

# Interpolación Lineal: Teoría y Aplicaciones

21 de abril de 2025

## 1. Definición

La interpolación lineal es un método numérico que permite estimar un valor desconocido dentro de un intervalo utilizando una línea recta que conecta dos puntos conocidos. Es el caso más simple de interpolación polinomial, donde se usa un polinomio de grado 1.

## 2. Fundamento Teórico

### 2.1. Fórmula de Interpolación Lineal

Dados dos puntos  $(x_0, y_0)$  y  $(x_1, y_1)$ , la ecuación de la recta que los une es:

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

donde:

- $x$  es el punto donde se desea interpolar.
- $y$  es el valor interpolado.

### 2.2. Condiciones de Aplicabilidad

- La función debe ser continua en el intervalo  $[x_0, x_1]$ .
- La interpolación es exacta en los puntos conocidos.
- El error aumenta si la curvatura de la función real es grande.

## 3. Ejemplo Numérico

### 3.1. Problema

Estimar el valor de  $f(2,5)$  dados los puntos  $(2, 4)$  y  $(3, 9)$ .

### 3.2. Solución

Aplicando la fórmula:

$$y = 4 + \frac{9-4}{3-2}(2,5-2) = 4 + 5 \times 0,5 = 6,5$$

## 4. Implementación en Python

```
1 def interpolacion_lineal(x0, y0, x1, y1, x):
2     """Interpola el valor de y en x usando los puntos (x0,y0) y (x1
3     ,y1)."""
4     return y0 + (y1 - y0) * (x - x0) / (x1 - x0)
5
6 # Ejemplo:
7 x0, y0 = 2, 4
8 x1, y1 = 3, 9
9 x_interp = 2.5
10 y_interp = interpolacion_lineal(x0, y0, x1, y1, x_interp)
11 print(f"Valor interpolado en x={x_interp}: {y_interp}")
```

## 5. Aplicaciones

- Estimación de datos entre mediciones discretas (ej. temperatura, precios de acciones).
- Gráficos por computadora (suavizado de líneas).
- Cálculo numérico en tablas de datos.

## 6. Limitaciones

- Solo es precisa para funciones aproximadamente lineales en el intervalo.
- No captura curvaturas (para ello se usan interpolaciones de mayor grado como cuadrática o cúbica).
- El error puede ser significativo si los puntos están muy separados.

## 7. Ejercicio Propuesto

Dados los puntos (1, 1) y (4, 16):

1. Calcule el valor interpolado en  $x = 2$ .
2. Grafique la función real  $f(x) = x^2$  y la interpolación lineal.
3. Calcule el error absoluto en  $x = 2$ .

# Interpolación Lineal: Teoría, Implementación y Gráficos

Aux. Yhorel Yhared Alvarez Alvarez

## 1 Teoría

La **interpolación lineal** es una técnica utilizada para estimar el valor de una función entre dos puntos conocidos  $(x_0, f(x_0))$  y  $(x_1, f(x_1))$ , asumiendo que la función es aproximadamente lineal entre estos dos puntos. El valor interpolado se calcula usando la ecuación de la recta que conecta los puntos:

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Esta fórmula nos da el valor aproximado de  $f(x)$  en cualquier  $x$  entre  $x_0$  y  $x_1$ .

## 2 Cómo Funciona

La interpolación lineal se basa en la suposición de que entre dos puntos conocidos, los valores intermedios de la función siguen una línea recta. Esto es útil en situaciones donde los datos son aproximados y se requiere una estimación rápida.

## 3 Código en Python

A continuación se presenta el código para implementar la interpolación lineal en Python:

```
# Función de interpolación lineal
def interpolacion_lineal(x0, y0, x1, y1, x):
    # Calcular la pendiente
    pendiente = (y1 - y0) / (x1 - x0)
    # Usar la fórmula de interpolación
    y = y0 + pendiente * (x - x0)
    return y
```

```
# Ejemplo de uso
x0, y0 = 1, 2 # Punto (x0, y0)
x1, y1 = 3, 3 # Punto (x1, y1)
x = 2 # Valor de x para interpolar
resultado = interpolacion_lineal(x0, y0, x1, y1, x)
print(f"El valor interpolado es {resultado}")
```

## 4 Explicación del Código Línea por Línea

- `def interpolacion_lineal(x0, y0, x1, y1, x):`  
Esta línea define una función que recibe cinco parámetros: los dos puntos conocidos  $(x_0, y_0)$  y  $(x_1, y_1)$ , y el valor de  $x$  donde queremos interpolar.
- `pendiente = (y1 - y0) / (x1 - x0)`  
Calcula la pendiente de la línea que pasa por los dos puntos conocidos. La pendiente es crucial porque determina la tasa de cambio entre los puntos.
- `y = y0 + pendiente * (x - x0)`  
Aplica la fórmula de la interpolación lineal para estimar el valor  $f(x)$ . Aquí,  $x_0$  y  $f(x_0)$  son utilizados como referencia, y la pendiente ajusta la diferencia en  $x$ .
- `return y`  
Devuelve el valor interpolado  $f(x)$ , que es el resultado de la función en  $x$ .
- `x0, y0 = 1, 2 # Punto (x0, y0)`  
Define el primer punto  $(x_0, y_0)$  donde  $x_0 = 1$  y  $f(x_0) = 2$ .
- `x1, y1 = 3, 3 # Punto (x1, y1)`  
Define el segundo punto  $(x_1, y_1)$ , donde  $x_1 = 3$  y  $f(x_1) = 3$ .
- `x = 2 # Valor de x para interpolar`  
Establece el valor de  $x = 2$ , el cual es el punto intermedio donde deseamos interpolar.
- `resultado = interpolacion_lineal(x0, y0, x1, y1, x)`  
Llama a la función de interpolación lineal, la cual calculará el valor de  $f(2)$ .
- `print(f"El valor interpolado es {resultado}")`  
Muestra el valor interpolado para  $x = 2$ , que es el resultado de la interpolación.

## 5 Gráfica de la Interpolación Lineal

Finalmente, la visualización de los puntos conocidos y el valor interpolado nos ayuda a entender el proceso de interpolación lineal:

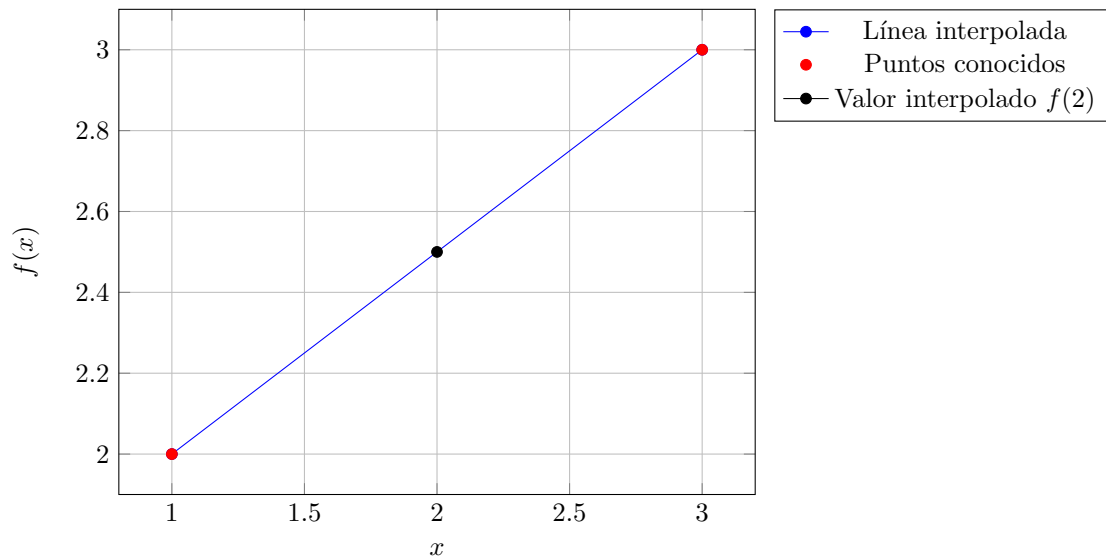


Figure 1: Interpolación Lineal entre dos puntos conocidos.

En esta gráfica, los puntos rojos  $(1, 2)$  y  $(3, 3)$  representan los puntos conocidos. La línea azul es la interpolación lineal entre esos dos puntos, y el punto negro es el valor interpolado en  $x = 2$ .

## 6 Conclusión

La interpolación lineal es una técnica simple y efectiva para estimar valores intermedios entre dos puntos conocidos. Sin embargo, es importante tener en cuenta que solo es precisa si la función subyacente es aproximadamente lineal entre los puntos. En casos donde la función cambia drásticamente, es mejor utilizar interpolación de mayor orden, como la interpolación cuadrática.

# Guía de Programación: Interpolación Lineal

Departamento de Computación Científica

21 de abril de 2025

## 1. Lógica de Programación

La interpolación lineal sigue estos pasos lógicos:

1. **Entradas:** Recibir dos puntos  $(x_0, y_0)$  y  $(x_1, y_1)$ , y el valor  $x$  a interpolar.
2. **Validación:** Verificar que  $x_0 \neq x_1$  para evitar división por cero.
3. **Cálculo de pendiente:** Calcular  $m = \frac{y_1 - y_0}{x_1 - x_0}$ .
4. **Interpolación:** Aplicar  $y = y_0 + m(x - x_0)$ .
5. **Salida:** Retornar el valor interpolado  $y$ .

## 2. Implementación en Python

```
1 def interpolar_lineal(x0, y0, x1, y1, x):
2     """
3     Interpola el valor y correspondiente a x usando los puntos (x0,y0) y (x1,y1).
4
5     Par metros:
6         x0, y0: Coordenadas del primer punto conocido
7         x1, y1: Coordenadas del segundo punto conocido
8         x: Valor donde se desea interpolar
9
10    Retorna:
11        Valor y interpolado
12    """
13    # Validación de datos de entrada
14    if x0 == x1:
15        raise ValueError("Los valores x0 y x1 no pueden ser iguales (división por cero)")
16
17    # Cálculo de la pendiente (m)
18    pendiente = (y1 - y0) / (x1 - x0)
19
20    # Aplicación de la fórmula de interpolación
21    y = y0 + pendiente * (x - x0)
22
23    return y
```

Listing 1: Implementación básica

## 3. Explicación del Código

### Validación de Entrada

- `if x0 == x1:` Comprueba que no haya división por cero.
- `raise ValueError:` Genera un error explícito si los puntos tienen igual  $x$ .

## Cálculo de Pendiente

- $(y_1 - y_0) / (x_1 - x_0)$ : Implementa la fórmula matemática  $m = \frac{\Delta y}{\Delta x}$ .

## Fórmula de Interpolación

- $y_0 + \text{pendiente} * (x - x_0)$ : Aplica  $y = y_0 + m(x - x_0)$  punto-pendiente.

## 4. Ejemplo de Uso

```
1 # Puntos conocidos
2 punto1 = (2, 4) # (x0, y0)
3 punto2 = (5, 10) # (x1, y1)
4
5 # Valor a interpolar
6 x_interpoliar = 3.5
7
8 # Llamada a la función
9 y_interpolado = interpolar_lineal(*punto1, *punto2, x_interpoliar)
10
11 print(f"Para x = {x_interpoliar}, y {y_interpolado:.2f}")
```

Listing 2: Ejemplo práctico

## Salida Esperada

Para x = 3.5, y 7.00

## 5. Consideraciones Importantes

- **Precisión:** La interpolación lineal solo es exacta para funciones lineales.
- **Extrapolación:** No debe usarse para valores fuera del intervalo  $[x_0, x_1]$ .
- **Error:** El error aumenta con la curvatura de la función real.

# Laboratorio: Interpolación Lineal con Python

Departamento de Computación Científica

21 de abril de 2025

## Instrucciones

- Implemente las soluciones en Python usando la función de interpolación lineal provista
- Documente adecuadamente su código con comentarios
- Presente resultados con 4 decimales

### 1. Ejercicio 1: Termodinámica

Se midió la viscosidad ( $\mu$ ) de un aceite a dos temperaturas diferentes:

T (°C)	$\mu$ (Pa·s)
20	0.0157
50	0.0083

1. Implemente una función de interpolación lineal
2. Calcule la viscosidad a 35°C
3. Determine la temperatura para  $\mu = 0,0100$  Pa·s

### 2. Ejercicio 2: Finanzas

Un acción tiene los siguientes precios históricos:

Día	Precio (USD)
5	28.50
15	31.20

1. Interpole el precio en el día 10
2. Grafique los puntos y la recta interpolante (usando matplotlib)
3. Calcule el error si el precio real en día 10 fue \$29.80



**NOMBRE:** MAMANI MORALES IVAR MIJAIL  
**C.I.:** 8362307

## Laboratorio: Interpolación Lineal con Python

### Ejercicio 1: Termodinámica

Se midió la viscosidad ( $\mu$ ) de un aceite a dos temperaturas diferentes:

T (°C)	$\mu$ (Pa·s)
20	0.0157
50	0.0083

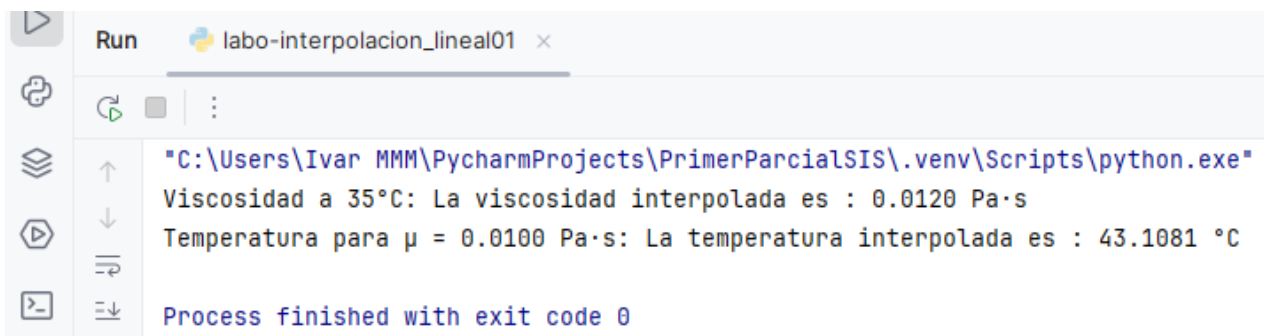
1. Implemente una función de interpolación lineal
2. Calcule la viscosidad a 35°C
3. Determine la temperatura para  $\mu = 0,0100$  Pa·s

```
# Ejercicio 1: Termodinámica
def interpolacion_lineal(x0, y0, x1, y1, x):
    y = y0 + (y1 - y0) / (x1 - x0) * (x - x0)
    return y

# Datos
T0 = 20
mu0 = 0.0157
T1 = 50
mu1 = 0.0083

# Calcular la viscosidad a 35°C
T_interp = 35
mu_interp = interpolacion_lineal(T0, mu0, T1, mu1, T_interp)
print(f"Viscosidad a 35°C: La viscosidad interpolada es : {mu_interp:.4f} Pa·s")

# Calcular la temperatura para mu = 0.0100 Pa·s
mu_target = 0.0100
T_target = interpolacion_lineal(mu0, T0, mu1, T1, mu_target)
print(f"Temperatura para  $\mu = 0.0100$  Pa·s: La temperatura interpolada es : {T_target:.4f} °C")
```



```
Run labo-interpolacion_lineal01 x
"C:\Users\Ivar MMM\PycharmProjects\PrimerParcialSIS\.venv\Scripts\python.exe"
Viscosidad a 35°C: La viscosidad interpolada es : 0.0120 Pa·s
Temperatura para  $\mu = 0.0100$  Pa·s: La temperatura interpolada es : 43.1081 °C
Process finished with exit code 0
```

1. Viscosidad a 35°C: La viscosidad interpolada es **0.0120 Pa·s**
2. Temperatura para  $\mu = 0.0100$  Pa·s: La temperatura interpolada es aproximadamente **43.1081°C**.

## Ejercicio 2: Finanzas

Un acción tiene los siguientes precios históricos:

Día	Precio (USD)
5	28.50
15	31.20

1. Interpole el precio en el día 10
2. Grafique los puntos y la recta interpolante (usando matplotlib)
3. Calcule el error si el precio real en día 10 fue \$29.80

```
# Ejercicio 2: Finanzas
import matplotlib.pyplot as plt

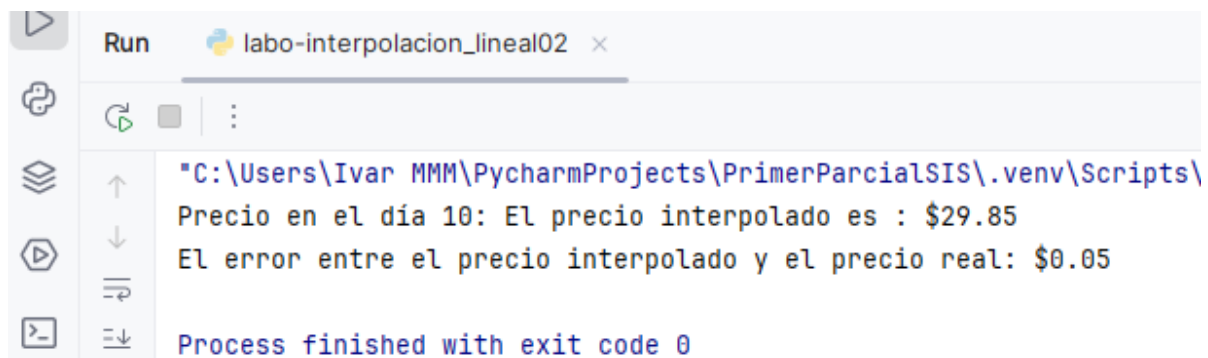
def interpolacion_lineal(x0, y0, x1, y1, x):
    y = y0 + (y1 - y0) / (x1 - x0) * (x - x0)
    return y

# Datos
dia0 = 5
precio0 = 28.50
dia1 = 15
precio1 = 31.20

# Calcular el precio en el día 10
dia_interp = 10
precio_interp = interpolacion_lineal(dia0, precio0, dia1, precio1, dia_interp)
print(f"Precio en el día 10: El precio interpolado es : ${precio_interp:.2f}")

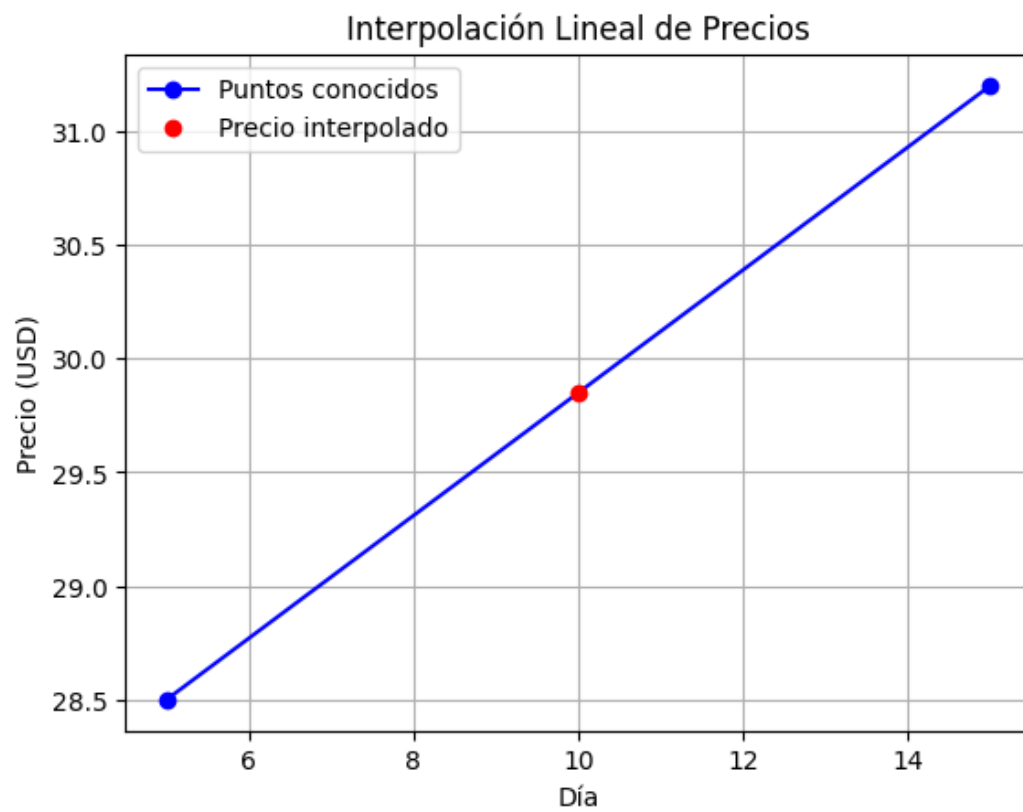
# Graficar
plt.plot([dia0, dia1], [precio0, precio1], 'bo-', label='Puntos conocidos')
plt.plot(dia_interp, precio_interp, 'ro', label='Precio interpolado')
plt.xlabel('Día')
plt.ylabel('Precio (USD)')
plt.legend()
plt.title('Interpolación Lineal de Precios')
plt.grid(True)
plt.show()

# Calcular el error
precio_real = 29.80
error = abs(precio_interp - precio_real)
print(f"El error entre el precio interpolado y el precio real: ${error:.2f}")
```



```
Run labo-interpolacion_lineal02 x
"C:\Users\Ivar MMM\PycharmProjects\PrimerParcialSIS\.venv\Scripts\
Precio en el día 10: El precio interpolado es : $29.85
El error entre el precio interpolado y el precio real: $0.05
Process finished with exit code 0
```

1. Precio en el día 10: El precio interpolado es: **\$29.85**.
2. Gráfica: La gráfica muestra los puntos conocidos y el precio interpolado en el día 10.



3. El error entre el precio interpolado y el precio real es: **\$0.05**

# Implementación de Interpolación Cuadrática

## 1. Lógica del Programa

La interpolación cuadrática requiere:

1. Tres puntos de datos  $(x_0, y_0)$ ,  $(x_1, y_1)$ ,  $(x_2, y_2)$ .
2. Resolver el sistema de ecuaciones para  $a$ ,  $b$ ,  $c$  o usar la fórmula de Lagrange.
3. Evaluar el polinomio  $P(x)$  en el punto deseado.

## 2. Código en Python

```
1 def interpolacion_cuadratica(x0, y0, x1, y1, x2, y2, x):
2     """
3     Interpolación cuadrática usando el método de Lagrange.
4
5     Parámetros:
6         x0, y0, x1, y1, x2, y2: Tres puntos de datos.
7         x: Punto a interpolar.
8
9     Retorna:
10        Valor interpolado y = P(x).
11    """
12    term0 = y0 * ((x - x1) * (x - x2)) / ((x0 - x1) * (x0 - x2))
13    term1 = y1 * ((x - x0) * (x - x2)) / ((x1 - x0) * (x1 - x2))
14    term2 = y2 * ((x - x0) * (x - x1)) / ((x2 - x0) * (x2 - x1))
15    return term0 + term1 + term2
16
17 # Ejemplo de uso
18 x0, y0 = 1, 2
19 x1, y1 = 3, 4
20 x2, y2 = 5, 3
21 x_interpoliar = 2
22
23 resultado = interpolacion_cuadratica(x0, y0, x1, y1, x2, y2,
24                                     x_interpoliar)
25 print(f"El valor interpolado en x={x_interpoliar} es y={resultado:.4f}")
```

### 3. Explicación del Código

- `term0`, `term1`, `term2` corresponden a los términos del polinomio de Lagrange.
- El resultado final es la suma de estos tres términos.
- El ejemplo interpola el valor en  $x = 2$  usando los puntos  $(1, 2)$ ,  $(3, 4)$  y  $(5, 3)$ .

# Interpolación Cuadrática: Fundamentos Teóricos

## 1. Introducción

La interpolación cuadrática es un método numérico para estimar valores desconocidos de una función utilizando un polinomio de segundo grado (cuadrático). Es una mejora sobre la interpolación lineal al considerar la curvatura de los datos.

## 2. Definición

Dados tres puntos  $(x_0, y_0)$ ,  $(x_1, y_1)$  y  $(x_2, y_2)$ , donde  $x_0 < x_1 < x_2$ , el polinomio interpolante cuadrático  $P(x)$  tiene la forma:

$$P(x) = a(x - x_0)^2 + b(x - x_0) + c$$

Los coeficientes  $a$ ,  $b$  y  $c$  se determinan resolviendo el sistema de ecuaciones:

$$\begin{cases} P(x_0) = y_0 = c \\ P(x_1) = y_1 = a(x_1 - x_0)^2 + b(x_1 - x_0) + c \\ P(x_2) = y_2 = a(x_2 - x_0)^2 + b(x_2 - x_0) + c \end{cases}$$

## 3. Método de Lagrange

Una alternativa es usar el polinomio de Lagrange para interpolación cuadrática:

$$P(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

## 4. Aplicaciones

- Estimación de valores intermedios en datos discretos.
- Optimización numérica (método de Brent).
- Gráficos por computadora para suavizar curvas.

# Laboratorio 3: Interpolación Cuadrática

5 de mayo de 2025

## 1. Objetivos

- Comprender los fundamentos teóricos de la interpolación cuadrática.
- Implementar el método en Python para resolver problemas prácticos.
- Analizar gráficamente los resultados obtenidos.

## 2. Marco Teórico

### 2.1. Interpolación Cuadrática

Dados tres puntos no colineales  $(x_0, y_0)$ ,  $(x_1, y_1)$  y  $(x_2, y_2)$ , el polinomio interpolante cuadrático  $P(x)$  puede expresarse como:

$$P(x) = a(x - x_0)^2 + b(x - x_0) + c$$

donde los coeficientes se calculan resolviendo:

$$\begin{cases} c = y_0 \\ a(x_1 - x_0)^2 + b(x_1 - x_0) + c = y_1 \\ a(x_2 - x_0)^2 + b(x_2 - x_0) + c = y_2 \end{cases}$$

### 2.2. Fórmula de Lagrange

Alternativamente, usando Lagrange:

$$P(x) = \sum_{k=0}^2 y_k \prod_{\substack{i=0 \\ i \neq k}}^2 \frac{x - x_i}{x_k - x_i}$$

## 3. Problema 1: Análisis Teórico

**Datos:**

- Punto A:  $(1, 2)$
- Punto B:  $(3, 1)$
- Punto C:  $(4, 3)$

**Preguntas:**

1. Demuestre que los puntos no son colineales calculando el determinante:

$$\begin{vmatrix} 1 & 2 & 1 \\ 3 & 1 & 1 \\ 4 & 3 & 1 \end{vmatrix}$$

2. Escriba el sistema de ecuaciones para encontrar  $a$ ,  $b$ ,  $c$  en  $P(x) = a(x - 1)^2 + b(x - 1) + c$ .
3. Calcule manualmente los coeficientes usando el método de Lagrange.

## 4. Problema 2: Implementación en Python

Implemente una función en Python que realice interpolación cuadrática y grafique los resultados.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def interpolacion_cuadratica(x0, y0, x1, y1, x2, y2, x):
5     # Implemente aquí la fórmula de Lagrange
6     pass
7
8 # Datos del problema 1
9 x0, y0 = 1, 2
10 x1, y1 = 3, 1
11 x2, y2 = 4, 3
12
13 # Generar puntos para graficar
14 x_vals = np.linspace(0, 5, 100)
15 y_vals = [interpolacion_cuadratica(x0, y0, x1, y1, x2, y2, x) for x in x_vals]
16
17 # Graficar
18 plt.figure(figsize=(8, 6))
19 plt.scatter([x0, x1, x2], [y0, y1, y2], color='red', label='Puntos dados')
20 plt.plot(x_vals, y_vals, label='Polinomio interpolante', linestyle='--')
21 plt.xlabel('x')
22 plt.ylabel('y')
23 plt.legend()
24 plt.grid()
25 plt.title('Interpolación Cuadrática')
26 plt.show()
```

### Tareas:

1. Complete la función `interpolacion_cuadratica`.
2. Ejecute el código y adjunte la gráfica resultante.
3. Interpole el valor en  $x = 2,5$  y discuta la precisión del resultado.

## 5. Análisis de Resultados

Responda:

1. ¿Por qué la interpolación cuadrática es más precisa que la lineal para estos datos?
2. ¿Qué ocurriría si los tres puntos estuvieran alineados?
3. ¿Cómo afecta la elección de los puntos a la forma de la curva?



## Solución al Problema 1: Análisis Teórico

### 1. Demostrar que los puntos no son colineales calculando el determinante:

Para demostrar que los puntos no son colineales, calculamos el determinante de la matriz formada por los puntos dados:

$$\begin{vmatrix} 1 & 2 & 1 \\ 3 & 1 & 1 \\ 4 & 3 & 1 \end{vmatrix}$$

El determinante se calcula como:

$$\text{Determinante} = 1 \cdot (1 \cdot 1 - 1 \cdot 3) - 2 \cdot (3 \cdot 1 - 1 \cdot 4) + 1 \cdot (3 \cdot 3 - 1 \cdot 4)$$

$$= 1 \cdot (1 - 3) - 2 \cdot (3 - 4) + 1 \cdot (9 - 4)$$

$$= 1 \cdot (-2) - 2 \cdot (-1) + 1 \cdot 5$$

$$= -2 + 2 + 5 = 5$$

Como el determinante es diferente de cero (5), los puntos no son colineales.

### 2. Escribir el sistema de ecuaciones para encontrar $a, b, c$ en $P(x) = a(x - 1)^2 + b(x - 1) + c$ :

Para encontrar los coeficientes  $a, b, c$ , usamos los puntos dados:

1. Para el punto  $(1, 2)$ :

$$P(1) = a(1 - 1)^2 + b(1 - 1) + c = c = 2$$

2. Para el punto  $(3, 1)$ :

$$P(3) = a(3 - 1)^2 + b(3 - 1) + c = 4a + 2b + c = 1$$

3. Para el punto  $(4, 3)$ :

$$P(4) = a(4 - 1)^2 + b(4 - 1) + c = 9a + 3b + c = 3$$

Por lo tanto, el sistema de ecuaciones es:

$$\begin{cases} c = 2 \\ 4a + 2b + c = 1 \\ 9a + 3b + c = 3 \end{cases}$$

### 3. Calcular manualmente los coeficientes usando el método de Lagrange:

Para usar el método de Lagrange, primero resolvemos el sistema de ecuaciones:

1. De la primera ecuación, sabemos que  $c = 2$ .
2. Sustituyendo  $c = 2$  en las otras ecuaciones:

$$4a + 2b + 2 = 1 \implies 4a + 2b = -1$$

$$9a + 3b + 2 = 3 \implies 9a + 3b = 1$$

3. Resolviendo el sistema de ecuaciones:

$$\begin{cases} 4a + 2b = -1 \\ 9a + 3b = 1 \end{cases}$$

Multiplicamos la primera ecuación por 3 y la segunda por 2 para eliminar  $b$ :

$$12a + 6b = -3$$

$$18a + 6b = 2$$

Restamos la primera ecuación de la segunda:

$$(18a + 6b) - (12a + 6b) = 2 - (-3)$$

$$6a = 5 \implies a = \frac{5}{6}$$

Sustituyendo  $a = \frac{5}{6}$  en  $4a + 2b = -1$ :

$$4\left(\frac{5}{6}\right) + 2b = -1 \implies \frac{20}{6} + 2b = -1 \implies \frac{10}{3} + 2b = -1$$

$$2b = -1 - \frac{10}{3} = -\frac{13}{3} \implies b = -\frac{13}{6}$$

Por lo tanto, los coeficientes son:

$$a = \frac{5}{6}, \quad b = -\frac{13}{6}, \quad c = 2$$

El polinomio interpolante cuadrático es:

$$P(x) = \frac{5}{6}(x-1)^2 - \frac{13}{6}(x-1) + 2$$

---

## Análisis de Resultados

### Respuestas:

**1. ¿Por qué la interpolación cuadrática es más precisa que la lineal para estos datos?**

La interpolación cuadrática es más precisa que la lineal para estos datos porque los puntos dados no son colineales. La interpolación lineal solo puede ajustarse a una línea recta, mientras que la interpolación cuadrática puede ajustarse a una curva, lo que permite una mejor aproximación de los datos que no siguen una línea recta.

**2. ¿Qué ocurriría si los tres puntos estuvieran alineados?**

Si los tres puntos estuvieran alineados, la interpolación lineal sería suficiente para describir la relación entre los puntos. En este caso, la interpolación cuadrática no aportaría ninguna ventaja adicional, ya que una línea recta sería la mejor aproximación para los datos. Además, el determinante de la matriz formada por los puntos sería cero, indicando colinealidad.

**3. ¿Cómo afecta la elección de los puntos a la forma de la curva?**

La elección de los puntos afecta significativamente a la forma de la curva interpolada. Diferentes conjuntos de puntos pueden llevar a diferentes formas de la curva, incluso si se utiliza el mismo método de interpolación. La posición y el valor de los puntos determinan los coeficientes del polinomio interpolante, lo que a su vez define la forma de la curva. Por ejemplo, puntos más dispersos pueden llevar a una curva más compleja, mientras que puntos más cercanos pueden resultar en una curva más suave.

# Método de Interpolación de Lagrange

## Definición del Método de Interpolación de Lagrange

El método de interpolación de Lagrange es una técnica usada para construir un polinomio de interpolación que pasa exactamente por un conjunto dado de puntos. Supongamos que tenemos  $n + 1$  puntos  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , y deseamos encontrar un polinomio de grado  $n$ ,  $P(x)$ , que pase por estos puntos. El polinomio de Lagrange para estos puntos se define como:

$$P(x) = \sum_{i=0}^n y_i L_i(x)$$

donde cada  $L_i(x)$  es un polinomio base de Lagrange, definido como:

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

Cada  $L_i(x)$  tiene la propiedad de que es igual a 1 en  $x = x_i$  y 0 en los demás puntos  $x = x_j$  con  $j \neq i$ . De este modo, el polinomio  $P(x)$  toma el valor  $y_i$  en el punto  $x = x_i$ , garantizando que pase por todos los puntos dados.

## Ejemplo Práctico 1

Consideremos los puntos  $(1, 2)$ ,  $(2, 3)$ , y  $(3, 5)$ . Queremos encontrar el polinomio de interpolación que pase por estos puntos.

1. **Definir los términos**  $L_0(x)$ ,  $L_1(x)$ , y  $L_2(x)$ :

$$\begin{aligned} L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 2)(x - 3)}{(1 - 2)(1 - 3)} = \frac{(x - 2)(x - 3)}{2}, \\ L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x - 1)(x - 3)}{(2 - 1)(2 - 3)} = -(x - 1)(x - 3), \\ L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(x - 1)(x - 2)}{(3 - 1)(3 - 2)} = \frac{(x - 1)(x - 2)}{2}. \end{aligned}$$

2. **Construir el polinomio de interpolación**  $P(x)$ :

$$P(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x)$$

Sustituyendo los valores de  $y_0 = 2$ ,  $y_1 = 3$ , y  $y_2 = 5$ :

$$P(x) = 2 \cdot \frac{(x-2)(x-3)}{2} + 3 \cdot (-(x-1)(x-3)) + 5 \cdot \frac{(x-1)(x-2)}{2}$$

Simplificando cada término, obtenemos el polinomio final.

## Ejemplo Práctico 2

Supongamos que tenemos los puntos  $(0, 1)$ ,  $(1, e)$ , y  $(2, e^2)$ , donde  $e$  es el número de Euler (aproximadamente 2.718). Queremos encontrar el polinomio que pasa por estos puntos.

1. **Definir los términos  $L_0(x)$ ,  $L_1(x)$ , y  $L_2(x)$ :**

$$L_0(x) = \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{(x-1)(x-2)}{2},$$

$$L_1(x) = \frac{(x-0)(x-2)}{(1-0)(1-2)} = -x(x-2),$$

$$L_2(x) = \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{x(x-1)}{2}.$$

2. **Construir el polinomio de interpolación  $P(x)$ :**

Sustituyendo los valores  $y_0 = 1$ ,  $y_1 = e$ , y  $y_2 = e^2$ :

$$P(x) = 1 \cdot \frac{(x-1)(x-2)}{2} + e \cdot (-x(x-2)) + e^2 \cdot \frac{x(x-1)}{2}$$

Simplificando cada término, obtenemos el polinomio de interpolación que pasa por los puntos dados.

## Análisis

El polinomio de interpolación de Lagrange garantiza una solución única para el conjunto de puntos dado, sin necesidad de resolver un sistema de ecuaciones para los coeficientes del polinomio. Cada término  $L_i(x)$  se calcula de forma independiente, permitiendo una construcción directa del polinomio que pasa por todos los puntos.

# Interpolación de Lagrange: Implementación en Python y Octave

## Interpolación de Lagrange

El método de interpolación de Lagrange permite aproximar el valor de una función en un punto intermedio usando los valores conocidos de la función en puntos específicos. Dado un conjunto de puntos  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , el polinomio de interpolación de Lagrange  $P(x)$  está dado por:

$$P(x) = \sum_{i=0}^{n-1} y_i \cdot L_i(x)$$

donde  $L_i(x)$  es el polinomio de Lagrange asociado al  $i$ -ésimo punto y se define como:

$$L_i(x) = \prod_{\substack{0 \leq j \leq n-1 \\ j \neq i}} \frac{x - x_j}{x_i - x_j}$$

## Implementación en Python

El siguiente código en Python implementa el método de interpolación de Lagrange y calcula el valor de  $P(x)$  para un valor específico de  $x$ .

```
import numpy as np

def lagrange_interpolation(x_points, y_points, x):
    n = len(x_points) # Número de puntos
    P = 0 # Polinomio de interpolación

    # Itera sobre cada punto
    for i in range(n):
        # Calcula el término de Lagrange L_i(x) para cada i
        Li = 1
        for j in range(n):
            if i != j:
                Li *= (x - x_points[j]) / (x_points[i] - x_points[j])
        P += y_points[i] * Li
```

```

        # Suma el término al polinomio con el valor correspondiente de y_points[i]
        P += y_points[i] * Li

    return P

```

## Explicación de la lógica del código

- `import numpy as np`: Importa la biblioteca NumPy, la cual es útil para trabajar con datos numéricos y puede facilitar el manejo de datos en cálculos avanzados.
- `def lagrange_interpolation(x_points, y_points, x)`: Define la función de interpolación de Lagrange. La función recibe tres parámetros:
  - `x_points`: Lista de los valores  $x$  en los puntos conocidos.
  - `y_points`: Lista de los valores  $y$  en los puntos conocidos.
  - `x`: El valor de  $x$  donde queremos evaluar el polinomio.
- `n = len(x_points)`: Calcula el número de puntos,  $n$ , en el conjunto de datos proporcionado.
- `P = 0`: Inicializa el valor del polinomio de interpolación,  $P$ , en 0. Este valor irá acumulando la suma de cada término de Lagrange multiplicado por su correspondiente valor  $y_i$ .
- `for i in range(n)`: Bucle principal que recorre cada índice  $i$  para calcular cada polinomio de Lagrange  $L_i(x)$ .
- `Li = 1`: Inicializa el término de Lagrange  $L_i(x)$  para el índice actual  $i$ . Se actualiza en el bucle interno.
- `for j in range(n)`: Segundo bucle para calcular el producto en  $L_i(x)$ . Este bucle recorre todos los valores de  $j$ , salvo cuando  $j = i$ .
- `if i != j`: Condicional que evita la división por cero y asegura que no se usa el mismo índice  $i$  en el cálculo de  $L_i(x)$ .
- `Li *= (x - x_points[j]) / (x_points[i] - x_points[j])`: Actualiza  $L_i(x)$  multiplicando por el factor correspondiente para el valor de  $j$ .
- `P += y_points[i] * Li`: Suma el producto de  $y_i$  y  $L_i(x)$  al polinomio  $P$ , acumulando el resultado parcial para obtener el polinomio final de interpolación.
- `return P`: Devuelve el valor de  $P(x)$ , el polinomio de interpolación evaluado en el valor de  $x$  dado.

## Implementación en Octave

La siguiente implementación en Octave realiza la misma función que el código en Python:

```
function P = lagrange_interpolation(x_points, y_points, x)
    n = length(x_points); % Número de puntos
    P = 0; % Polinomio de interpolación inicializado en 0

    % Itera sobre cada punto
    for i = 1:n
        % Calcula el término de Lagrange L_i(x) para cada i
        Li = 1;
        for j = 1:n
            if i != j
                Li *= (x - x_points(j)) / (x_points(i) - x_points(j));
            end
        end

        % Suma el término al polinomio con el valor correspondiente de y_points(i)
        P += y_points(i) * Li;
    end
end
```

## Ejemplo Práctico

Supongamos que tenemos los siguientes puntos:

$$x = [1, 2, 3]$$

$$y = [2, 3, 5]$$

Queremos encontrar el valor del polinomio de interpolación en  $x = 2.5$ .

### Python

```
x_points = [1, 2, 3]
y_points = [2, 3, 5]
x = 2.5

result = lagrange_interpolation(x_points, y_points, x)
print("El valor del polinomio en x = 2.5 es:", result)
```

### Octave

```
x_points = [1, 2, 3];
y_points = [2, 3, 5];
```



```
x = 2.5;

result = lagrange_interpolation(x_points, y_points, x);
disp(["El valor del polinomio en x = 2.5 es: ", num2str(result)]);
```

En ambos casos, el valor calculado representa el valor aproximado de la función en  $x = 2.5$  usando interpolación de Lagrange.

# Metodo de Interpolacion de Diferencias Divididas

## Definicion y Fundamentos del Metodo de Diferencias Divididas

El metodo de diferencias divididas es una tecnica de interpolacion utilizada para encontrar un polinomio de interpolacion que pase por un conjunto de puntos dados. Es particularmente util por su estructura recursiva y eficiencia en el calculo de coeficientes cuando se anaden nuevos puntos.

Dado un conjunto de  $n+1$  puntos  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , el polinomio de interpolacion en diferencias divididas esta definido como:

$$P(x) = f[x_0] + f[x_0, x_1](x-x_0) + f[x_0, x_1, x_2](x-x_0)(x-x_1) + \dots + f[x_0, x_1, \dots, x_n](x-x_0)(x-x_1) \dots (x-x_{n-1})$$

Donde  $f[x_i, x_{i+1}, \dots, x_j]$  representa las diferencias divididas y se calculan de forma recursiva:

- Primera diferencia dividida:

$$f[x_i] = y_i$$

- Segunda diferencia dividida:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

- Tercera y siguientes:

$$f[x_i, x_{i+1}, \dots, x_j] = \frac{f[x_{i+1}, \dots, x_j] - f[x_i, \dots, x_{j-1}]}{x_j - x_i}$$

Las diferencias divididas permiten construir el polinomio de forma progresiva y adaptarlo a nuevos puntos sin necesidad de recalcular desde cero.

## Funcionamiento del Metodo

El metodo de diferencias divididas sigue un procedimiento que se puede esquematizar en los siguientes pasos:

1. Se construye una tabla de diferencias divididas con las columnas correspondientes a  $x_i$  y  $f[x_i]$ ,  $f[x_i, x_{i+1}]$ ,  $f[x_i, x_{i+1}, x_{i+2}]$ , etc.
2. Los elementos de la primera columna corresponden a los valores de  $y_i$ .
3. Cada columna sucesiva se calcula usando la formula recursiva de diferencias divididas.
4. Una vez completada la tabla, los coeficientes del polinomio de interpolacion se obtienen de la primera fila de cada columna.

## Ejemplo Practico

Dado el siguiente conjunto de puntos:  $(1, 2)$ ,  $(2, 3)$ ,  $(4, 5)$ , construiremos el polinomio de diferencias divididas.

### Paso 1: Construir la Tabla de Diferencias Divididas

$x$	$f[x]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$
1	2		
2	3	$\frac{3-2}{2-1} = 1$	
4	5	$\frac{5-3}{4-2} = 1$	$\frac{1-1}{4-1} = 0$

### Paso 2: Construir el Polinomio

Usando los coeficientes obtenidos de la tabla, el polinomio es:

$$P(x) = 2 + 1(x - 1) + 0(x - 1)(x - 2)$$

## Implementacion en Python

```
def diferencias_divididas(x_points, y_points):
    n = len(x_points)
    coef = np.zeros([n, n])
    coef[:, 0] = y_points

    for j in range(1, n):
        for i in range(n - j):
            coef[i, j] = (coef[i + 1, j - 1] - coef[i, j - 1]) / (x_points[i + j] - x_points[i])

    return coef[0, :]

# Ejemplo de uso
x_points = [1, 2, 4]
y_points = [2, 3, 5]
coefs = diferencias_divididas(x_points, y_points)
print("Coeficientes del polinomio:", coefs)
```

## Explicacion Linea por Linea delCodigo en Python

- `def diferencias_divididas(x_points, y_points):` Define la funcion que recibe los puntos  $x$  e  $y$ .
- `n = len(x_points):` Calcula la cantidad de puntos  $n$ .
- `coef = np.zeros([n, n]):` Inicializa una matriz de ceros de  $n \times n$  para almacenar las diferencias divididas.
- `coef[:, 0] = y_points:` Asigna la primera columna de la matriz con los valores de  $y$ .
- `for j in range(1, n)::` Itera sobre las columnas de la matriz, comenzando desde la segunda columna.
- `for i in range(n - j)::` Itera sobre las filas para calcular las diferencias divididas en la columna actual.
- `coef[i, j] = (coef[i + 1, j - 1] - coef[i, j - 1]) / (x_points[i + j] - x_points[i]):` Calcula el valor de la diferencia dividida usando la formula recursiva.
- `return coef[0, :]:` Retorna la primera fila de la matriz, que contiene los coeficientes del polinomio.

## Implementacion en Octave

```
function coefs = diferencias_divididas(x_points, y_points)
    n = length(x_points);
    coef = zeros(n, n);
    coef(:, 1) = y_points';

    for j = 2:n
        for i = 1:(n - j + 1)
            coef(i, j) = (coef(i + 1, j - 1) - coef(i, j - 1)) / (x_points(i + j - 1) - x_points(i));
        end
    end

    coefs = coef(1, :);
end

% Ejemplo de uso
x_points = [1, 2, 4];
y_points = [2, 3, 5];
coefs = diferencias_divididas(x_points, y_points);
fprintf("Coeficientes del polinomio:\n");
fprintf("%f ", coefs);
```

# Metodo de Interpolacion de Diferencias Divididas

## Definicion y Fundamentos del Metodo de Diferencias Divididas

El metodo de diferencias divididas es una tecnica de interpolacion utilizada para encontrar un polinomio de interpolacion que pase por un conjunto de puntos dados. Es particularmente util por su estructura recursiva y eficiencia en el calculo de coeficientes cuando se anaden nuevos puntos.

Dado un conjunto de  $n+1$  puntos  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ , el polinomio de interpolacion en diferencias divididas esta definido como:

$$P(x) = f[x_0] + f[x_0, x_1](x-x_0) + f[x_0, x_1, x_2](x-x_0)(x-x_1) + \dots + f[x_0, x_1, \dots, x_n]$$

$$(x-x_0)(x-x_1) \cdots (x-x_{n-1})$$

Donde  $f[x_i, x_{i+1}, \dots, x_j]$  representa las diferencias divididas y se calculan de forma recursiva:

- Primera diferencia dividida:

$$f[x_i] = y_i$$

- Segunda diferencia dividida:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

- Tercera y siguientes:

$$f[x_i, x_{i+1}, \dots, x_j] = \frac{f[x_{i+1}, \dots, x_j] - f[x_i, \dots, x_{j-1}]}{x_j - x_i}$$

Las diferencias divididas permiten construir el polinomio de forma progresiva y adaptarlo a nuevos puntos sin necesidad de recalcular desde cero.

## Funcionamiento del Metodo

El metodo de diferencias divididas sigue un procedimiento que se puede esquematizar en los siguientes pasos:

1. Se construye una tabla de diferencias divididas con las columnas correspondientes a  $x_i$  y  $f[x_i]$ ,  $f[x_i, x_{i+1}]$ ,  $f[x_i, x_{i+1}, x_{i+2}]$ , etc.
2. Los elementos de la primera columna corresponden a los valores de  $y_i$ .
3. Cada columna sucesiva se calcula usando la formula recursiva de diferencias divididas.
4. Una vez completada la tabla, los coeficientes del polinomio de interpolacion se obtienen de la primera fila de cada columna.

## Ejemplo Practico

Dado el siguiente conjunto de puntos:  $(1, 2)$ ,  $(2, 3)$ ,  $(4, 5)$ , construiremos el polinomio de diferencias divididas.

### Paso 1: Construir la Tabla de Diferencias Divididas

$x$	$f[x]$	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$
1	2		
2	3	$\frac{3-2}{2-1} = 1$	
4	5	$\frac{5-3}{4-2} = 1$	$\frac{1-1}{4-1} = 0$

### Paso 2: Construir el Polinomio

Usando los coeficientes obtenidos de la tabla, el polinomio es:

$$P(x) = 2 + 1(x - 1) + 0(x - 1)(x - 2)$$

## Conclusion

El metodo de interpolacion de diferencias divididas es una herramienta poderosa que permite construir polinomios de forma eficiente y adaptativa. Su estructura recursiva facilita el calculo de nuevos coeficientes al agregar puntos, haciendo que el proceso sea mas comodo comparado con otros metodos de interpolacion.

# Laboratorio Lagrange y Diferencias Divididas

Alvarez Alvarez Yhorel Yhared

May 2025

## Objetivos

- Implementar los métodos de interpolación de Lagrange y Diferencias Divididas mediante programación.
- Aplicar lógica de programación para resolver problemas de ingeniería y ciencias.

## 1. Ejercicios de Interpolación de Lagrange

### Ejercicio 1: Temperatura en Reactor Químico

Se midió la temperatura (en °C) de un reactor en distintos horarios:

Hora (x)	Temperatura (°C) (y)
1	68
3	72
5	80

#### Tareas:

1. Encontrar el polinomio interpolante de Lagrange manualmente.
2. Implementar en Python la función `lagrange(x, puntos)`.
3. Estimar la temperatura a las 4:00 (usar el código).

## Ejercicio 2: Producción de Cultivos

Producción de trigo (ton) vs agua aplicada (cm):

Agua (cm) (x)	Producción (ton) (y)
10	5
20	7
30	6

**Tareas:**

1. Generalizar el código para  $n$  puntos usando bucles.
2. Calcular la producción con 25 cm de agua.

## 2. Ejercicios de Diferencias Divididas

### Ejercicio 3: Velocidad de un Cohete

Datos de velocidad (km/s) vs tiempo (s):

Tiempo (s) (x)	Velocidad (km/s) (y)
2	5
4	17
6	37

**Tareas:**

1. Construir la tabla de diferencias divididas manualmente.
2. Programar la función `diferencias_divididas(puntos)`.
3. Estimar la velocidad en  $t = 5$  s.



Capacidad (MW) (x)	Costo (\$M) (y)
1	10
3	25
5	55

## Ejercicio 4: Costo de Energía Solar

Costo (\$) vs capacidad (MW):

**Tareas:**

1. Encontrar el polinomio interpolante y evaluar para 4 MW.
2. Optimizar el código para reutilizar la tabla de diferencias.

## Código Base

### Método de Lagrange

```
def lagrange(x, puntos):
    resultado = 0.0
    n = len(puntos)
    for i in range(n):
        xi, yi = puntos[i]
        term = yi
        for j in range(n):
            if j != i:
                xj, yj = puntos[j]
                term *= (x - xj) / (xi - xj)
        resultado += term
    return resultado
```

### Método de Diferencias Divididas

```
def diferencias_divididas(puntos):
    n = len(puntos)
    tabla = [ [0]*n for _ in range(n) ]
    for i in range(n):
```

```

        tabla[i][0] = puntos[i][1]
    for j in range(1, n):
        for i in range(n - j):
            tabla[i][j] = (tabla[i+1][j-1] - tabla[i][j-1])
                        / (puntos[i+j][0] - puntos[i][0])
    return tabla[0] # Coeficientes

```

## Preguntas de Análisis

1. ¿Qué ventajas tiene programar estos métodos vs resolverlos manualmente?
2. ¿Cómo afecta el número de puntos a la precisión del resultado?
3. ¿Qué modificaciones harías para mejorar la eficiencia del código?