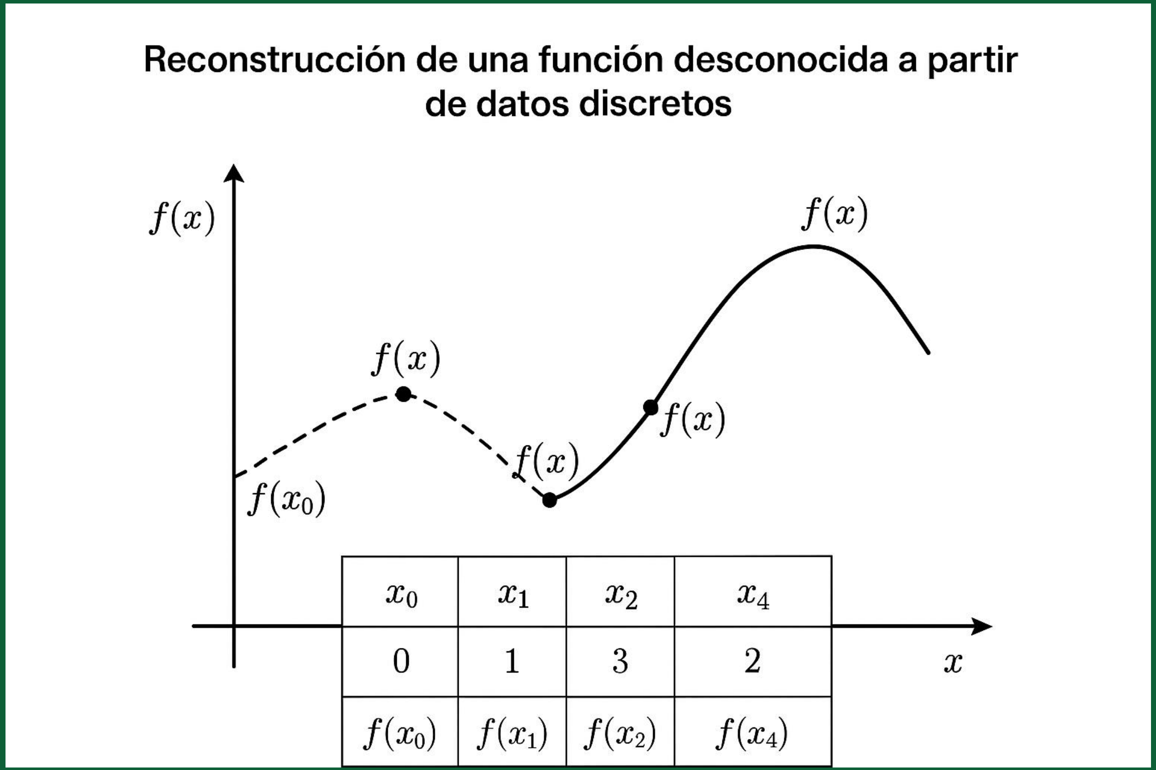
Reconstrucción de funciones a partir de datos discretos

en muchos problemas científicos y de ingeniería no conocemos la función exacta que describe un fenómeno, sino únicamente muestras discretas de sus valores en ciertos puntos. el objetivo es reconstruir o aproximar esa función para poder:

- ✓ evaluarla en puntos intermedios (interpolación).
- ✓ aproximar sus variaciones instantáneas (derivadas).
- ✓ analizar su comportamiento global.

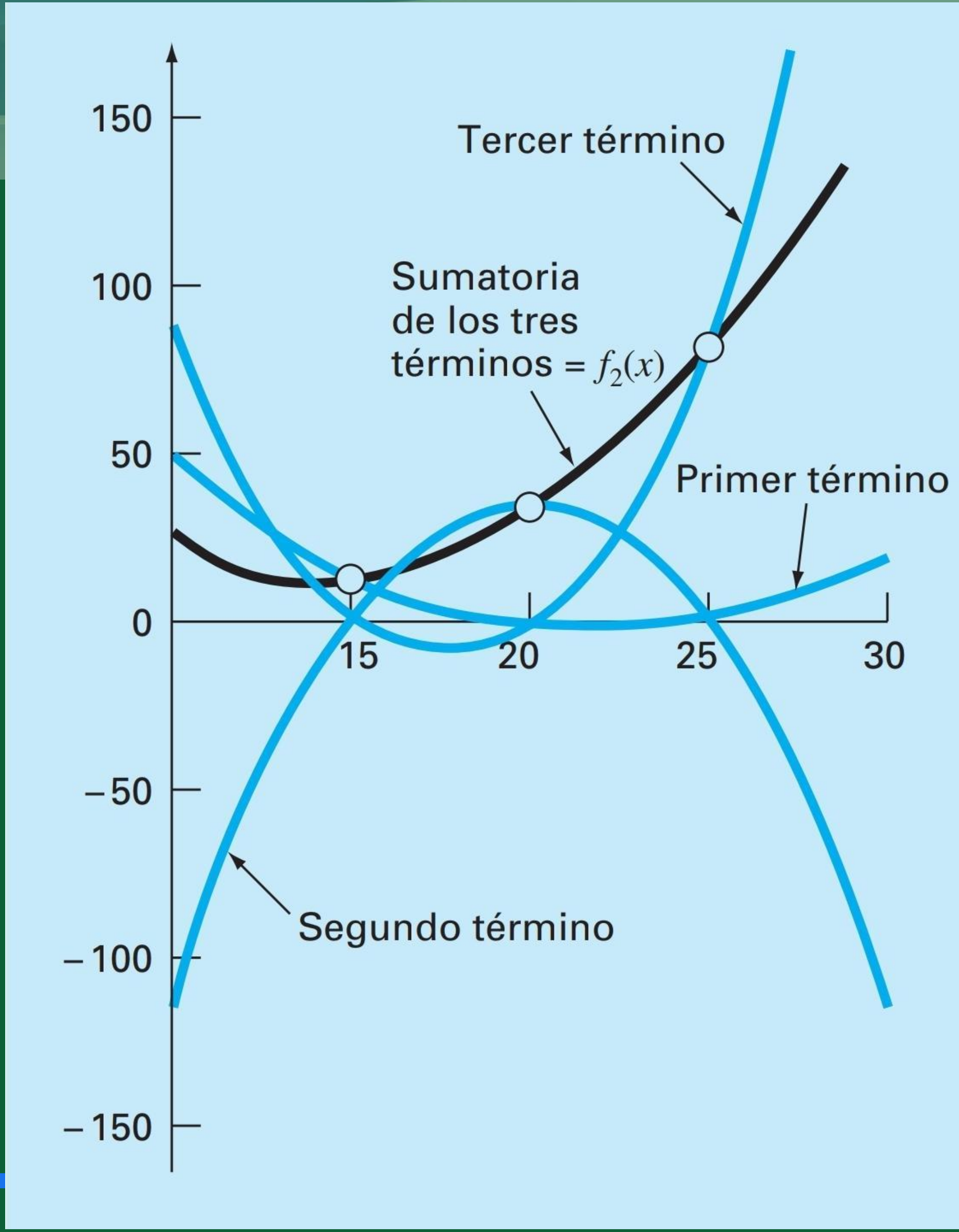


Polinomio Interpolante de Lagrange

El polinomio interpolante de Lagrange es una técnica matemática utilizada para aproximar funciones basándose en un conjunto de puntos dados. este método construye un polinomio que pasa exactamente por esos puntos.

$$P_{(x)} = \sum_{i=0}^{n} y_i L_{i(x)}$$

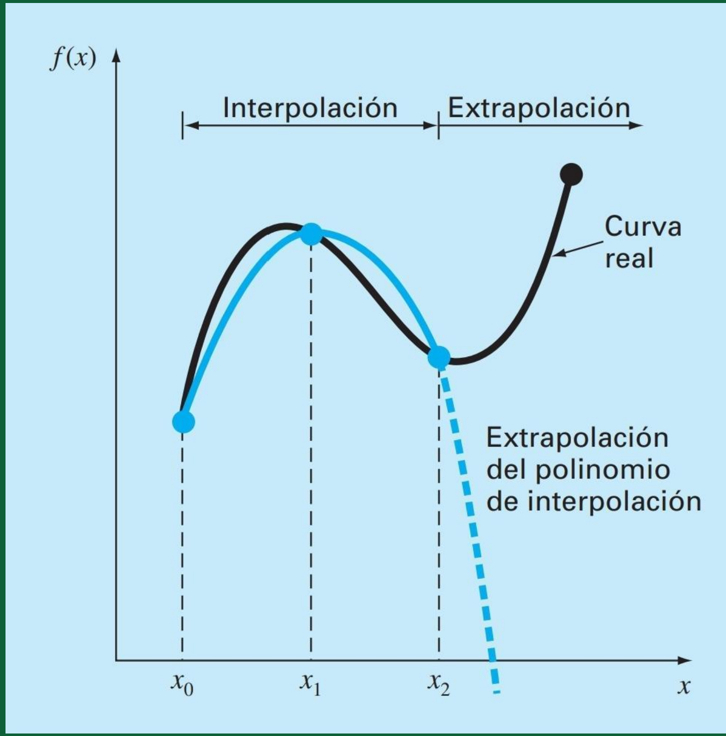
$$L_{i(x)} = \prod_{\substack{j=0 \\ i\neq j}}^{n} \frac{x - x_j}{x_i - x_j}$$



Existencia y unicidad

Siempre es posible construir un polinomio de interpolación de Lagrange para un conjunto de puntos distintos (x_i, y_i), como $P_{(x)}$.

$$P_{(x)} = \sum_{i=0}^{n} y_i L_{i(x)}$$



Unicidad

Dado un conjunto $(n+1)$ puntos distintos (x_i, y_i), existe un polinomio **grado n** que pasa exactamente por todos esos puntos. Esto se garantiza por el teorema fundamental de la interpolación.

Errores locales y Globales

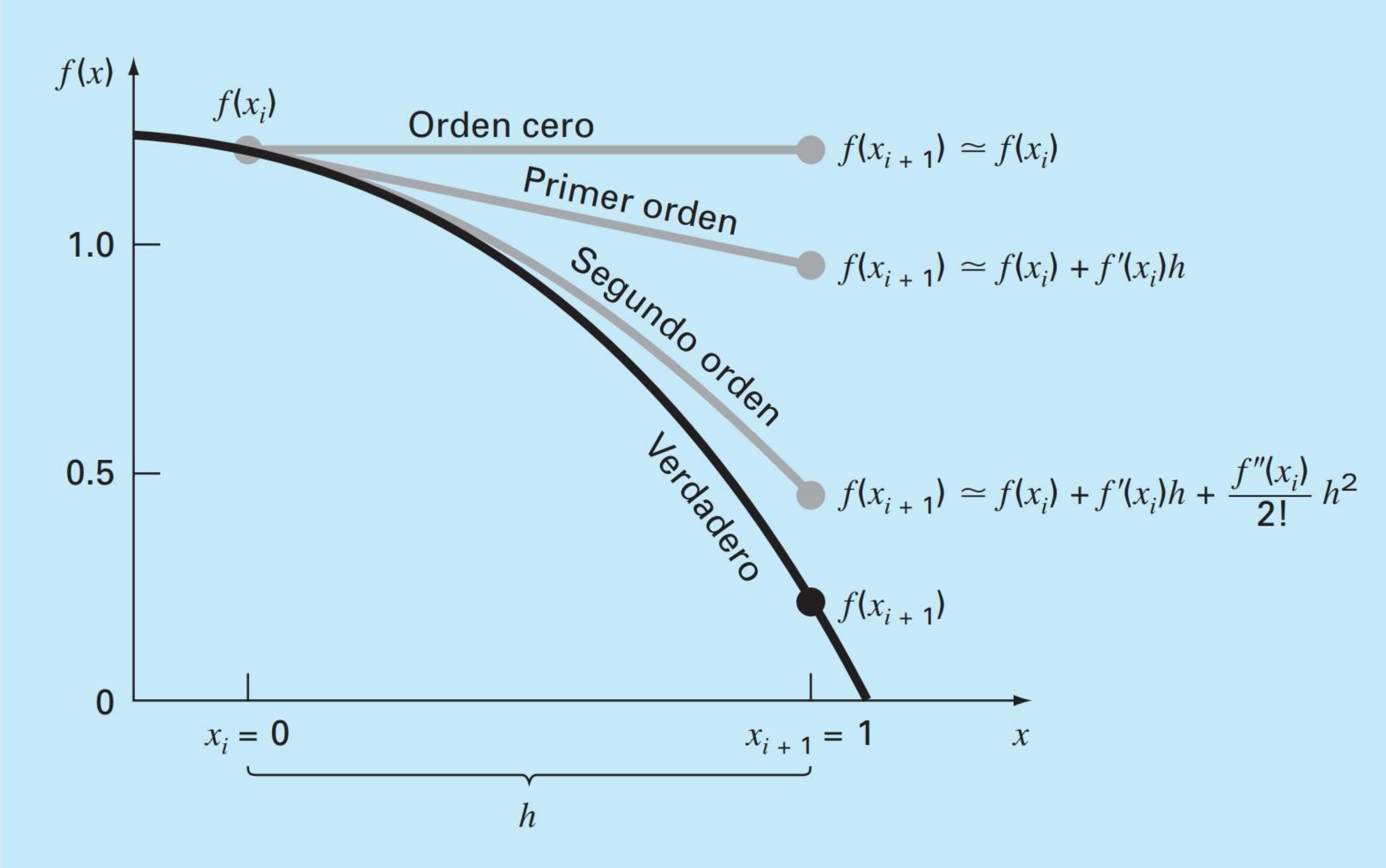
El error de interpolación: el error entre $f_{(x)}$ y el polinomio $P_{(x)}$,

donde $x_0 < \xi < x_n$, es un número en el intervalos de los puntos.

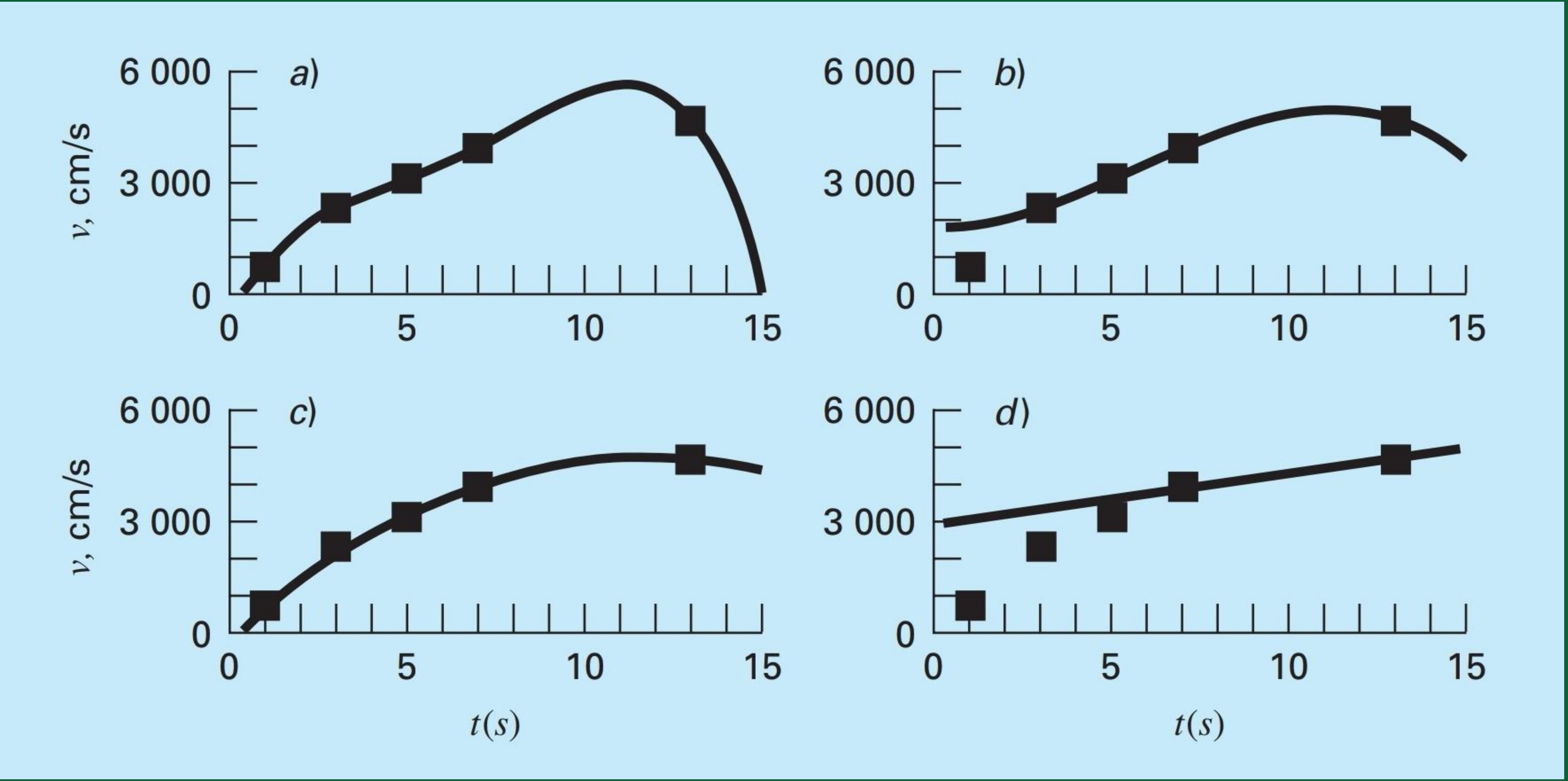
$$f_{(x)} - P_{(x)} = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

Aproximación de Taylor

la serie de Taylor es una herramienta matemática que permite aproximar funciones mediante una suma infinita de términos polinómicos. cada término se calcula utilizando las derivadas de la función en un punto específico.



Polinomio Interpolante distintos grados



Calculo de errores

1. Construir un polinomio interpolante de Lagrange.

$$P_{(x)} = \sum_{i=0}^{n} y_i L_{i(x)}$$

2. Calcular el termino del producto

$$\prod_{i=0}^{n} (x - x_i)$$

3. Determinar el valor máximo en la derivada

$$M_{n+1} = max\left|f^{(n+1)}(\xi)\right|$$

4. Aplicar la cota

$$\left|E_{(x)}\right| \leq \frac{M_{n+1}}{(n+1)!}\left|max \prod_{i=0}^{n} (x - x_i)\right|$$

5. Verificar con el valor real (error local)

$$\left|E_{(x)}\right| = \left|f_{(x)} - P_{(x)}\right|$$

```
import numpy as np
import matplotlib.pyplot as plt

def polinomio_lagrange(x, x_puntos, y_puntos):
    n = len(x_puntos)
    L = 0
    for i in range(n):
        # Calcular el polinomio base de Lagrange
        li = 1
        for j in range(n):
            if i != j:
                li *= (x - x_puntos[j]) / (x_puntos[i] - x_puntos[j])
        L += y_puntos[i] * li
    return L

def reconstruccion_lagrange(x_puntos, y_puntos):
    n = len(x_puntos)
    coeficientes = np.zeros(n)
    for i in range(n):
        # Calcular el polinomio base de Lagrange
        li = np.poly1d([1])
        for j in range(n):
            if i != j:
                li *= np.poly1d([1, -x_puntos[j]]) / (x_puntos[i] - x_puntos[j])
        coeficientes += y_puntos[i] * li.coefficients
    return coeficientes
```

```
# Definir los puntos dados
x_puntos = np.array([0,1,2, 3,4])
y_puntos = np.array([1,2,0, 2,3])
# Crear un conjunto de valores x para graficar
x = np.linspace(min(x_puntos) - 1, max(x_puntos) + 1, 400)
y = [polinomio_lagrange(xi, x_puntos, y_puntos) for xi in x]

# Graficar los puntos dados
plt.scatter(x_puntos, y_puntos, color='red', label='Puntos dados')

# Graficar el polinomio de Lagrange
plt.plot(x, y, label='Polinomio de Lagrange')

# Añadir leyenda y mostrar la gráfica
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Interpolación de Lagrange')
plt.grid(True)
plt.show()
# Reconstruir el polinomio de Lagrange
coeficientes = reconstruccion_lagrange(x_puntos, y_puntos)
polinomio = np.poly1d(coeficientes)
# Imprimir el polinomio reconstruido
print(f"El polinomio de Lagrange es:\n{polinomio}")
```

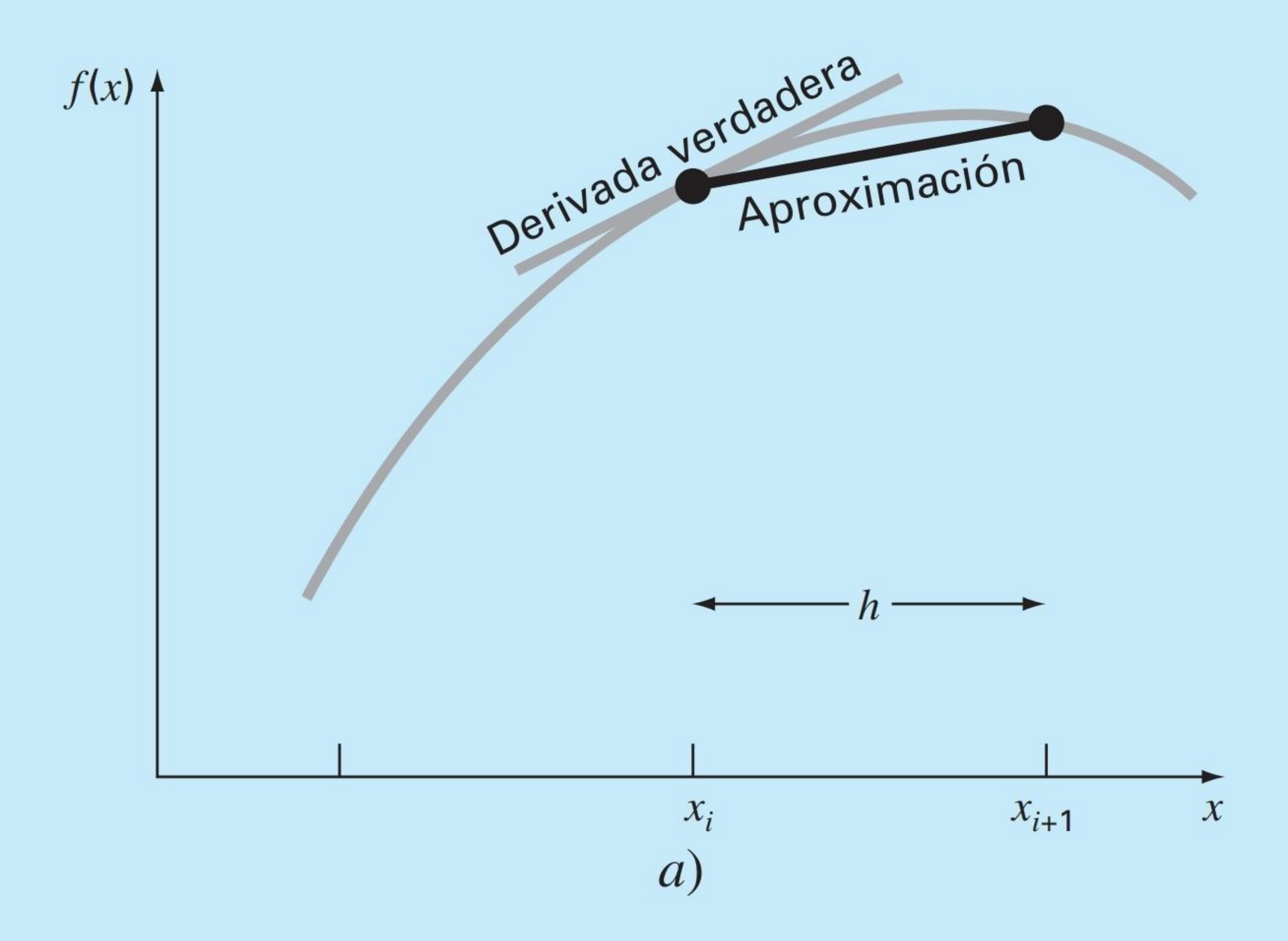
Derivación numérica desde datos discretos

Si solo se conoce un paquete discreto de datos, la derivada debe aproximarse mediante **fórmulas de diferencias finitas**:

Diferencias finitas progresivas

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$$

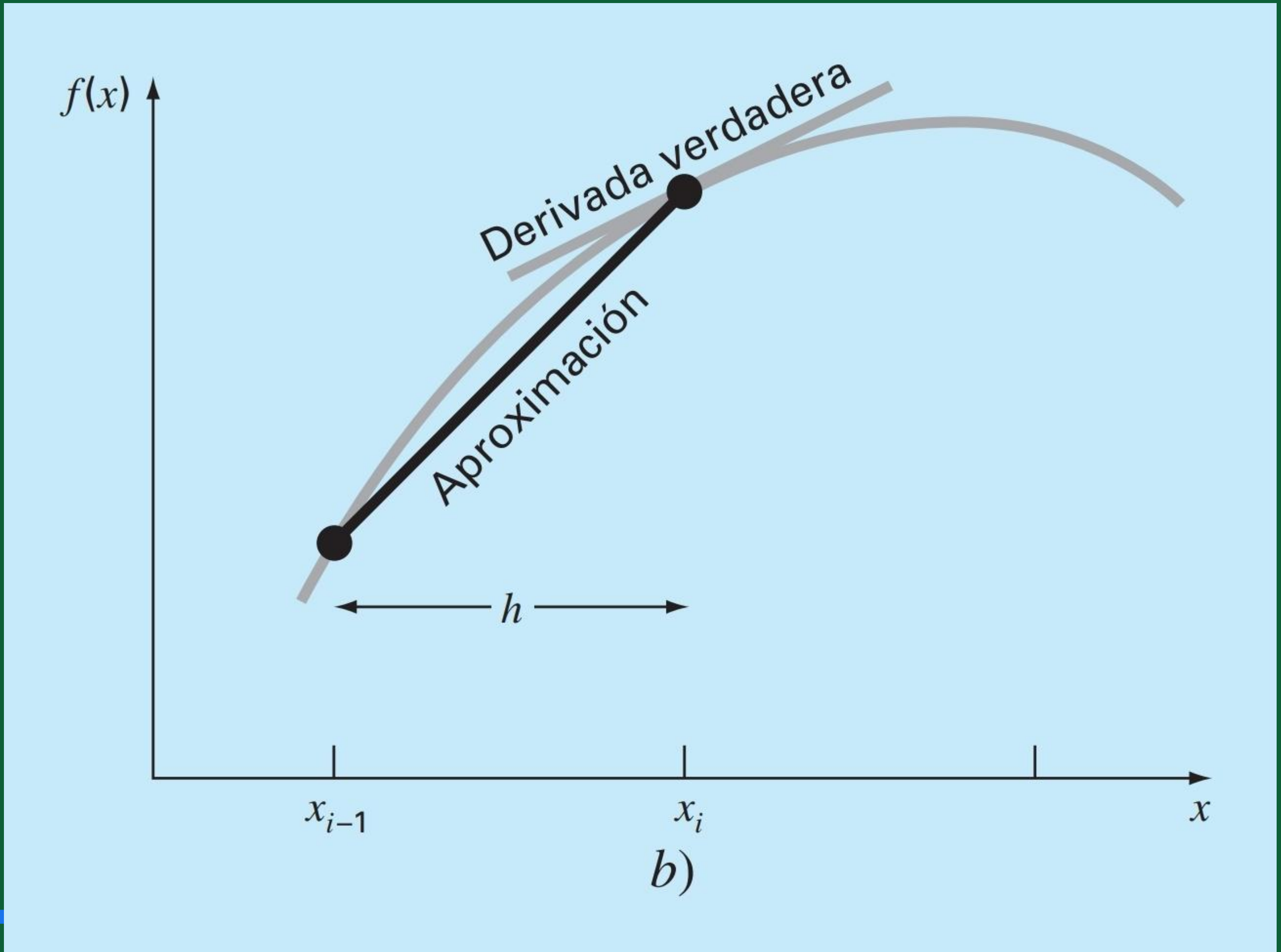
$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_i+1) + f(x_i)}{h^2}$$



Diferencias finitas Regresivas

$$f'_{(x_i)} = \frac{f_{(x_i)} - f_{(x_{i-1})}}{h}$$

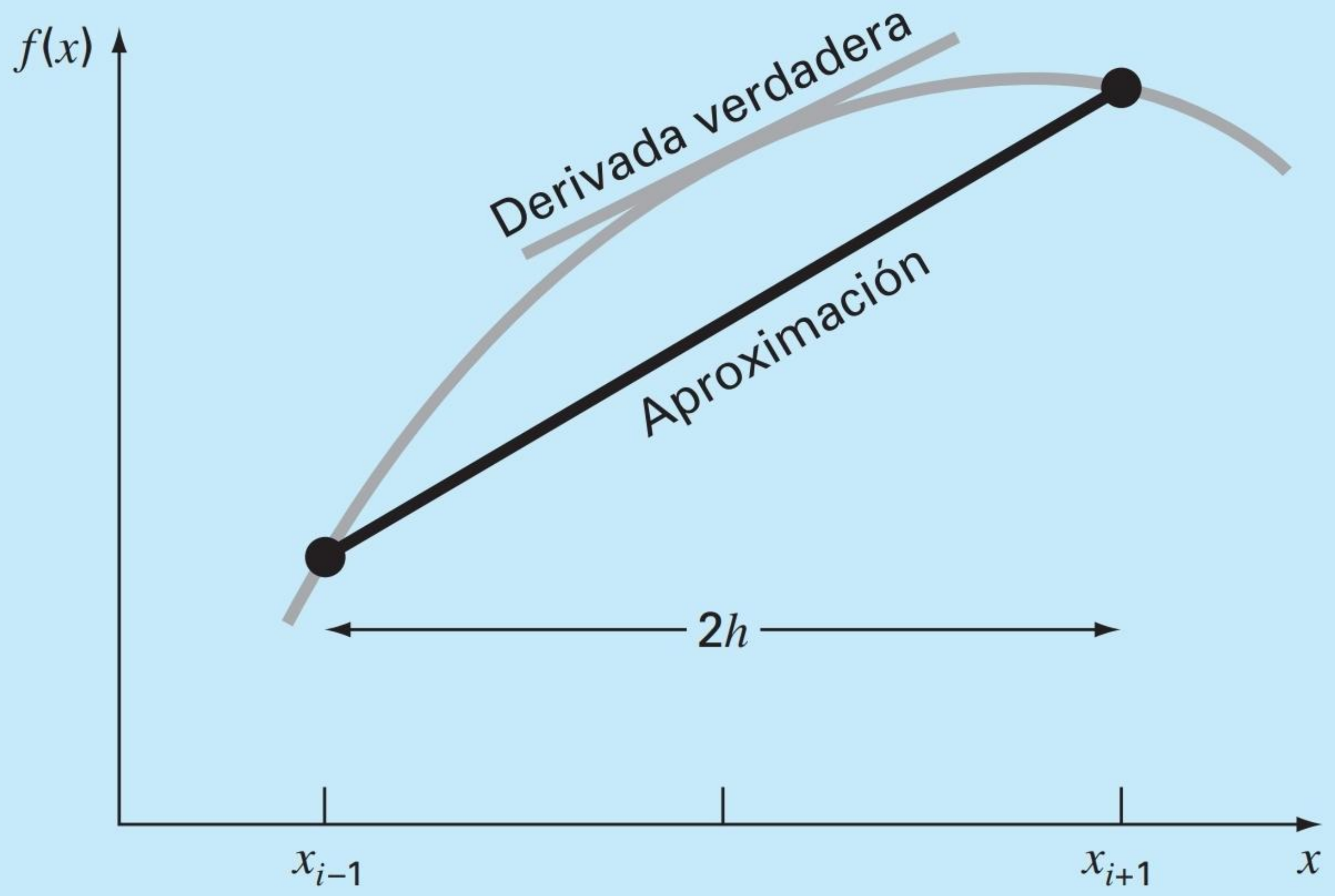
$$f''_{(x_i)} = \frac{f_{(x_i)} - 2f_{(x_{i-1})} + f_{(x_{i-2})}}{h^2}$$



Diferencias finitas Centrales

$$f'_{(x_i)} = \frac{f_{(x_{i+1})} - f_{(x_{i-1})}}{2h}$$

$$f''_{(x_i)} = \frac{f_{(x_{i+1})} - 2f_{(x_i)} + f_{(x_{i-1})}}{h^2}$$



c)

Derivadas parciales numéricas

Si $f(x, y)$ se conoce solo en una malla discreta:
•Derivada parcial respecto a x:

$$\boxed{f'_{x(xi,yi)} = \frac{f_{(x_{i+1}\,,yi)} - f_{(x_{i-1,}\,\mathrm{yi})}}{2hx}}$$

$$\boxed{f'_{y(xi,yi)} = \frac{f_{(x_i\,,yi+1)} - f_{(x_{i,}\,\mathrm{yi-1})}}{2hy}}$$

En resumen, la reconstrucción de funciones desconocidas a partir de datos discretos se aborda con interpolación (polinomios, splines), mientras que las variaciones instantáneas (derivadas) se aproximan con fórmulas de diferencias finitas. Los errores y la elección del paso juegan un rol central en la calidad de los resultados.

Derivadas parciales numéricas (aplicaciones)

1. Modelado en 2D y 3D a partir de datos experimentales

Cuando una función depende de varias variables (ej. $f(x, y)$), las derivadas parciales permiten medir **cómo cambia la función respecto a cada variable**, manteniendo la otra fija.

Ejemplo:

- $f(x, y)$ = temperatura medida en una placa en función de la posición (x, y).
- $fx(x, y)$ indica **gradiente de temperatura en dirección horizontal**.
- $fy(x, y)$ indica **gradiente en dirección vertical**.

Aplicación: **detección de flujos de calor** en ingeniería térmica.

2. Dinámica de fluidos y física computacional

En ecuaciones de la física aparecen derivadas parciales:

- **Ecuación del calor**: $\partial u/\partial t = \alpha \nabla^2 u$
- **Ecuaciones de Navier–Stokes** (fluidos).
- **Ecuación de Schrödinger** en mecánica cuántica.

Cuando no conocemos la solución analítica, se aproximan las derivadas parciales en una malla de puntos → **métodos de diferencias finitas / elementos finitos**.

3. Dinámica de fluidos y física computacional

4. Procesamiento de imágenes

5. Reconstrucción de superficies y campos

```
# Definir la funcion
def f(x):
return x**3 - 2*x + 1
# Punto y paso
x = 2
h = 0.1

# Diferencias finitas centradas para la primera derivada
def primera_derivada(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)

# Diferencias finitas centradas para la segunda derivada
def segunda_derivada(f, x, h):
    return (f(x + h) - 2*f(x) + f(x - h)) / (h**2)

# Calcular las derivadas
primera = primera_derivada(f, x, h)
segunda = segunda_derivada(f, x, h)

# Imprimir los resultados
print(f"Primera derivada en x = {x}: {primera}")
print(f"Segunda derivada en x = {x}: {segunda}")
```