+1)) {
```

```
      F[i,j] <- (F[i+1,j-1] - F[i,j-1]) / (x[i+j-1] - x[i])
    }
  }
  F
}

# --- Evaluacion del polinomio de Newton ---
newton_eval <- function(x, F, xq) {
  n <- length(x)
  sapply(xq, function(xx) {
    p <- F[1,1]
    prod <- 1
    for (j in 2:n) {
      prod <- prod * (xx - x[j-1])
      p <- p + F[1,j]*prod
    }
    p
  })
}

# --- Ejemplo ---
x <- c(0,1,2,3)
y <- c(1,0,-1,2)
F <- newton_diffdiv(x,y)
xx <- seq(0,3,length.out=41)
yy <- newton_eval(x,F,xx)

# plot(xx,yy,type="l",col="blue"); points(x,y,col="red",pch=19)
```

```
ALGORITMO TablaDiferenciasProgresivas(x[0..n], y[0..n]):
    // Requiere nodos equiespaciados: x[i] = x[0] + i*h
    VerificarEquiespaciado(x)
    T := matriz (n+1) x (n+1) llena de 0
    para i := 0..n hacer
        T[i,0] := y[i]
    fin para
    para j := 1..n hacer
        para i := 0..n-j hacer
            T[i,j] := T[i+1,j-1] - T[i,j-1]   //
        fin para
    fin para
    retornar T
FIN
```

```
ALGORITMO NewtonAdelanteEval(x[0..n], T, xq):
    // T es la tabla de diferencias progresivas
    h := x[1] - x[0]
    p := (xq - x[0]) / h
    P := T[0,0]
    prod := 1
    para k := 1..n hacer
        prod := prod * (p - (k-1))          // p(p-1)(p-2)...
        P := P + (prod / k!) * T[0,k]       // usa factorial de k
    fin para
    retornar P
FIN



ALGORITMO TablaDiferenciasRegresivas(x[0..n], y[0..n]):
    VerificarEquiespaciado(x)
    T := matriz (n+1) x (n+1) llena de 0
    para i := 0..n hacer
        T[i,0] := y[i]
    fin para
    para j := 1..n hacer
        para i := n..j hacer
            T[i,j] := T[i,j-1] - T[i-1,j-1]  //
        fin para
    fin para
    retornar T
FIN


ALGORITMO NewtonAtrasEval(x[0..n], T, xq):
    h := x[1] - x[0]
    q := (xq - x[n]) / h
    P := T[n,0]
    prod := 1
    para k := 1..n hacer
        prod := prod * (q + (k-1))          // q(q+1)(q+2)...
        P := P + (prod / k!) * T[n,k]
    fin para
    retornar P
FIN


# ============================================================
```

```
# Utilidades
# ============================================================

is_equispaced <- function(x, tol = 1e-9) {
  if (length(x) < 2) return(TRUE)
  h <- diff(x)
  max(abs(h - h[1])) < tol
}

falling_prod <- function(p, k) {
  # p(p-1)(p-2)...(p-k+1), falling factorial
  if (k == 0) return(1)
  prod(p - 0:(k-1))
}

rising_prod <- function(q, k) {
  # q(q+1)(q+2)...(q+k-1), rising factorial
  if (k == 0) return(1)
  prod(q + 0:(k-1))
}

# ============================================================
# 1) Tabla de diferencias progresivas (Newton hacia adelante)
# ============================================================
forward_diff_table <- function(x, y, check_equispaced = TRUE) {
  n <- length(x)
  if (length(y) != n) stop("x e y deben tener la misma longitud.")
  if (check_equispaced && !is_equispaced(x)) {
    stop("Los nodos no son equiespaciados.")
  }
  T <- matrix(0, nrow = n, ncol = n)
  T[, 1] <- y
  if (n >= 2) {
    for (j in 2:n) {
      for (i in 1:(n - j + 1)) {
        T[i, j] <- T[i + 1, j - 1] - T[i, j - 1]  # Delta^j y_i
      }
    }
  }
  T
}

# ============================================================
# 2) Evaluacion Newton hacia adelante
#    P(x) = y0 + p y0 + p(p-1)/2! ^2 y0 + ...
```

```
# ============================================================
newton_forward_eval <- function(x, T, xq) {
  n <- length(x)
  h <- x[2] - x[1]
  p <- (xq - x[1]) / h
  # T[1, k] contiene ^(k-1) y_0, con k empezando en 1
  P <- T[1, 1]
  for (k in 2:n) {
    coef <- falling_prod(p, k - 1) / factorial(k - 1)
    P <- P + coef * T[1, k]
  }
  P
}

# ============================================================
# 3) Tabla de diferencias regresivas (Newton hacia atras)
# ============================================================
backward_diff_table <- function(x, y, check_equispaced = TRUE) {
  n <- length(x)
  if (length(y) != n) stop("x e y deben tener la misma longitud.")
  if (check_equispaced && !is_equispaced(x)) {
    stop("Los nodos no son equiespaciados.")
  }
  T <- matrix(0, nrow = n, ncol = n)
  T[, 1] <- y
  if (n >= 2) {
    for (j in 2:n) {
      for (i in n:j) {
        T[i, j] <- T[i, j - 1] - T[i - 1, j - 1]  # Nabla^j y_i
      }
    }
  }
  T
}

# ============================================================
# 4) Evaluacion Newton hacia atras
#    P(x) = y_n + q y_n + q(q+1)/2! ^2 y_n + ...
# ============================================================
newton_backward_eval <- function(x, T, xq) {
  n <- length(x)
  h <- x[2] - x[1]
  q <- (xq - x[n]) / h
  P <- T[n, 1]
  for (k in 2:n) {
```

```
    coef <- rising_prod(q, k - 1) / factorial(k - 1)
    P <- P + coef * T[n, k]
  }
  P
}

# ============================================================
# 5) Helpers para tabla "bonita" (opcional)
# ============================================================
format_diff_table <- function(x, T, type = c("forward", "backward")) {
  type <- match.arg(type)
  n <- length(x)
  df <- data.frame(x = x, y = T[, 1])
  colnames(df) <- c("x", "y")
  for (j in 2:n) {
    colname <- if (type == "forward") paste0("Delta^", j - 1)
               else paste0("Nabla^", j - 1)
    df[[colname]] <- T[, j]
  }
  df
}

# ============================================================
# 6) Ejemplos de uso
# ============================================================

## Ejemplo A: f(x) = x^2, x = 0,1,2,3; evaluar en 0.5 (adelante)
xA <- 0:3
yA <- xA^2
TA <- forward_diff_table(xA, yA)           # tabla
PA <- newton_forward_eval(xA, TA, 0.5)     # ~ 0.25
dfA <- format_diff_table(xA, TA, "forward")
# print(dfA); cat("P(0.5) =", PA, "\n")

## Ejemplo B: f(x) = x^3, x = 1,2,3,4; evaluar en 1.5 (adelante)
xB <- 1:4
yB <- xB^3
TB <- forward_diff_table(xB, yB)
PB <- newton_forward_eval(xB, TB, 1.5)     # ~ 3.375
dfB <- format_diff_table(xB, TB, "forward")

## Ejemplo C: f(x) = x^2, x = 0,1,2,3; evaluar en 2.6 (atras)
xC <- 0:3
yC <- xC^2
TC <- backward_diff_table(xC, yC)
```

```
PC <- newton_backward_eval(xC, TC, 2.6)    # ~ 6.76
dfC <- format_diff_table(xC, TC, "backward")

## Ejemplo D: f(x) = x^3, x = 1,2,3,4; evaluar en 3.2 (atras)
xD <- 1:4
yD <- xD^3
TD <- backward_diff_table(xD, yD)
PD <- newton_backward_eval(xD, TD, 3.2)    # ~ 32.768
dfD <- format_diff_table(xD, TD, "backward")
```

# Capítulo 6

# Solución numérica de Ecuaciones Diferenciales

Muchas ecuaciones diferenciales ordinarias (EDO) que aparecen en ingeniería, física, química, biología, economía y otras ciencias no tienen solución analítica cerrada. En tales casos recurrimos a **métodos numéricos** para aproximar la solución con una precisión controlada.

Consideraremos problemas de valor inicial (PVI) de la forma

$$y' = f(t, y), \qquad y(t_0) = y_0.$$

La idea general es aproximar la solución $y(t)$ en un conjunto discreto de puntos

$$t_n = t_0 + nh, \qquad n = 0, 1, 2, \ldots, N,$$

donde $h > 0$ es el **tamaño de paso**, y construir una sucesión $\{y_n\}$ que aproxime

$$y_n \approx y(t_n).$$

Un $h$ más pequeño suele producir mayor precisión pero requiere más cómputo; un $h$ demasiado grande puede llevar a errores grandes o incluso inestabilidades numéricas.

## 6.1. Método de Euler (explícito)

Partimos del PVI

$$y' = f(t, y), \qquad y(t_0) = y_0.$$

La derivada se interpreta como razón de cambio:

$$y'(t) \approx \frac{y(t+h) - y(t)}{h}.$$

Aproximando $y'(t_n)$ por $f(t_n, y_n)$, se obtiene

$$\frac{y_{n+1} - y_n}{h} \approx f(t_n, y_n),$$

es decir,

$$\boxed{y_{n+1} = y_n + h\, f(t_n, y_n)}$$

Este es el **método de Euler explícito**.

**Ejemplo 1 (Euler): cálculo a mano**

Resolver numéricamente el PVI

$$y' = y - t^2 + 1, \qquad y(0) = 0.5,$$

en el intervalo $[0, 1]$ con tamaño de paso $h = 0.2$.

Definimos

$$f(t, y) = y - t^2 + 1.$$

Los puntos de malla son:

$$t_0 = 0.0, \quad t_1 = 0.2, \quad t_2 = 0.4, \quad t_3 = 0.6, \quad t_4 = 0.8, \quad t_5 = 1.0.$$

**Paso 0. Condición inicial.**

$$t_0 = 0, \qquad y_0 = 0.5.$$

**Paso 1: de $t_0 = 0$ a $t_1 = 0.2$.**

$$f(t_0, y_0) = f(0, 0.5) = 0.5 - 0^2 + 1 = 1.5.$$

$$y_1 = y_0 + hf(t_0, y_0) = 0.5 + 0.2(1.5) = 0.5 + 0.3 = 0.8.$$

**Paso 2: de $t_1 = 0.2$ a $t_2 = 0.4$.**

$$f(t_1, y_1) = f(0.2, 0.8) = 0.8 - 0.2^2 + 1 = 0.8 - 0.04 + 1 = 1.76.$$

$$y_2 = y_1 + 0.2f(t_1, y_1) = 0.8 + 0.2(1.76) = 0.8 + 0.352 = 1.152.$$

**Paso 3: de $t_2 = 0.4$ a $t_3 = 0.6$.**

$$f(t_2, y_2) = f(0.4, 1.152) = 1.152 - 0.4^2 + 1 = 1.152 - 0.16 + 1 = 1.992.$$

$$y_3 = 1.152 + 0.2(1.992) = 1.152 + 0.3984 = 1.5504.$$

**Paso 4: de $t_3 = 0.6$ a $t_4 = 0.8$.**

$$f(t_3, y_3) = f(0.6, 1.5504) = 1.5504 - 0.6^2 + 1 = 1.5504 - 0.36 + 1 = 2.1904.$$

$$y_4 = 1.5504 + 0.2(2.1904) = 1.5504 + 0.43808 = 1.98848.$$

**Paso 5: de $t_4 = 0.8$ a $t_5 = 1.0$.**

$$f(t_4, y_4) = f(0.8, 1.98848) = 1.98848 - 0.8^2 + 1 = 1.98848 - 0.64 + 1 = 2.34848.$$

$$y_5 = 1.98848 + 0.2(2.34848) = 1.98848 + 0.469696 = 2.458176.$$

**Tabla resumen (Euler).**

| $n$ | $t_n$ | $f(t_n, y_n)$ | $y_n$ |
|---|---|---|---|
| 0 | 0.0 | 1.50000 | 0.50000 |
| 1 | 0.2 | 1.76000 | 0.80000 |
| 2 | 0.4 | 1.99200 | 1.15200 |
| 3 | 0.6 | 2.19040 | 1.55040 |
| 4 | 0.8 | 2.34848 | 1.98848 |
| 5 | 1.0 | – | 2.45818 |

**Implementación en R (Euler, versión manual)**

```
# Método de Euler para y' = y - t^2 + 1, y(0) = 0.5

f <- function(t, y) {
  y - t^2 + 1
}

t0 <- 0
y0 <- 0.5
T  <- 1
h  <- 0.2

t_vals <- seq(t0, T, by = h)
N <- length(t_vals)

y_vals <- numeric(N)
y_vals[1] <- y0  # condición inicial

for (n in 1:(N-1)) {
  t_n <- t_vals[n]
  y_n <- y_vals[n]
  y_vals[n+1] <- y_n + h * f(t_n, y_n)
}

data.frame(n = 0:(N-1),
           t = t_vals,
           y_aprox = y_vals)
```

**Implementación en R (Euler, versión automatizada)**

```
# Implementación genérica del método de Euler

euler <- function(f, t0, y0, T, h) {
  t_vals <- seq(t0, T, by = h)
  N <- length(t_vals)
  y_vals <- numeric(N)
```

```
  y_vals[1] <- y0

  for (n in 1:(N-1)) {
    y_vals[n+1] <- y_vals[n] + h * f(t_vals[n], y_vals[n])
  }

  data.frame(t = t_vals, y = y_vals)
}

# Ejemplo de uso:
f <- function(t, y) y - t^2 + 1
sol_euler <- euler(f = f, t0 = 0, y0 = 0.5, T = 1, h = 0.2)
```

## 6.2. Método de Euler Mejorado (Heun)

El método de Euler mejorado (Heun) utiliza dos evaluaciones de $f$ por paso.

$$k_1 = f(t_n, y_n),$$

$$\tilde{y}_{n+1} = y_n + hk_1,$$

$$k_2 = f(t_n + h, \tilde{y}_{n+1}),$$

$$\boxed{y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2)}$$

Este método es de orden 2 en el error global, por lo que proporciona mayor precisión que Euler con el mismo tamaño de paso.

**Ejemplo 2 (Heun): cálculo a mano**

Resolvámos el PVI

$$y' = y + t, \qquad y(0) = 1,$$

en el intervalo $[0, 0.3]$ con $h = 0.1$.

Definimos

$$f(t, y) = y + t.$$

Los puntos de malla:

$$t_0 = 0.0, \quad t_1 = 0.1, \quad t_2 = 0.2, \quad t_3 = 0.3.$$

**Paso 0. Condición inicial.**

$$t_0 = 0, \qquad y_0 = 1.$$

**Paso 1: de $t_0 = 0$ a $t_1 = 0.1$.**

$$k_1 = f(0, 1) = 1 + 0 = 1.$$

$$\tilde{y}_1 = y_0 + hk_1 = 1 + 0.1(1) = 1.1.$$

$$k_2 = f(0.1, 1.1) = 1.1 + 0.1 = 1.2.$$

$$y_1 = y_0 + \frac{h}{2}(k_1 + k_2) = 1 + 0.05(1 + 1.2) = 1 + 0.05(2.2) = 1.11.$$

Los pasos siguientes se calculan de forma análoga.

**Implementación en R (Heun, versión manual)**

```
# Método de Heun para y' = y + t, y(0) = 1

f <- function(t, y) {
  y + t
}

t0 <- 0
y0 <- 1
T  <- 0.3
h  <- 0.1

t_vals <- seq(t0, T, by = h)
N <- length(t_vals)
y_vals <- numeric(N)
y_vals[1] <- y0

for (n in 1:(N-1)) {
  t_n <- t_vals[n]
  y_n <- y_vals[n]

  k1 <- f(t_n, y_n)
  y_tilde <- y_n + h * k1
  k2 <- f(t_n + h, y_tilde)

  y_vals[n+1] <- y_n + (h/2) * (k1 + k2)
}

data.frame(n = 0:(N-1),
           t = t_vals,
           y_aprox = y_vals)
```

**Implementación automatizada en R (Heun)**

```
# Implementación genérica del método de Heun
```

```
heun <- function(f, t0, y0, T, h) {
  t_vals <- seq(t0, T, by = h)
  N <- length(t_vals)
  y_vals <- numeric(N)
  y_vals[1] <- y0

  for (n in 1:(N-1)) {
    t_n <- t_vals[n]
    y_n <- y_vals[n]

    k1 <- f(t_n, y_n)
    y_tilde <- y_n + h * k1
    k2 <- f(t_n + h, y_tilde)

    y_vals[n+1] <- y_n + (h/2) * (k1 + k2)
  }

  data.frame(t = t_vals, y = y_vals)
}

# Ejemplo de uso
f <- function(t, y) y + t
sol_heun <- heun(f = f, t0 = 0, y0 = 1, T = 0.3, h = 0.1)
```

## 6.3. Método de Runge–Kutta de cuarto orden (RK4)

El método de Runge–Kutta de cuarto orden (RK4) calcula cuatro pendientes:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f\left(t_n + \frac{h}{2},\, y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2},\, y_n + \frac{h}{2}k_2\right), \\ k_4 &= f(t_n + h,\, y_n + hk_3)\,. \end{aligned}$$

La actualización es

$$\boxed{y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4).}$$

Este método es de orden 4, con error global $O(h^4)$.

**Ejemplo 3 (RK4): cálculo a mano**

Consideremos

$$y' = -3y, \qquad y(0) = 5,$$

con $h = 0.2$. El primer paso (de $t_0 = 0$ a $t_1 = 0.2$) se calcula como:

$$k_1 = f(0,5) = -3 \cdot 5 = -15,$$
$$k_2 = f\left(0.1, 5 + \frac{0.2}{2}(-15)\right) = f(0.1, 3.5) = -3 \cdot 3.5 = -10.5,$$
$$k_3 = f\left(0.1, 5 + \frac{0.2}{2}(-10.5)\right) = f(0.1, 3.95) = -3 \cdot 3.95 = -11.85,$$
$$k_4 = f\left(0.2, 5 + 0.2(-11.85)\right) = f(0.2, 2.63) = -3 \cdot 2.63 = -7.89.$$

$$y_1 = 5 + \frac{0.2}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big).$$

**Implementación en R (RK4 manual)**

```
# Método RK4 para y' = -3y, y(0) = 5

f <- function(t,y) {
  -3*y
}

t0 <- 0
y0 <- 5
T  <- 0.6
h  <- 0.2

t_vals <- seq(t0, T, by=h)
N <- length(t_vals)
y_vals <- numeric(N)
y_vals[1] <- y0

for (n in 1:(N-1)) {
  t_n <- t_vals[n]
  y_n <- y_vals[n]

  k1 <- f(t_n, y_n)
  k2 <- f(t_n + h/2, y_n + h*k1/2)
  k3 <- f(t_n + h/2, y_n + h*k2/2)
  k4 <- f(t_n + h,   y_n + h*k3)

  y_vals[n+1] <- y_n + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
}

data.frame(n = 0:(N-1),
           t = t_vals,
           y_aprox = y_vals)
```

**Implementación automatizada en R (RK4)**

```
# Implementación genérica del método RK4

rk4 <- function(f, t0, y0, T, h) {
  t_vals <- seq(t0, T, by=h)
  N <- length(t_vals)
  y_vals <- numeric(N)
  y_vals[1] <- y0

  for (n in 1:(N-1)) {
    t_n <- t_vals[n]
    y_n <- y_vals[n]

    k1 <- f(t_n, y_n)
    k2 <- f(t_n + h/2, y_n + h*k1/2)
    k3 <- f(t_n + h/2, y_n + h*k2/2)
    k4 <- f(t_n + h,   y_n + h*k3)

    y_vals[n+1] <- y_n + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
  }

  data.frame(t = t_vals, y = y_vals)
}

# Ejemplo de uso
f <- function(t, y) -3*y
sol_rk4 <- rk4(f = f, t0 = 0, y0 = 5, T = 0.6, h = 0.2)
```

## 6.4. Método de Euler implícito

Para el PVI

$$y' = f(t, y), \qquad y(t_0) = y_0,$$

el método de Euler implícito define

$$\boxed{y_{n+1} = y_n + h\, f(t_{n+1}, y_{n+1}).}$$

Aquí $y_{n+1}$ aparece en ambos lados. En general, para $f$ no lineal en $y$, se debe resolver en cada paso la ecuación

$$G(y_{n+1}) := y_{n+1} - h\, f(t_{n+1}, y_{n+1}) - y_n = 0$$

mediante un método numérico (por ejemplo, Newton).

**Caso lineal:** $f(t, y) = a(t)y + b(t)$

Si

$$f(t, y) = a(t)y + b(t),$$

entonces

$$y_{n+1} = y_n + h\,[a(t_{n+1})y_{n+1} + b(t_{n+1})].$$

Reordenando:

$$y_{n+1}\big(1 - ha(t_{n+1})\big) = y_n + hb(t_{n+1}),$$

$$\boxed{y_{n+1} = \frac{y_n + hb(t_{n+1})}{1 - ha(t_{n+1})}.}$$

**Ejemplo 4 (Euler implícito): cálculo a mano**
Consideremos la ecuación rígida

$$y' = -50y + 10, \qquad y(0) = 0,$$

en el intervalo $[0, 0.2]$ con $h = 0.05$.

Aquí

$$a(t) = -50, \qquad b(t) = 10.$$

Entonces

$$y_{n+1} = \frac{y_n + h \cdot 10}{1 - h(-50)} = \frac{y_n + 0.5}{1 + 2.5} = \frac{y_n + 0.5}{3.5}.$$

Puntos de malla:

$$t_0 = 0.00, \quad t_1 = 0.05, \quad t_2 = 0.10, \quad t_3 = 0.15, \quad t_4 = 0.20.$$

**Paso 0. Condición inicial.**

$$y_0 = 0.$$

**Paso 1.**

$$y_1 = \frac{0 + 0.5}{3.5} \approx 0.142857.$$

**Paso 2.**

$$y_2 = \frac{y_1 + 0.5}{3.5} = \frac{0.142857 + 0.5}{3.5} \approx 0.183673.$$

**Paso 3.**

$$y_3 = \frac{0.183673 + 0.5}{3.5} \approx 0.195335.$$

**Paso 4.**

$$y_4 = \frac{0.195335 + 0.5}{3.5} \approx 0.198667.$$

**Tabla resumen (Euler implícito).**

| $n$ | $t_n$ | $y_n$ |
|---|---|---|
| 0 | 0.00 | 0.000000 |
| 1 | 0.05 | 0.142857 |
| 2 | 0.10 | 0.183673 |
| 3 | 0.15 | 0.195335 |
| 4 | 0.20 | 0.198667 |

**Implementación en R (Euler implícito, versión manual)**

```
# Euler implícito para y' = -50y + 10, y(0) = 0, h = 0.05

h <- 0.05
t_vals <- seq(0, 0.2, by = h)
N <- length(t_vals)

y_vals <- numeric(N)
y_vals[1] <- 0  # y0

for (n in 1:(N-1)) {
  y_vals[n+1] <- (y_vals[n] + 0.5) / 3.5
}

data.frame(n = 0:(N-1),
           t = t_vals,
           y_aprox = y_vals)
```

**Implementación automatizada en R (Euler implícito lineal)**

```
# Euler implícito para y' = a(t)*y + b(t)

euler_imp_lineal <- function(a_fun, b_fun, t0, y0, T, h) {
  t_vals <- seq(t0, T, by = h)
  N <- length(t_vals)
  y_vals <- numeric(N)
  y_vals[1] <- y0

  for (n in 1:(N-1)) {
    t_next <- t_vals[n+1]
    a_next <- a_fun(t_next)
    b_next <- b_fun(t_next)

    # y_{n+1} = (y_n + h*b_next) / (1 - h*a_next)
    y_vals[n+1] <- (y_vals[n] + h * b_next) / (1 - h * a_next)
  }

  data.frame(t = t_vals, y = y_vals)
}

# Ejemplo: y' = -50y + 10
a_fun <- function(t) -50
b_fun <- function(t) 10

sol_imp <- euler_imp_lineal(a_fun = a_fun,
                            b_fun = b_fun,
                            t0 = 0, y0 = 0,
                            T = 0.2, h = 0.05)
```

## 6.5. Ejercicios propuestos

Resuelva numéricamente con Euler, construyendo la tabla de iteraciones y, si es posible, comparando con la solución exacta.

1. $y' = y - t, \quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
2. $y' = 3t^2 - y, \quad y(0) = 2, \quad h = 0.05, \quad$ intervalo $[0, 0.5]$.
3. $y' = \sin(t) - y^2, \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
4. $y' = ty, \quad y(0) = 1, \quad h = 0.2, \quad$ intervalo $[0, 1]$.
5. $y' = e^{-t} - y, \quad y(0) = 3, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
6. $y' = y(1 - y)$ (modelo logístico), $\quad y(0) = 0.2, \quad h = 0.1, \quad$ intervalo $[0, 2]$.
7. $y' = 4y + t, \quad y(0) = 1, \quad h = 0.05, \quad$ intervalo $[0, 0.5]$.
8. $y' = \cos(t) + y, \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
9. $y' = -5y, \quad y(0) = 2, \quad h = 0.2, \quad$ intervalo $[0, 1]$.
10. $y' = t^3 - y^2, \quad y(0) = 1, \quad h = 0.05, \quad$ intervalo $[0, 0.5]$.

Aplique el método de Euler mejorado (Heun), mostrando los valores de $k_1$ y $k_2$ en cada paso.

1. $y' = y + t^2, \quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
2. $y' = \cos(t) - y, \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
3. $y' = t - y^2, \quad y(0) = 0.5, \quad h = 0.05, \quad$ intervalo $[0, 0.5]$.
4. $y' = 2y + e^t, \quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
5. $y' = y(1 + t), \quad y(0) = 1, \quad h = 0.2, \quad$ intervalo $[0, 1]$.
6. $y' = -3y + t, \quad y(0) = 2, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
7. $y' = t^2\sqrt{y}$ (suponer $y \geq 0$), $\quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
8. $y' = y^2 - y, \quad y(0) = 0.1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
9. $y' = 1 - ty, \quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
10. $y' = 5 - 4y, \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.

Use el método de Runge–Kutta de cuarto orden, calculando explícitamente $k_1, k_2, k_3, k_4$.

1. $y' = y - t^2 + 1, \quad y(0) = 0.5, \quad h = 0.2, \quad$ intervalo $[0, 1]$.
2. $y' = -2y + \sin(t), \quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.

3. $y' = y^2 + t, \quad y(0) = 0, \quad h = 0.05, \quad$ intervalo $[0, 0.5]$.
4. $y' = e^{-t} - y, \quad y(0) = 2, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
5. $y' = 3y(1 - y)$ (modelo logístico), $\quad y(0) = 0.1, \quad h = 0.1, \quad$ intervalo $[0, 2]$.
6. $y' = t\cos(y), \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
7. $y' = -5y + 5, \quad y(0) = 0, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
8. $y' = \sqrt{t + y}$ (suponer $t + y \geq 0$), $\quad y(0) = 1, \quad h = 0.1, \quad$ intervalo $[0, 1]$.
9. $y' = y\ln(t + 1), \quad y(0) = 1, \quad h = 0.2, \quad$ intervalo $[0, 2]$.
10. $y' = -50y + \cos(t), \quad y(0) = 0, \quad h = 0.02, \quad$ intervalo $[0, 0.4]$.

# Capítulo 7

# Tareas del curso

**Ejercicio 24.** *Convertir los siguientes números de base 10 a base 2.*

1. 324
2. 27
3. 1423
4. 235.25
5. 41.596

**Ejercicio 25.**

1. *Realizar una revisión de la historia de los métodos numéricos, elaborar un documento de hasta dos cuartillas.*
2. *Realiza las siguientes conversiones de base* 10 *a base* 2*:*
   a) *246*
   b) *345.68*
   c) *4586632.2846*
   d) *984365.27463*
   e) *79905523*
3. *Elabora el código en R para realizar la conversión de base* 10 *a base* 2*.*
4. *Describe exhaustivamente los tipos de errores que existen*

**Nota 32.** *En los siguientes ejercicios se indicará cuales ejercicios pueden realizarse sin el apoyo de R, se deben de resolver al menos dos ejercicios de cada serie sin el apoyo de R, es decir, se tienen que resolver manualmente.*

**Ejercicio 26.** *Resolver por eliminación Gaussiana Simple los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} x_1 - 2x_2 + 0.5x_3 &= -5 \\ -2x_1 + 5x_2 - 1.5x_3 &= 0 \\ -0.2x_1 + 1.75x_2 - x_3 &= 10 \end{aligned}$$

*2.*

$$\begin{aligned} 3x_1 - x_2 + 6x_4 &= 2.3 \\ 4x_1 + 2x_2 - x_3 - 5x_4 &= 6.9 \\ -5x_1 + x_2 - 3x_3 &= -36 \\ 10x_2 - 4x_3 + 7x_4 &= -36 \end{aligned}$$

*3.*

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

*utilizar redondeo a 4 cifras significativas.*

*4.*

$$\begin{aligned} 4x_1 + 2x_2 &= 2 \\ 2x_1 + 3x_2 + x_3 &= -1 \\ x_2 + \frac{5}{2}x_3 &= 3 \end{aligned}$$

*5.*

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*6.* ***

$$\begin{aligned} 8x_1 + 2x_2 - 2x_3 &= -2 \\ 10x_1 + 2x_2 + 4x_3 &= 4 \\ 12x_1 + 2x_2 + 2x_3 &= 6 \end{aligned}$$

*7.* ***

$$\begin{aligned} 5x_1 + 2x_2 + x_3 - x_4 &= 1 \\ 2x_1 - x_2 + 3x_3 + 2x_4 &= 12 \\ 4x_1 + x_2 - 2x_3 + 3x_4 &= 5 \\ -2x_1 + 2x_2 + x_3 + x_4 &= 2 \end{aligned}$$

*8.* ***

$$\begin{aligned} 2x_1 + 3x_2 + 2x_3 + 4x_4 &= 4 \\ 4x_1 + 10x_2 - 4x_3 &= -8 \\ -3x_1 - 2x_2 - 5x_3 - 2x_4 &= -4 \\ -2x_1 + 4x_2 + 4x_3 - 7x_4 &= -1 \end{aligned}$$

*9.*

$$\begin{aligned} 1.133x_1 + 5.281x_2 - 2.454x_3 &= 6.414 \\ 24.14x_1 - 1.21x_2 + 5.281x_3 &= 113.8 \\ -10.123x_1 + 6.387x_2 - x_3 &= 1 \end{aligned}$$

*10.*

$$A = \begin{pmatrix} 2 & 3 & 2 & 4 \\ 4 & 10 & -4 & 0 \\ -3 & -2 & -5 & -2 \\ -2 & 4 & 4 & -7 \end{pmatrix}$$

*y*

$$b = \begin{pmatrix} \\ 9 \\ -15 \\ 6 \\ 2 \end{pmatrix}$$

**Ejercicio 27.** *Resolver por eliminación gaussiana con pivoteo parcial los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} 0.4x_1 - 1.5x_2 + 0.75x_3 &= -20 \\ -0.5x_1 - 15x_2 + 10x_3 &= -10 \\ -10x_1 - 9x_2 + 2.5x_3 &= 30 \end{aligned}$$

*2.* ***

$$\begin{aligned} 5x_1 - 8x_2 + x_3 &= -71 \\ -2x_1 + 6x_2 - 9x_3 &= 134 \\ 3x_1 - 5x_2 + 2x_3 &= -58 \end{aligned}$$

*3.*

$$\begin{aligned} 0.003000x_1 + 59.14x_2 &= 59.17 \\ 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

*utilizar redondeo a 4 cifras significativas.*

*4.* ***

$$\begin{aligned} -x_2 + 4x_3 - x_4 &= -1 \\ -x_1 + 4x_2 - x_3 &= 2 \\ -x_1 - x_3 + 4x_4 &= 4 \\ 4x_1 - x_2 &= 10 \end{aligned}$$

*5.*

$$\begin{aligned} 0.00031000x_1 + 1.000000x_2 &= 3.000000 \\ 1.00045534x_1 + 1.00034333x_2 &= 7.000 \end{aligned}$$

*6.* ***** *Resolver para* $A = \begin{pmatrix} 14 & 14 & -9 & 3 & -5 \\ 14 & 52 & -15 & 2 & -32 \\ -9 & -15 & 36 & -5 & 16 \\ 3 & 2 & -5 & 47 & 49 \\ -5 & 32 & 16 & 49 & 79 \end{pmatrix}$, $b = \begin{pmatrix} -15 \\ -100 \\ 106 \\ 329 \\ 463 \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$

*7. Resolver para* $A = \begin{pmatrix} \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & 1 & 2 \end{pmatrix}$, $b = \begin{pmatrix} 9 \\ 8 \\ 8 \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$

*8. Resolver para* $A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{pmatrix}$, $b = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{7} \\ \frac{1}{8} \\ \frac{1}{9} \end{pmatrix}$ *y* $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$

*9. Resolver el sistema*

$$\begin{aligned} 2x_1 + x_2 - x_3 + x_4 - 3x_5 &= 7 \\ x_1 + 2x_3 - x_4 + x_5 &= 2 \\ -2x_2 - x_3 + x_4 - x_5 &= -5 \\ 3x_1 + x_2 - 4x_3 + 5x_5 &= 6 \\ x_1 - x_2 - x_3 - x_4 + x_5 &= 3 \end{aligned}$$

*10. Resolver el sistema*

$$\begin{aligned} 3.333x_1 + 15920x_2 - 10.333x_3 &= 15913 \\ 2.222x_1 + 16.71x_2 9.612x_2 &= 28.544 \\ 1.5611x_1 + 5.1791x_2 + 1.6852x_3 &= 8.4254 \end{aligned}$$

**Ejercicio 28.** *Resolver por el método de Gauss-Jordan los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 1 \\ -0.4x_1 + 2x_2 - x_3 &= 10 \\ 0.5x_1 - 3x_2 + x_3 &= 15 \end{aligned}$$

*2.*

$$\begin{aligned} 2x_1 - 0.9x_2 + 3x_3 &= -3.61 \\ -0.5x_1 + 0.1x_2 - x_3 &= 2.035 \\ x_1 - 6.35x_2 - 0.45x_3 &= 15.401 \end{aligned}$$

*3.*

$$\begin{aligned} 0.7x_1 + 2.7x_2 - 6x_3 + 0.7x_4 &= 1.6487 \\ 2x_1 - 0.8x_2 + 3x_3 - x_4 &= -2.342 \\ -x_1 - 1.5x_2 + 1.4x_3 + 3x_4 &= -4.189 \\ 7x_2 - 1.56x_3 + x_4 = 15.792 & \end{aligned}$$

*4.*

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*5.* ***

$$\begin{aligned} 10x_1 + 2x_2 - x_3 &= 27 \\ -3x_1 - 6x_2 + 2x3 &= -61.5 \\ x_1 + x_2 + 5x_3 &= -21.5 \end{aligned}$$

*6.* $A = \begin{pmatrix} 1 & 3 & -2 & 1 \\ 1 & 3 & -1 & 2 \\ 0 & 1 & -1 & 4 \\ 2 & 6 & 1 & 2 \end{pmatrix}$ *y* $b = \begin{pmatrix} 4 \\ 1 \\ 5 \\ 2 \end{pmatrix}$

*7.*

$$\begin{aligned} 6x_1 - x_2 - x_3 + 4x_4 &= 17 \\ x_1 - 10x_2 + 2x_3 - x_4 &= -17 \\ 3x_1 - 2x_2 + 8x_3 - x_4 &= 19 \\ x_1 + x_2 + x_3 - 5x_4 &= -14 \end{aligned}$$

*8.* ***

$$\begin{aligned} x + 2y + 3z + 4w &= 1 \\ x - 4y + z + 11w &= 2 \\ -x + 8y + 7z + 6w &= -2 \\ 16x + 8y - 5z + 6w &= 11 \end{aligned}$$

*9.* ***

$$\begin{aligned} x_1 + x_2 &= 3 \\ x_1 + 2x_2 + x_3 &= -1 \\ x_2 + 3x_3 + x_4 &= 2 \\ x_3 + 4x_4 + x_5 &= 1 \\ x_4 + 5x_5 &= 3 \end{aligned}$$

*10.*

$$\begin{aligned} 15x_1 - 18x_2 + 15x_3 - 3x_4 &= 11 \\ -18x_1 + 24_2 - 18x_3 + 4x_4 &= 10 \\ 15x_1 - 18x_2 + 18x_3 - 3x_4 &= 11 \\ -3x_1 + 4x_2 - 3x_3 + x_4 &= 13 \end{aligned}$$

**Ejercicio 29.** *Resolver por el método de Gauss-Seidel los siguientes sistemas de ecuaciones lineales*

*1.*

$$\begin{aligned} 3x_1 - 0.2x_2 - 0.5x_3 &= 8 \\ 0.1x_1 + 7x_2 + 0.4x_3 &= -19.5 \\ 0.4x_1 - 0.1x_2 + 10x_3 &= 72.4 \end{aligned}$$

*2.*

$$\begin{aligned} -5x_1 + 1.4x_2 - 2.7x_3 &= 94.2 \\ 0.7x_1 - 2.5x_2 + 15x_3 &= -6 \\ 3.3x_1 - 11x_2 + 4.4x_3 &= -27.5 \end{aligned}$$

*3.*

$$\begin{aligned} 3x_1 - 0.5x_2 + 0.6x_3 &= 5.24 \\ 0.3x_1 - 4x_2 - x_3 &= -0.387 \\ -0.7x_1 + 2x_2 + 7x_3 &= 14.803 \end{aligned}$$

*4.*

$$\begin{aligned} 5x_1 - 0.2x_2 + x_3 &= 1.5 \\ 0.1x_1 + 3x_2 - 0.5x_3 &= -2.7 \\ -0.3x_1 + x_2 - 7x_3 &= 9.5 \end{aligned}$$

*5.* ***

$$\begin{aligned} -3x_2 + 7x_3 &= 2 \\ x_1 + 2x_2 - x_3 &= 3 \\ 12x_1 + 2x_2 + 2x_3 &= 6 \end{aligned}$$

*6.*

$$\begin{aligned} 0.15_1 + 2.11x_2 + 30.75x_3 &= -26.38 \\ 0.64x_1 + 1.21x_2 + 2.05x_3 &= 1.01 \\ 3.21x_1 + 1.53x_2 + 1.04x_3 &= 5.23 \end{aligned}$$

*7.* ***

$$\begin{aligned} x_1 + x_2 - x_3 &= -3 \\ 6x_1 + 2x_2 + 2x_3 &= 2 \\ -3x_1 + 4x_2 + x_3 &= 1 \end{aligned}$$

*8.* ***

$$\begin{aligned} 2x_1 + x_2 - x3 &= 1 \\ 5x_1 + 2x_2 + 2x_3 &= -4 \\ 3x_1 + x_2 + x_3 &= 5 \end{aligned}$$

*9.*

$$\begin{aligned} 3x - 0.1y - 0.2z &= 7.85 \\ 0.1x + 7y - 0.3z &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

*10.*

$$\begin{aligned} 17x_1 - 2x_2 - 3x_3 &= 500 \\ -5x_1 + 21x_2 - 2x_3 &= 200 \\ -5x_1 - 5x_2 + 22x_3 &= 30 \end{aligned}$$

**Ejercicio 30.** *Aplicar el método de Jacobi para resolver los siguientes sistemas de ecuaciones lineales*

1. $A = \left(\begin{array}{cccc|c} 10 & 2 & -1 & 0 & 26 \\ 1 & 20 & -2 & 3 & -15 \\ -2 & 1 & 30 & 0 & 53 \\ 1 & 2 & 3 & 20 & 47 \end{array}\right)$

2. $*A = \left(\begin{array}{ccc|c} -1 & 2 & 10 & 11 \\ 11 & -1 & 2 & 12 \\ 1 & 5 & 2 & 8 \end{array}\right)$

3. $A = \left(\begin{array}{ccc|c} 8 & 2 & 3 & 51 \\ 2 & 5 & 1 & 23 \\ -3 & 1 & 6 & 20 \end{array}\right)$

4. $A = \left(\begin{array}{cccc|c} 2 & -1 & 1 & 3 & 10 \\ 2 & 2 & 2 & 2 & 1 \\ -1 & -1 & 2 & 2 & -5 \\ 3 & 1 & -1 & 4 & 6 \end{array}\right)$

5. $A = \left(\begin{array}{cccc|c} 3 & 1 & 1 & -1 & 5 \\ 0 & 2 & 1 & 4 & 0 \\ 1 & 1 & -1 & 9 & 1 \\ 2 & 4 & 6 & 3 & 0 \end{array}\right)$

6. $A = \left(\begin{array}{cccc|c} 10 & -1 & 2 & 0 & 6 \\ -1 & 11 & -1 & 3 & 25 \\ 2 & -1 & 10 & -1 & -11 \\ 0 & 2 & -1 & 8 & 15 \end{array}\right)$

7. *

$$\begin{aligned} x_1 + 2x_2 - 2x_3 &= 7 \\ x_1 + x_2 + x_3 &= 2 \\ 2x_1 + 2x_2 + x_3 &= 5 \end{aligned}$$

8.

$$\begin{aligned} -4x_1 + 14x_2 &= 10 \\ -5x_1 + 13x_2 &= 8 \\ -x_1 + 2x_3 &= 1 \end{aligned}$$

*9.* ***

$$\begin{aligned} x + y + 2z &= 1 \\ x + 2y + z &= 1 \\ 2x + y + z &= 1 \end{aligned}$$

*10.*

$$\begin{aligned} 6x_1 - 2x_2 + 2x_3 + 4x_4 &= 10 \\ 12x_1 - 8x_2 + 6x_3 + 10x_4 &= 20 \\ 3x_1 - 13x_2 + 9x_3 + 3x_4 &= 2 \\ -6x_1 + 4x_2 + x_3 - 18x_4 &= -19 \end{aligned}$$

**Ejercicio 31.** *La siguiente serie de ejercicios hay que resolverlos con el apoyo de R, excepto los indicados por un* ***

1. *** Resuelva el sistema*

$$\begin{cases} 2x + y - z = 1, \\ -x + 3y + 2z = 12, \\ x + 2y + 3z = 7 \end{cases}$$

*mediante eliminación gaussiana simple (sin pivoteo).*

2. *Resuelva el mismo sistema anterior pero ahora aplicando eliminación gaussiana con pivoteo parcial. Compare los pasos con el ejercicio anterior.*

3. *Aplique eliminación gaussiana con pivoteo y escalamiento al sistema*

$$\begin{cases} 10x + 2y + z = 7, \\ 2x + 20y + 2z = 9, \\ x + 2y + 30z = 12 \end{cases}$$

*y analice la importancia del escalamiento.*

4. ***Utilice el método de Gauss–Jordan para calcular la inversa de*

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

5. ***Resuelva $Ax = b$ con $A$ y $b$ dados por*

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

*utilizando factorización $LU$ y sustitución hacia adelante y hacia atrás.*

6. *Calcule la factorización de Cholesky de*

$$A = \begin{bmatrix} 25 & 15 & -5 \\ 15 & 18 & 0 \\ -5 & 0 & 11 \end{bmatrix}$$

*y resuelva* $Ax = b$ *con* $b = (35, 33, 6)^\top$.

7. ***Resuelva mediante sustitución hacia atrás el sistema triangular superior:*

$$\begin{cases} 2x + 3y - z = 5, \\ -y + 2z = 4, \\ 3z = 6. \end{cases}$$

8. ***Resuelva mediante sustitución hacia adelante el sistema triangular inferior:*

$$\begin{cases} x = 3, \\ 2y + x = 5, \\ z - y + 2x = 10. \end{cases}$$

10. *Aplique el método de Jacobi para resolver*

$$\begin{cases} 10x - y + 2z = 6, \\ -x + 11y - z + 3w = 25, \\ 2x - y + 10z - w = -11, \\ 3y - z + 8w = 15, \end{cases}$$

*realizando 3 iteraciones con* $x^{(0)} = \mathbf{0}$.

11. *Repita el ejercicio anterior con el método de Gauss–Seidel. Compare la velocidad de convergencia con Jacobi.*

12. *Aplique el método SOR con* $\omega = 1.25$ *al mismo sistema y compare las tres trayectorias de convergencia.*

13. *Escriba la matriz de iteración* $T_J$ *y el vector* $c_J$ *del método de Jacobi para el sistema*

$$\begin{cases} 4x + y = 9, \\ x + 3y = 7. \end{cases}$$

*Verifique si* $\rho(T_J) < 1$.

14. *Investigue experimentalmente en R cuál es el valor óptimo aproximado de* $\omega$ *para el método SOR en el sistema* $4 \times 4$ *del ejercicio 10.*

15. *Genere una matriz aleatoria simétrica definida positiva* $5 \times 5$ *en R y resuelva* $Ax = b$*:*

*(a) Usando factorización $LU$.*

*(b) Usando factorización de Cholesky.*

*(c) Usando Gauss–Seidel con 20 iteraciones.*

*Compare el tiempo de cómputo y la precisión de cada método.*

**Ejercicio 32.** *Genera un archivo tipo Rmd con las prácticas realizadas en el laboratorio, en las que deberá de desplegarse una barra lateral izquierda con un nivel de profundidad de 4: seccion, subseccion, subsubseccion y subsubsubseccion.*

**Ejercicio 33.** *Revisa en R los códigos para resolver sistemas de ecuaciones lineales, esto en un archivo tipo Rmd, mismo que deberás de entregar.*

# Capítulo 8

# Introducción al uso de R

**Programas y rutinas en R**

## 8.1. Sesiones en RStudio

Al utilizar R, existen varios entornos que facilitan la gestión y ejecución de rutinas, *archivos con extensión .R*, el más popular es *RStudio* o bien directamente desde la terminal o ejecutando simplemente *R*, la ventaja de *RStudio* es que permite que en una pantalla podamos visualizar: Consola (lugar donde se ejecutan los comandos directamente), History (el histórico de las variables y funciones definidas mismo que puede guardarse para ser invocado posteriormente), Plots (ventana en la que se muestran los gráficos generados), Help (la ayuda sobre comandos, funciones, sintaxis en R), Files (lugar donde se manejan los archivos, es decir leer, guardar, mover o renombrar archivos), y Packages (espacio para instalar o cargar paquetes de manera gráfica), todo esto para facilitar el manejo y ejecución de rutinas compatibles con R. Por otra parte **Workspace** es un entorno en el que se incluyen todos los objetos definidos, al final de una sesión de R, este entorno puede guardarse una imagen del mismo para ser cargada posteriormente.

Durante el uso de R en ocasiones se requiere limpiar la consola, para esto al presionar **Ctrl+L**.

## 8.2. Uso de R

- **Constantes**: $\pi$, $exp(1)$
- Las constantes pueden ser de tipo *integer*, *double* o *complex*, el tipo de constante se puede consultar con la función **typeof()**

```
> typeof('mi constante')
[1] "character"
```

- Operadores: $<, >, >=, >=, != ,$ (Not), $\|$ (OR), & (And), == (comparar)

- Operadores aritméticos: $+$, $-$, $*$, `^` potencia, `%%` resto de la división entera, `%/%` división entera.
- Logaritmos y exponenciales: `log` logaritmo natural, `log(x,b)` ($log_b x$) y `exp(x)` ($e^x$).
- Funciones trigonométricas`cos(x)`,`sin(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`.
- Funciones misceláneas `abs(x)`, `sqrt(x)`, `floor(x)`, `ceiling(x)`, `max(x)`, `sign(x)`.
- Comando `options(digits=k)`:

```
> 1/3.0
[1] 0.3333333
> options(digits=3)
> 1/3
[1] 0.333
> 1/17.0
[1] 0.0588
> options(digits=3)
> 1/17.0
[1] 0.0588
> options(digits=5)
> 1/17.0
[1] 0.058824
> options(digits=9)
> 1/17.0
[1] 0.0588235294
```

- Comando `round(x,n)` redondea $x$ a $n$ decimales, el valor por defecto es $n = 6$.
- Comando `cat('caracter1','caracter2')` concatena dos cadenas o valores y el resultado lo convierte a un objeto tipo *string*.

## 8.3. Funciones

```
nombrefun = function(a1,a2,...,an) {
# código ...
instruccion-1
instruccion-2
# ...
return( ... ) #valor que retorna (o también la última instrucción, si ésta retorna
}
```

## 8.4. Clase en Laboratorio de Cómputo

**Ejercicio 34.** *1. Generar un archivo tipo Rmd, personalizar de manera básica*

*2. Realizar las siguientes operaciones*

```
x = c(1.1, 1.2, 1.3, 1.4, 1.5)
x = 1:5
x = seq(1,3, by =0.5)
x = seq(5,1, by =-1)
x = rep(1, times = 5)
length(x)
rep(x, 2)
set.seed(123)
x = sample(1:10, 5)
```

*3.*
```
x = 1:5
y = rep(0.5, times = length(x))
x+y
x*y
x^y
1/(1:5)
```

*4.*
```
x = 1:5
2*x
1/x^2
x+3
```

*5.*
```
A = matrix(rep(0,9), nrow = 3, ncol= 3);
B = matrix(c(1,2,3,5,6,7), nrow = 2, byrow=T);
x = 1:3; y = seq(1,2, by = 0.5); z = rep(8, 3) ; x; y; z
C = matrix(c(x,y,z), nrow = length(x)); C # ncol no es necesario declararlo
xi = seq(1,2, by 0.1); yi = seq(5,10, by = 0.5)
rbind(xi,yi)
cbind(xi,yi)
```

*6.*
```
A = diag(c(3,1,3));
diag(A)
n = 3
I = diag(1, n);
D = diag(diag(A))
J = diag(1, 3, 4);
```

*7.*
```
B = matrix(c( 1, 1 ,8,
```

```
                2, 0, 8,
                3, 2, 8), nrow = 3, byrow=TRUE); B

B[2, 3]
B[3,]
B[,2]
B[1:2,c(2,3)]
```

*8.*
```
A = matrix(c( 1, 1 ,8,
                2, 0, 8,
                3, 2, 8), nrow = 3, byrow=TRUE); A
A[c(1,3), ] = A[c(3,1), ]
A[2, ] = A[2, ] - A[2,1]/A[1,1]*A[1, ]
```

*9.*
```
x = c(2, -6, 7, 8, 0.1,-8.5, 3, -7, 3)
which.max(x)
which.max(abs(x))
```

*10.*
```
A = matrix(1:9, nrow=3); A # Por columnas
B = matrix(rep(1,9), nrow=3); B

A+B
A*B
A%*%B
A^2
A-2
3*A
t(A)
det(A)
C <- A-diag(1,3); det(C)
```

*11.*
```
notas = matrix(c(80, 40, 70, 30, 90, 67, 90,
                 40, 40, 30, 90, 100, 67, 90,
                 100,100,100, 100, 70, 76, 95), nrow=3, byrow=TRUE); notas
# crear columa con la suma de los renglones
#agregar la columna al final de la matriz
```

*12.*
```
u=c(1,2)
v=c(-2,3)
w=c(3,-5)
```

```
norma=function(u){sqrt(sum(u^2))}
```

*13.*

```
t(u-2*v)%*%w
norma(u+v+w)
norma(u)+norma(v)+norma(w)
t(u-v)%*%(v-w)
```

*14.*

```
u=c(8,3);a=c(4,-5)
ProyOrto=function(u,a){(t(u)%*%a)*a/norma(a)}
ProyOrto(u,a)
ProyOrto(c(2,1,-4),c(-5,3,11))
```

*15.*

```
A=matrix(c(1,0,0,1/3,4,0,1/2,3,2),ncol=3,byrow=TRUE)
B=matrix(c(9,0,0,0,1,8,0,0,0,-2,7,0,0,0,-3,6),ncol=4)
solve(A)  # Inversa de A
det(A)
solve(B);det(B)
```

*16.*

```
A=matrix(c(4,4.001,4.001,4.002),ncol=2)
B=A;B[2,2]=4.002001
solve(A)
solve(B)
```

*17.*

```
X=matrix(runif(12),ncol=3)
u=runif(4)
A=t(X)%*%X
B=u%*%t(u)
DA=eigen(A)$values
TA=eigen(A)$vectors
t(TA)%*%TA
prod(DA);det(A)
qr(A)$rank  # Rango de una matriz
sum(diag(DA)!=0)
```

*18.*

```
DB=eigen(B)$values
TB=eigen(B)$vectors
t(TB)%*%TB
prod(DB);det(B)
```

```
qr(B)$rank
sum(diag(DB)!=0)
```

*19.*

```
A=matrix(c(3,2,0,2,3,0,0,0,3),ncol=3)
eig_A=eigen(A)
eig_A$values
eigen(A%*%A)$values
eigen(solve(A))$values
eig_A$vectors%*%diag(eigen(A%*%A)$values)%*%t(eig_A$vectors);A%*%A
```

*20.*

```
A=matrix(c(6,10,1,10,6,5),ncol=2)
ginvMP=function(A){
        res=svd(A)
        res$v%*%diag(1/res$d)%*%t(res$u)}
B=ginvMP(A)
A%*%B%*%A
```

*21.*

```
es.positiva=function(A){
 if (ncol(A)!=nrow(A)) stop("Esto se hace para matrices cuadradas")
 v=eigen(A)$values
 tol=ncol(A)*max(abs(v))* .Machine$double.eps
 if (sum(v>tol)==length(v)) return(TRUE) else return(FALSE)}

A=matrix(c(2,-3/2,-3/2,3),ncol=2);es.positiva(A)
A=matrix(c(1,0.8,0.5,0.8,0.6,0.4,0.5,0.4,0.25),ncol=3);es.positiva(A)
```

*22.*

```
matrixA=function(m){
A=matrix(c(1,-2,-2,m),ncol=2)
return(A)}

dmatrixA=function(m){det(matrixA(m))}

m=seq(-4,10,len=101)
plot(m,mapply(dmatrixA,m=m),type="l") # Dibuja el determinante en funci?n de m
abline(h=0)
A=matrixA(-2)
print(z<-eigen(A))
```

*23.*

```
A=cbind(c(3,1,0),c(1,3,0),c(0,0,3))
B=matrix(0,ncol=3,nrow=3);B[3,3]=2
eigen(A)
eigen(B)

eigen(A+B) # Trampilla
```

**Sesión 1**

## 8.5. Operadores lógicos

```
17<5
17>5
17<=5
17>=5
17!=5
17==5
```

## 8.6. OPERADORES ARITMETICOS

### 8.6.1. SUMA, RESTA, MULTIPLICACION, DIVISION, POTENCIA, MODULO, DIVISION ENTERA

```
17+5
17*5
17*5
17^5
17%/%5
17%%5
```

### 8.6.2. LOGARITMOS Y EXPONENCIALES

```
log(1)
log(12)
log(12,2)
exp(12)
exp(1)
```

### 8.6.3. FUNCIONES TRIGONOMETRICAS

```
sin(45)
cos(45)
```

```
tan(45)
asin(0.96)
acos(0.97)
atan(0.45)
```

### 8.6.4. FUNCIONES VARIAS

```
abs(-34)
sqrt(8)
floor(1.56)
ceiling(1.56)
max(4,7,2,12)
min(4,7,2,12)
sign(-45)
```

### 8.6.5. EJERCICIOS DE PRACTICA

```
# calcular la expresion cos(pi/6+pi/2)+e^2
# calcular la expresion cos(pi/6+pi/2)+e^2*log(5)+arc cos(1/raiz(2))
# introducir las siguientes expresiones:
# a) 1/7
# b) options(digits=3); 1/7
# c) options(digits=6); 1/7
# d) round(67.45)
# e) round(75.324568,2)
# f) options(digits=7);
# g) signif(56.345458234234,2)
# h) signif(56.345458234234)
# i) exp(-30)
# j) options(scipen= 999)
# k) exp(-30)
# l) options(scipen=0)
```

**Sesión 2**

## 8.7. EJERCICIOS DE PRACTICA

### 8.7.1. DEFINICION DE CONSTANTES

```
e = exp(1);
x = 0.0034
e <- exp(1)
x <- 0.034;
x0 = e^(2*x)
```

### 8.7.2. CONCATENAR Y PEGAR EXPRESIONES

```
txt = "El valor de x0 es _"
cat(txt, x0)
paste(txt,x0)
paste0(txt,x0)
```

### 8.7.3. ASIGNACION E IMPRESION

```
x0 <- 1
x1 <- x0 - pi*x0 + 1
(x1 <- x0 - pi*x0 + 1 )
print(x1)
```

### 8.7.4. LISTADO DE OBJETOS DEFINIDOS

```
ls()
# Eliminar todos los objetos
rm(list= ls())
ls()
```

### 8.7.5. IMPRIMIR PEGAR AVANZADO

```
x0 <- 1
x1 <- x0 - pi*x0 + 1
cat("x0 =", x0, "\n","x1 =", x1)
```

### 8.7.6. EJERCICIOS DE PRACTICA

**Sesión 3**

### 8.7.7. DEFINICION DE FUNCIONES

```
# nombre_funcion <- function(param1,param2,param3,...,paramn){
# instruccion 1
# instruccion 2
# return(valor_de_retorno)
#}
```

### 8.7.8. Ejemplo 1

```
fun1 <- function(x,a,b,h,k){
  res <- a+b*cos(hx+k)
  return(res)
}
```

### 8.7.9. Ejemplo 2

```
Discriminante <- function(a,b,c){
  res <- b^2-4*a*c
  return(res)
}
```

### 8.7.10. GRAFICAS

```
fun2 <- function(x,h,k){
  res <- 1/h*sin(k*x)
  return(res)
}

f2 <- fun2(1:100,2,3)
plot(f2,type="l", col= "red", lwd=2,
     main= "Grafico de la funcion f2",
     xlab= "x",
     ylab="f(x)=1/h*sin(k*x)",
     axes= TRUE)
```

### 8.7.11. EJEMPLOS DE PRACTICA

Graficar: rectas, parabolas, cubicas, polinomios, exponenciales, logaritmos

## 8.8. Introducción a Factorización LU

### 8.8.1. Inversa de una matriz

```
‘‘‘{r}
A=matrix(c(1,3,-2,1,1,4,-1,2,0,1,-1,4,2,6,1,2),nrow = 4,byrow = TRUE)
(A)
Aorig<- A
‘‘‘
```

Paso 1)

```
‘‘‘{r}
I = diag(4);(I)
‘‘‘
```

Paso 2)

```{r}
AInv <- cbind(A,I); (AInv)
A <- AInv
```

Paso 3)

```{r}
l21 <- A[2,1]/A[1,1];(l21)
l31 <- A[3,1]/A[1,1];(l31)
l41 <- A[4,1]/A[1,1];(l41)
```

Paso 4)

```{r}
A[2,] <- A[2,]-l21*A[1,];
A[3,] <- A[3,]-l31*A[1,];
A[4,] <- A[4,]-l41*A[1,];
(A)
```

Paso 5)

```{r}
Atmp <- A[2,]
A[2,] <- A[3,]
A[3,] <- Atmp
(A)
```

Paso 6)

```{r}
l32 <- A[3,2]/A[2,2];(l32)
l42 <- A[4,2]/A[2,2];(l42)
```

Paso 7)

```{r}
A[3,] <- A[3,]-l32*A[2,];
A[4,] <- A[4,]-l42*A[2,];
(A)
```

Paso 8)

```{r}
esc <- 1/A[3,3]
A[3,] <- esc*A[3,]
(A)
```

Paso 9)

```{r}
l43 <- A[4,3]/A[3,3];(l43)
```

Paso 10)

```{r}
A[4,] <- A[4,]-l43*A[3,];
(A)
```

Paso 11)

```{r}
esc <- 1/A[4,4]
A[4,] <- esc*A[4,]
(A)
```

Paso 12)

```{r}
l34 <- A[3,4]/A[4,4];(l34); A[3,] <- A[3,]-l34*A[4,]
l24 <- A[2,4]/A[4,4];(l24); A[2,] <- A[2,]-l24*A[4,]
l14 <- A[1,4]/A[4,4];(l14); A[1,] <- A[1,]-l14*A[4,]
(A)
```

Paso 13)

```{r}
l23 <- A[2,3]/A[3,3];(l23); A[2,] <- A[2,]-l23*A[3,]
l13 <- A[1,3]/A[3,3];(l13); A[1,] <- A[1,]-l13*A[3,]
(A)

```

Paso 14)

```{r}
l12 <- A[1,2]/A[2,2];(l12); A[1,] <- A[1,]-l12*A[2,]
(A)
```

Paso 15)

```{r}
AInv <- A[,5:8]; (AInv)
```

Paso 16)

```{r}
IdCalc <- Aorig%*%AInv; (IdCalc)
```

### 8.8.2. Factorización LU

**Ejemplo 74.** ```{r}
A=matrix(c(1,1,1,1,2,3,1,5,-1,1,-5,3,3,1,7,-2),byrow = TRUE,nrow = 4); (A)
```

```{r}
b=matrix(c(10,31,-2,18),nrow = 4);(b)
```

```{r}
Ab <- cbind(A,b); (Ab)
```

*Los pivotes se definen como* $a_{kk}^{(k)}$*, luego construimos los multiplicadores:* $l_{i,k} = a_{ik}^{(k)}/a_{kk}^{(k)}$, $l_{21} = a_{21}/a_{aa}$ *y* $l_{31} = a_{31}/a_{11}$ *y* $l_{41} = a_{41}/a_{11}$

```{r}
A <- Ab;
l21 <- A[2,1]/A[1,1];(l21)
l31 <- A[3,1]/A[1,1];(l31)
l41 <- A[4,1]/A[1,1];(l41)
```

*Construimos la matriz* $U$*, donde las entradas* $a_{ij}^{(k+1)} = a_{ik}^{(k)} - l_{ik} \times a_{kj}^{(k)}$
```

```{r}
A[2,] <- A[2,]-l21*A[1,]; (A[2,])
A[3,] <- A[3,]-l31*A[1,]; (A[3,])
A[4,] <- A[4,]-l41*A[1,]; (A[4,])
```

*Es decir la matriz resultante es:*

```{r}
(A)
```

*entonces el pivote es $A_{22} = 1$, calculemos los $l_{32}$ y $l_{42}$*

```{r}
l32 <- A[3,2]/A[2,2];(l32)
l42 <- A[4,2]/A[2,2];(l42)
```

*Haciendo cero debajo del pivote*

```{r}
A[3,] <- A[3,]-l32*A[2,];(A[3,])
A[4,] <- A[4,]-l42*A[2,];(A[4,])
```

*La matriz A queda de la forma:*

```{r}
(A)
```

*por tanto el pivote es $A_{33} = -2$, y resta calcylar $l_{43}$*

```{r}
l43 <- A[4,3]/A[3,3];(l43)
```

*haciendo cero debajo del pivote*

```{r}
A[4,] <- A[4,]-l43*A[3,];(A[4,])
```

*Por lo tanto la matriz A resultante es*

```{r}
(A)
```

*entonces podemos construir la matriz L con los valores $l_{ij}$ calculados en los pasos anteriores*

```{r}
L=diag(4);(L)
```

```{r}
L[2,1] <- l21
L[3,1] <- l31
L[4,1] <- l41
L[3,2] <- l32
L[4,2] <- l42
L[4,3] <- l43
(L)
```

```{r}
U <- A[,1:4];(U)
```

*Verifiquemos que efectivamente $A = LU$*

```{r}
Acalculada <- L%*%U; (Acalculada)
```

**Ejemplo 75.** *Apliquemos lo desarrollado para la siguiente matriz A*

```{r}
A=matrix(c(4,0,1,1,3,1,3,1,0,1,2,0,3,2,4,1),nrow = 4,byrow = TRUE); (A)
```

```{r}
b=matrix(c(5,6,13,1),nrow = 4);(b)
```

*Paso 1)*

```{r}
Ab <- cbind(A,b)
A <- Ab;
```

*Paso 2)*

```{r}
l21 <- A[2,1]/A[1,1];(l21)
l31 <- A[3,1]/A[1,1];(l31)
l41 <- A[4,1]/A[1,1];(l41)
```

*Paso 3)*

```{r}
A[2,] <- A[2,]-l21*A[1,]; (A[2,])
A[3,] <- A[3,]-l31*A[1,]; (A[3,])
A[4,] <- A[4,]-l41*A[1,]; (A[4,])
```

*Paso 4)*

```{r}
l32 <- A[3,2]/A[2,2];(l32)
l42 <- A[4,2]/A[2,2];(l42)
```

*Paso 5)*

```{r}
A[3,] <- A[3,]-l32*A[2,];(A[3,])
A[4,] <- A[4,]-l42*A[2,];(A[4,])
```

*Paso 6)*

```{r}
l43 <- A[4,3]/A[3,3];(l43)
```

*Paso 7)*

```{r}
A[4,] <- A[4,]-l43*A[3,];(A[4,])
```

*Paso 8)*

```{r}
L=diag(4);(L)
```

*Paso 9)*

```{r}
L[2,1] <- l21
L[3,1] <- l31
L[4,1] <- l41
L[3,2] <- l32
L[4,2] <- l42
L[4,3] <- l43;
(L)
```

*Paso 10)*

```{r}
U <- A[,1:4];(U)
```

*Paso 11)*

```{r}
Acalculada <- L%*%U; (Acalculada)
```

**Ejercicio 35.** *Aplicar la factorización LU a la matriz A definida por*

```{r}
A=matrix(c(2,3,2,4,
           4,10,-4,0,
           -3,-2,-5,-2,
           -2,4,4,-7),nrow = 4,byrow = TRUE)
(A)
```

### 8.8.3. Notas importantes

**Nota 33.** *Si se fija $l_{ii} = 1$ se le denomina factorizacion Doolitle.*

**Nota 34.** ***Nota 2** Una matriz A tiene factorización de Doolitle si y sólo sí se le puede aplicar el metodo de eliminación de Gauss sin pivoteo.*

**Nota 35.** *Si $U = L^T$ entonces la factorización se le denomina factorización de Cholesky.*

**Nota 36.** *Si la matriz es simétrica y definida positiva, entonces $A = LL^T$.*

# 8.9. Métodos de Eliminación directa

## 8.9.1. Gaussiana Simple

```
```{r,echo=FALSE}
gauss_simple <- function(A, b, tolerancia, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  if (!is.numeric(b)) stop("b debe ser numérico.")
  n <- nrow(A)
  if (ncol(A) != n) stop("A debe ser cuadrada.")
  if (length(b) != n) stop("Longitud de b debe ser igual al número de filas de A.")
  Ab <- cbind(A, b);
  numcols = ncol(Ab)
  for (k in 1:(n - 1)) {
    pivote <- Ab[k, k]
    if (abs(pivote) < tolerancia) {
      stop(sprintf("Pivote casi cero en fila %d (%.3e).
      El método sin pivoteo falla.", k, pivote))
    }
    if (k + 1 <= n) {
      for (i in (k + 1):n) {
        m <- Ab[i, k] / pivote;
        Ab[i, k:numcols] <- Ab[i, k:numcols] - m * Ab[k, k:numcols]
        if (abs(Ab[i, k]) < tolerancia) Ab[i, k] <- 0
        if (verbose) {
        cat(sprintf("k=%d, i=%d, m=%.6g\n", k, i, m));
        print(Ab)}
      }
    }
  }
  x <- numeric(n)
  for (i in n:1) {
    if (i == n) {
      suma <- 0
    } else {suma <- sum(Ab[i, (i + 1):n] * x[(i + 1):n])}
    x[i] <- (Ab[i, n + 1] - suma) / Ab[i, i]
  }
  list(x=x,U=Ab[, 1:n],Ab=Ab)
}
```
```

**Ejemplo 76.**

```
```{r,echo=FALSE}
A <- matrix(c(
  2,  1, -1,
```
```

```
 -3, -1,  2,
 -2,  1,  2
), nrow = 3, byrow = TRUE)
b <- c(8, -11, -3);
tolerancia <- 1e-12;
res <- gauss_simple(A, b,tolerancia);
res <- gauss_simple(A, b,tolerancia, verbose = TRUE)
res$x;res$U;res$Ab;A %*% res$x
```

### 8.9.2. Gaussiana con pivoteo parcial

```{r}
gauss_piv_parcial <- function(A, b, tolerancia, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A);
  if (ncol(A) != n) stop("A debe ser cuadrada.")
  if (length(b) != n) stop("Dimensiones de b incorrectas.")
  Ab <- cbind(A, b)
  for (k in 1:(n-1)) {
    # Selección del pivote (máximo en valor absoluto en la
    #columna k desde la fila k)
    max_row <- which.max(abs(Ab[k:n, k])) + (k - 1)
    if (abs(Ab[max_row, k]) < tolerancia){
    stop(sprintf("Pivote casi nulo en columna %d", k))}
    # Intercambio de filas si es necesario
    if (max_row != k) {
      Ab[c(k, max_row), ] <- Ab[c(max_row, k), ]
      if (verbose) {
      cat(sprintf("Intercambio de fila %d con fila %d\n", k, max_row));
      print(Ab)}
    }
    for (i in (k + 1):n) {
      m <- Ab[i, k] / Ab[k, k];
      Ab[i, k:ncol(Ab)] <- Ab[i, k:ncol(Ab)] - m * Ab[k, k:ncol(Ab)]
      if (abs(Ab[i, k]) < tolerancia) Ab[i, k] <- 0
      if (verbose) {
      cat(sprintf("k=%d, i=%d, m=%.6g\n", k, i, m));print(Ab)}
    }
  }
  x <- numeric(n)
  for (i in n:1) {
    if (i == n) {
      suma <- 0
```

```
    } else {
    suma <- sum(Ab[i, (i + 1):n] * x[(i + 1):n])
    }
    x[i] <- (Ab[i, n + 1] - suma) / Ab[i, i]
  }
  list(x = x,U = Ab[, 1:n],Ab = Ab)
}
```

**Ejemplo 77.**

```
```{r,echo=FALSE}
A <- matrix(c(
  0, 2, 1,
  1, -2, -3,
  -1, 1, 2), nrow = 3, byrow = TRUE)
b <- c(3, -3, -1);
tolerancia <- 1e-12
res <- gauss_piv_parcial(A, b, tolerancia, verbose = TRUE);
res$x
```
```

### 8.9.3. Gauss Jordan

```
```{r}
gauss_jordan <- function(A, b, tolerancia, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A);
  m <- ncol(A)
  if (!is.null(b) && length(b) != n)
  stop("Dimensiones incompatibles entre A y b.")
  Ab <- cbind(A, as.matrix(b));
   ncols <- ncol(Ab); row <- 1
  for (col in 1:m) {
    if (row > n) break
    pivot_row_rel <- which.max(abs(Ab[row:n, col]))
    pivot_row <- row + pivot_row_rel - 1
    if (abs(Ab[pivot_row, col]) < tolerancia) {
      if (verbose) cat(sprintf("Sin pivote usable en col=%d (|piv <tol). Se omite c
      next
    }
    if (pivot_row != row) {
      Ab[c(row, pivot_row), ] <- Ab[c(pivot_row, row), ]
      if (verbose) {
      cat(sprintf("Swap filas %d <-> %d (col=%d)\n", row,
      pivot_row, col));
      print(Ab)}
```
```

```
    }
    pivote <- Ab[row, col];
    Ab[row, ] <- Ab[row, ] / pivote
    if (verbose) {
    cat(sprintf("Normaliza fila %d por pivote %.6g (col=%d)\n",
    row, pivote, col));
    print(Ab)}
    for (r in 1:n) {
      if (r == row) next
      factor <- Ab[r, col]
      if (abs(factor) > tolerancia) {
        Ab[r, ] <- Ab[r, ] - factor * Ab[row, ]
        if (verbose) {
        cat(sprintf("R%d := R%d - (%.6g)*R%d (col=%d)\n",
        r, r, factor, row, col));print(Ab)}
      }
    }
    row <- row + 1
  }
  Ab[abs(Ab) < tolerancia] <- 0;
  out <- list(RREF = Ab);
  X <- Ab[, (m + 1):ncols, drop = FALSE]
  out$x <- if (ncol(X) == 1) as.vector(X) else X
  return(out)
}
```

**Ejemplo 78.** ```{r,echo=FALSE}

```
A <- matrix(c(
  2,  1, -1,
 -3, -1,  2,
 -2,  1,  2), nrow = 3, byrow = TRUE)
b <- c(8, -11, -3);
tolerancia <- 1e-12;
res <- gauss_jordan(A, b, tolerancia, verbose = TRUE);
res$x
```

### 8.9.4. Cálculo de Inversa

```{r}
gauss_jordan_inversa <- function(A,tolerancia, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A); m <- ncol(A)
  if (n != m) stop("Para invertir, A debe ser cuadrada.")
  Ab <- cbind(A, diag(n));
```

```
ncols <- ncol(Ab);
row <- 1
for (col in 1:m) {
  if (row > n) break
  pivot_row_rel <- which.max(abs(Ab[row:n, col]))
  pivot_row <- row + pivot_row_rel - 1
  if (abs(Ab[pivot_row, col]) < tolerancia) {
    stop(sprintf("No hay pivote en la columna %d (|piv|<tol).",
    col))
  }
  if (pivot_row != row) {
    Ab[c(row, pivot_row), ] <- Ab[c(pivot_row, row), ]
    if (verbose) {
    cat(sprintf("Intercambia filas %d <-> %d (col=%d)\n",
    row, pivot_row, col));
    print(Ab)
    }
  }
  pivote <- Ab[row, col];
  Ab[row, ] <- Ab[row, ] / pivote
  if (verbose) {
  cat(sprintf("Normaliza fila %d por pivote %.6g (col=%d)\n",
  row, pivote, col));
  print(Ab)}
  for (r in 1:n) {
    if (r == row) next
    factor <- Ab[r, col]
    if (abs(factor) > tolerancia) {
      Ab[r, ] <- Ab[r, ] - factor * Ab[row, ]
      if (verbose){
      cat(sprintf("R%d := R%d - (%.6g)*R%d (col=%d)\n",
      r, r, factor, row, col));
      print(Ab)
      }
    }
  }
  row <- row + 1
}
Ab[abs(Ab) < tolerancia] <- 0;
LHS <- Ab[, 1:m, drop = FALSE]
if (!all(LHS == diag(n))) {
stop("La matriz es singular o la tolerancia es muy estricta.")
}
Ainv <- Ab[, (m + 1):ncols, drop = FALSE]
list(Ainv = Ainv, RREF = Ab)
```

```
}
```
```

**Ejemplo 79.** ```{r,echo=FALSE}
```
tolerancia <- 1e-12
A <- matrix(c(
  1, 2, 3,
  0, 1, 4,
  5, 6, 0), nrow = 3, byrow = TRUE)
res <- gauss_jordan_inversa(A, tolerancia,
 verbose = TRUE);
 res$Ainv;
 round(A %*% res$Ainv, 6)
```
```

### 8.9.5. Factorización LU

#### Sin Pivoteo

```{r}
lu_simple <- function(A, tol = 1e-12, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A); m <- ncol(A)
  if (n != m) stop("A debe ser cuadrada.")
  U <- A;
  L <- diag(n)
  for (k in 1:(n - 1)) {
    piv <- U[k, k]
    if (abs(piv) < tol) {
    stop(sprintf("Pivote casi cero en k=%d. No se puede continuar.",
    k))
    }
    for (i in (k + 1):n) {
      m <- U[i, k] / piv;
      L[i, k] <- m;
      U[i, k:n] <- U[i, k:n] - m * U[k, k:n]
      if (verbose) {
      cat(sprintf("k=%d, i=%d, m=%.6g\n", k, i, m));
      print(U)
      }
    }
  }
  list(L = L, U = U)
}
```

**Ejemplo 80.** ```{r,echo=FALSE}

```
A <- matrix(c(
  2,  1, -1,
 -3, -1,  2,
 -2,  1,  2
), 3, 3, byrow = TRUE)
lu <- lu_simple(A);lu$L; lu$U
```

### Con Pivoteo

```{r}
lu_piv_parcial <- function(A, tol = 1e-12, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A); m <- ncol(A);
  if (n != m) stop("A debe ser cuadrada.")
  U <- A;
  L <- diag(n);
  P <- diag(n)
  for (k in 1:(n - 1)) {
    max_row <- which.max(abs(U[k:n, k])) + (k - 1)
    if (abs(U[max_row, k]) < tol) {
    stop(sprintf("Matriz singular (pivote ~ 0).", k))
    }
    if (max_row != k) {
      U[c(k, max_row), ] <- U[c(max_row, k), ];
      P[c(k, max_row), ] <- P[c(max_row, k), ]
      if (k > 1) {
      L[c(k, max_row), 1:(k - 1)] <- L[c(max_row, k), 1:(k - 1)]
      }
      if (verbose) {
      cat(sprintf("Intercambia filas %d <-> %d\n",
      k, max_row));
      print(U)
      }
    }
    for (i in (k + 1):n) {
      m <- U[i, k] / U[k, k];
      L[i, k] <- m;
      U[i, k:n] <- U[i, k:n] - m * U[k, k:n];
      if (verbose) {
      cat(sprintf("k=%d, i=%d, m=%.6g\n", k, i, m));
      print(U)
      }
    }
  }
```

```
  list(P = P, L = L, U = U)
}
```

**Ejemplo 81.** ```{r,echo=FALSE}

```
A <- matrix(c(
  0,  2,  1,
  1, -2, -3,
 -1,  1,  2), 3, 3, byrow = TRUE)
lu <- lu_piv_parcial(A, verbose = FALSE);
lu$P;
lu$L;
lu$U
```

### Solucion vía LU

```{r}
forward_sub <- function(L, b) {
  n <- nrow(L);
  y <- numeric(n)
  for (i in 1:n) {
  y[i] <- (b[i] - sum(L[i, 1:(i - 1)] * y[1:(i - 1)]))
  }
  y
}
```

```{r}
# x en Ux = y
back_sub <- function(U, y, tol = 1e-12) {
  n <- nrow(U);
  x <- numeric(n)
  for (i in n:1) {
    s <- if (i == n) 0 else sum(U[i, (i + 1):n] * x[(i + 1):n])
    if (abs(U[i, i]) < tol) stop(sprintf("Pivote ~0 en U[%d,%d].",
    i, i))
    x[i] <- (y - s) / U[i, i]
  }
  x
}
```

### Resolucion sin pivoteo

```{r}
```

```
solve_lu_simple <- function(A, b, tol = 1e-12) {
  lu <- lu_simple(A, tol = tol);
  y <- forward_sub(lu$L, b)
  x <- back_sub(lu$U, y, tol = tol)
  x
}
```

### 8.9.6. Resolucion con pivoteo

```{r}
solve_lu_piv_parcial <- function(A, b, tol = 1e-12) {
  lu <- lu_piv_parcial(A, tol = tol);
  Pb <- lu$P %*% b
  y  <- forward_sub(lu$L, as.vector(Pb))
  x  <- back_sub(lu$U, y, tol = tol)
  x
}
```

**Ejemplo 82.** ```{r,echo=FALSE}

```
A <- matrix(c(
  0,  2,  1,
  1, -2, -3,
 -1,  1,  2), 3, 3, byrow = TRUE)
b <- c(3, -3, -1);
x <- solve_lu_piv_parcial(A, b);
x
```

### 8.9.7. Factorización de Cholesky

```{r}
tolerancia <- 1e-12
cholesky_fact <- function(A,tolerancia) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A)
  if (ncol(A) != n) stop("A debe ser cuadrada.")
  if (!all(abs(A - t(A)) < tolerancia)) stop("A debe ser simétrica.")
  L <- matrix(0, n, n)
  for (j in 1:n) {
    suma <- sum(L[j, 1:(j-1)]^2)
    val <- A[j, j] - suma
    if (val <= 0) stop("A no es definida positiva (falló Cholesky).")
    L[j, j] <- sqrt(val)
    if (j < n) {
```

```
      for (i in (j+1):n) {
        suma <- sum(L[i, 1:(j-1)] * L[j, 1:(j-1)])
        L[i, j] <- (A[i, j] - suma) / L[j, j]
      }
    }
    cat(sprintf("Paso j=%d\n", j))
    print(L)
  }
  return(L)
}
```

**Ejemplo 83.**

```
# Matriz simétrica definida positiva
A <- matrix(c(
  4,  12, -16,
 12,  37, -43,
-16, -43,  98), nrow = 3, byrow = TRUE)

L <- cholesky_fact(A,tolerancia)
L
```

### 8.9.8. Solución vía Cholesky

```{r}
tolerancia <- 1e-12
solve_cholesky <- function(A, b, tolerancia) {
  L <- cholesky_fact(A, tolerancia)
  y <- forward_sub(L, b)
  x <- back_sub(t(L), y, tolerancia)
  x
}
```

**Ejemplo 84.**

```{r,echo=FALSE}
b <- c(1, 2, 3)
x <- solve_cholesky(A, b,tolerancia)
x
A %*% x
```

## 8.10. Métodos Iterativos

### 8.10.1. Gauss Jacobi

```{r}
jacobi <- function(A, b, x0 = NULL, tol = 1e-8, maxiter = 1000) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A)
  if (ncol(A) != n) stop("A debe ser cuadrada.")
  if (length(b) != n) stop("Dimensiones de b incompatibles.")
  if (is.null(x0)) x0 <- rep(0, n)
  x <- x0;
  D <- diag(diag(A));
  R <- A - D;
  for (k in 1:maxiter) {
    x_new <- (b - R %*% x) / diag(D)
    cat(sprintf("Iter %d: %s\n", k,
    paste(round(x_new, 6), collapse = " ")))
    if (sqrt(sum((x_new - x)^2)) < tol) {
    return(list(x = as.vector(x_new),
    iter = k, convergencia = TRUE))}
    x <- x_new
  }
  list(x = as.vector(x),
  iter = maxiter,
  convergencia = FALSE)
}
```

**Ejemplo 85.**

```{r,echo=FALSE}
A <- matrix(c(
  4, -1,  0,
 -1,  4, -1,
  0, -1,  3), 3, 3, byrow = TRUE)

b <- c(15, 10, 10)
res_jacobi <- jacobi(A, b, x0 = c(0, 0, 0), tol = 1e-8);
res_jacobi$x
```

### 8.10.2. Gauss Seidel

```{r}
gauss_seidel <- function(A, b, x0 = NULL,
```

```
tol = 1e-8, maxiter = 1000, verbose = FALSE) {
  if (!is.matrix(A)) stop("A debe ser una matriz.")
  n <- nrow(A)
  if (ncol(A) != n) stop("A debe ser cuadrada.")
  if (length(b) != n) stop("Dimensiones de b incompatibles.")
  x <- if (is.null(x0)) rep(0, n) else as.numeric(x0)
  if (length(x) != n) stop("Dimensión de x0 incompatible con A.")
  for (k in 1:maxiter) {
    x_old <- x
    for (i in 1:n) {
      aii <- A[i, i]
      if (abs(aii) < .Machine$double.eps) {
      stop(sprintf("A[%d,%d] = 0: Gauss{Seidel No se puede aplicar",
       i, i))}
      s1 <- if (i > 1) sum(A[i, 1:(i - 1)] * x[1:(i - 1)]) else 0
      s2 <- if (i < n) sum(A[i, (i + 1):n] * x_old[(i + 1):n]) else 0
      x[i] <- (b[i] - s1 - s2) / aii
    }
    dx  <- sqrt(sum((x - x_old)^2));
    res <- sqrt(sum((b - A %*% x)^2))
    cat(sprintf("Iter %4d |dx|=%.3e  |res|=%.3e   x=%s\n",
    k, dx, res, paste(round(x, 6), collapse=" ")))
    if (dx < tol && res < tol) {
      return(list(x = as.vector(x), iter = k,
      convergencia = TRUE,delta = dx, residuo = res))
    }
  }
  list(x = as.vector(x), iter = maxiter, convergencia = FALSE,
       delta = sqrt(sum((x - x)^2)),
       residuo = sqrt(sum((b - A %*% x)^2)))
}
```

**Ejemplo 86.**

```{r,echo=FALSE}
A <- matrix(c(
  4, -1,  0,
 -1,  4, -1,
  0, -1,  3
), 3, 3, byrow = TRUE)
b <- c(15, 10, 10)
res_gs <- gauss_seidel(A, b, x0 = c(0,0,0), tol = 1e-8);
res_gs$x
```

# 8.11. Clases

## 8.11.1. Métodos Iterativos

## 8.11.2. Gauss-Jacobi

$$A = \begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 6 \\ 25 \\ -11 \\ 15 \end{pmatrix}, \qquad \mathbf{x} = (1, 2, -1, 1)^T.$$

Iteración

$$\begin{aligned} x_1^{(k+1)} &= \frac{6 + x_2^{(k)} - 2x_3^{(k)}}{10}, \\ x_2^{(k+1)} &= \frac{25 + x_1^{(k)} + x_3^{(k)} - 3x_4^{(k)}}{11}, \\ x_3^{(k+1)} &= \frac{-11 - 2x_1^{(k)} + x_2^{(k)} + x_4^{(k)}}{10}, \\ x_4^{(k+1)} &= \frac{15 - 3x_2^{(k)} + x_3^{(k)}}{8}. \end{aligned}$$

Tomamos $\mathbf{x}^{(0)} = (0, 0, 0, 0)^\top$ y actualizamos cada $x_i^{(k+1)}$ usa sólo valores del paso $k$.

**Iteración 1:**

$$\begin{aligned} x_1^{(1)} &= \frac{6 + 0 - 20}{10} = 0.6, \\ x_2^{(1)} &= \frac{25 + 0 + 0 - 3 \cdot 0}{11} = \frac{25}{11} \approx 2.272727, \\ x_3^{(1)} &= \frac{-11 - 2 \cdot 0 + 0 + 0}{10} = -1.1, \\ x_4^{(1)} &= \frac{15 - 3 \cdot 0 + 0}{8} = \frac{15}{8} = 1.875. \end{aligned}$$

**Iteración 2** Con $\mathbf{x}^{(1)} = (0.6,\ 2.272727,\ -1.1,\ 1.875)$,

$$\begin{aligned} x_1^{(2)} &= \frac{6 + 2.272727 - 2(-1.1)}{10} = \frac{10.472727}{10} = 1.047273, \\ x_2^{(2)} &= \frac{25 + 0.6 + (-1.1) - 3(1.875)}{11} = \frac{18.875}{11} \approx 1.715909, \\ x_3^{(2)} &= \frac{-11 - 2(0.6) + 2.272727 + 1.875}{10} = \frac{-8.052273}{10} \approx -0.805227, \\ x_4^{(2)} &= \frac{15 - 3(2.272727) + (-1.1)}{8} = \frac{7.081819}{8} \approx 0.885227. \end{aligned}$$

**Iteración 3** Con $\mathbf{x}^{(2)} = (1.047273,\ 1.715909,\ -0.805227,\ 0.885227)$,

$$\begin{aligned}
x_1^{(3)} &= \frac{6 + 1.715909 - 2(-0.805227)}{10} = \frac{9.326363}{10} = 0.932636,\\
x_2^{(3)} &= \frac{25 + 1.047273 + (-0.805227) - 3(0.885227)}{11} = \frac{22.586365}{11} = 2.053306,\\
x_3^{(3)} &= \frac{-11 - 2(1.047273) + 1.715909 + 0.885227}{10} = \frac{-10.493410}{10} = -1.049341,\\
x_4^{(3)} &= \frac{15 - 3(1.715909) + (-0.805227)}{8} = \frac{9.047046}{8} = 1.130881.
\end{aligned}$$

**Iteración 4** Con $\mathbf{x}^{(3)} = (0.932636,\ 2.053306,\ -1.049341,\ 1.130881)$,

$$\begin{aligned}
x_1^{(4)} &= \frac{6 + 2.053306 - 2(-1.049341)}{10} = \frac{10.151988}{10} = 1.015199,\\
x_2^{(4)} &= \frac{25 + 0.932636 + (-1.049341) - 3(1.130881)}{11} = \frac{21.490652}{11} = 1.953696,\\
x_3^{(4)} &= \frac{-11 - 2(0.932636) + 2.053306 + 1.130881}{10} = \frac{-9.681085}{10} = -0.968109,\\
x_4^{(4)} &= \frac{15 - 3(2.053306) + (-1.049341)}{8} = \frac{7.790741}{8} = 0.973843.
\end{aligned}$$

**Iteración 5** Con $\mathbf{x}^{(4)} = (1.015199,\ 1.953696,\ -0.968109,\ 0.973843)$,

$$\begin{aligned}
x_1^{(5)} &= \frac{6 + 1.953696 - 2(-0.968109)}{10} = \frac{9.889914}{10} = 0.988991,\\
x_2^{(5)} &= \frac{25 + 1.015199 + (-0.968109) - 3(0.973843)}{11} = \frac{22.125561}{11} = 2.011415,\\
x_3^{(5)} &= \frac{-11 - 2(1.015199) + 1.953696 + 0.973843}{10} = \frac{-10.102859}{10} = -1.010286,\\
x_4^{(5)} &= \frac{15 - 3(1.953696) + (-0.968109)}{8} = \frac{8.170803}{8} = 1.021351.
\end{aligned}$$

Tabla de resultados

| k | $x_1^{(k)}$ | $x_2^{(k)}$ | $x_3^{(k)}$ | $x_4^{(k)}$ |
|---|---|---|---|---|
| 0 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 1 | 0.6000 | 2.2727 | -1.1000 | 1.8750 |
| 2 | 1.0473 | 1.7159 | -0.8052 | 0.8852 |
| 3 | 0.9326 | 2.0533 | -1.0493 | 1.1309 |
| 4 | 1.0152 | 1.9537 | -0.9681 | 0.9738 |
| 5 | 0.9890 | 2.0114 | -1.0103 | 1.0214 |

Tabla de errores

| k | $\|x^{(k)} - x^*\|_\infty$ | $\|x^{(k)} - x^{(k-1)}\|_\infty$ |
|---|---|---|
| 0 | 2.0000 | – |
| 1 | 0.8750 | 2.2727 |
| 2 | 0.2841 | 0.9898 |
| 3 | 0.1309 | 0.3374 |
| 4 | 0.0463 | 0.1570 |
| 5 | 0.0214 | 0.0577 |

Ejercicios:instrucciones: usa $x^{(0)} = \mathbf{0}$ y detén cuando $\|x^{(k+1)} - x^{(k)}\|_\infty < 10^{-3}$. Comprueba dominancia diagonal, escribe las fórmulas de Jacobi e itera con tabla.

**Ejercicio 36.**

$$\begin{pmatrix} 5 & 1 \\ 2 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -10 \end{pmatrix}.$$

**Ejercicio 37.**

$$\underbrace{\begin{pmatrix} 9 & -1 & 0 & 0 \\ -1 & 12 & -2 & 1 \\ 0 & -1 & 11 & -2 \\ 0 & 2 & -1 & 10 \end{pmatrix}}_{A_{4\times 4}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 7 \\ 26 \\ -15 \\ 15 \end{pmatrix}}_{b},$$

**Ejercicio 38.**

$$\underbrace{\begin{pmatrix} 10 & -1 & 2 & 0 & 0 \\ -1 & 12 & -2 & 3 & 0 \\ 2 & -1 & 11 & -2 & 1 \\ 0 & 3 & -1 & 10 & -2 \\ 0 & 0 & 2 & -1 & 9 \end{pmatrix}}_{A_{5\times 5}} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 15 \\ -17 \\ 26 \\ -7 \\ 13 \end{pmatrix}}_{b},$$

## 8.11.3. Gauss-Seidel

Considerar

$$A = \begin{pmatrix} 9 & -1 & 0 & 2 \\ -1 & 10 & -2 & 1 \\ 0 & -1 & 8 & -1 \\ 2 & 1 & -1 & 11 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 10 \\ 8 \\ 6 \\ 13 \end{pmatrix}, \quad \mathbf{x}^* = (1,1,1,1)^\top.$$

$A$ es diagonalmente dominante por renglones, por lo que Gauss–Seidel converge.

Iteración general Con ecuaciones por renglón:

$$\begin{aligned} 9x_1 - x_2 + 2x_4 = 10, &\quad \Rightarrow x_1 = \frac{10 + x_2 - 2x_4}{9}, \\ -x_1 + 10x_2 - 2x_3 + x_4 = 8, &\quad \Rightarrow x_2 = \frac{8 + x_1 + 2x_3 - x_4}{10}, \\ -x_2 + 8x_3 - x_4 = 6, &\quad \Rightarrow x_3 = \frac{6 + x_2 + x_4}{8}, \\ 2x_1 + x_2 - x_3 + 11x_4 = 13, &\quad \Rightarrow x_4 = \frac{13 - 2x_1 - x_2 + x_3}{11}. \end{aligned}$$

**Regla de GS:** cada $x_i^{(k+1)}$ usa los *valores más recientes disponibles* dentro del mismo paso $k+1$.

Partimos de $\mathbf{x}^{(0)} = (0,0,0,0)^\top$.

**Iteración 1**

$$x_1^{(1)} = \frac{10+0-20}{9} = 1.111111,$$
$$x_2^{(1)} = \frac{8+\underline{x_1^{(1)}}+20-0}{10} = \frac{8+1.111111}{10} = 0.911111,$$
$$x_3^{(1)} = \frac{6+\underline{x_2^{(1)}}+0}{8} = \frac{6+0.911111}{8} = 0.863889,$$
$$x_4^{(1)} = \frac{13-2\underline{x_1^{(1)}}-\underline{x_2^{(1)}}+\underline{x_3^{(1)}}}{11} = \frac{13-2(1.111111)-0.911111+0.863889}{11} = 0.975505.$$

**Iteración 2**

$$x_1^{(2)} = \frac{10+x_2^{(1)}-2x_4^{(1)}}{9} = \frac{10+0.911111-2(0.975505)}{9} = 0.995567,$$
$$x_2^{(2)} = \frac{8+\underline{x_1^{(2)}}+2x_3^{(1)}-x_4^{(1)}}{10} = \frac{8+0.995567+2(0.863889)-0.975505}{10} = 0.974784,$$
$$x_3^{(2)} = \frac{6+\underline{x_2^{(2)}}+x_4^{(1)}}{8} = \frac{6+0.974784+0.975505}{8} = 0.993786,$$
$$x_4^{(2)} = \frac{13-2\underline{x_1^{(2)}}-\underline{x_2^{(2)}}+\underline{x_3^{(2)}}}{11} = \frac{13-2(0.995567)-0.974784+0.993786}{11} = 1.002534.$$

**Iteración 3**

$$x_1^{(3)} = \frac{10+0.974784-2(1.002534)}{9} = 0.996635,$$
$$x_2^{(3)} = \frac{8+\underline{0.996635}+2(0.993786)-1.002534}{10} = 0.998167,$$
$$x_3^{(3)} = \frac{6+\underline{0.998167}+1.002534}{8} = 1.000088,$$
$$x_4^{(3)} = \frac{13-2\underline{0.996635}-\underline{0.998167}+\underline{1.000088}}{11} = 1.000786.$$

**Iteración 4**

$$x_1^{(4)} = \frac{10+0.998167-2(1.000786)}{9} = 0.999622,$$
$$x_2^{(4)} = \frac{8+\underline{0.999622}+2(1.000088)-1.000786}{10} = 0.999901,$$
$$x_3^{(4)} = \frac{6+\underline{0.999901}+1.000786}{8} = 1.000086,$$
$$x_4^{(4)} = \frac{13-2\underline{0.999622}-\underline{0.999901}+\underline{1.000086}}{11} = 1.000086.$$

**Iteración 5**

$$x_1^{(5)} = \frac{10 + 0.999901 - 2(1.000086)}{9} = 0.999970,$$
$$x_2^{(5)} = \frac{8 + \underline{0.999970} + 2(1.000086) - 1.000086}{10} = 1.000006,$$
$$x_3^{(5)} = \frac{6 + \underline{1.000006} + 1.000086}{8} = 1.000011,$$
$$x_4^{(5)} = \frac{13 - 2\underline{0.999970} - \underline{1.000006} + \underline{1.000011}}{11} = 1.000006.$$

Tabla de resultados

| k | $x_1^{(k)}$ | $x_2^{(k)}$ | $x_3^{(k)}$ | $x_4^{(k)}$ |
|---|---|---|---|---|
| 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 1 | 1.111111 | 0.911111 | 0.863889 | 0.975505 |
| 2 | 0.995567 | 0.974784 | 0.993786 | 1.002534 |
| 3 | 0.996635 | 0.998167 | 1.000088 | 1.000786 |
| 4 | 0.999622 | 0.999901 | 1.000086 | 1.000086 |
| 5 | 0.999970 | 1.000006 | 1.000011 | 1.000006 |

Tabla de errores

| k | $\|x^{(k)} - x^*\|_\infty$ | $\|x^{(k)} - x^{(k-1)}\|_\infty$ |
|---|---|---|
| 0 | 1.000000 | – |
| 1 | 0.136111 | 1.111111 |
| 2 | 0.025216 | 0.129897 |
| 3 | 0.003365 | 0.023383 |
| 4 | 0.000378 | 0.002986 |
| 5 | 0.000030 | 0.000348 |

Instrucciones: usa $x^{(0)} = \mathbf{0}$ y detén cuando $\|x^{(k+1)} - x^{(k)}\|_\infty < 10^{-3}$. Escribe las fórmulas de GS e itera con tabla.

**Ejercicio 39.**

$$\underbrace{\begin{pmatrix} 6 & -1 \\ 1 & 5 \end{pmatrix}}_{A_{2\times 2}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \underbrace{\begin{pmatrix} 13 \\ -3 \end{pmatrix}}_{b}.$$

**Ejercicio 40.**

$$\underbrace{\begin{pmatrix} 8 & 1 & -1 & 0 \\ 2 & 10 & -2 & 1 \\ 1 & -1 & 9 & -1 \\ 0 & 2 & -1 & 7 \end{pmatrix}}_{A_{4\times 4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \underbrace{\begin{pmatrix} 6 \\ -3 \\ 20 \\ -9 \end{pmatrix}}_{b}.$$

**Ejercicio 41.**

$$\underbrace{\begin{pmatrix} 10 & -1 & 0 & 0 & 2 \\ -1 & 11 & -1 & 0 & 0 \\ 0 & -1 & 12 & -2 & 1 \\ 0 & 0 & -2 & 9 & -1 \\ 2 & 0 & 1 & -1 & 8 \end{pmatrix}}_{A_{5\times 5}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \underbrace{\begin{pmatrix} 10 \\ 21 \\ 1 \\ -10 \\ 11 \end{pmatrix}}_{b}.$$