INTERPOLACIÓN: MÉTODOS DE INTERPOLACIÓN

La interpolación se refiere a la técnica de estimar valores intermedios entre puntos conocidos en un conjunto de datos. En otras palabras, se utiliza para encontrar valores dentro de un rango de datos a partir de valores discretos que ya están disponibles. Esta técnica se aplica en diversas áreas, como gráficos computacionales, análisis de datos, procesamiento de señales, entre otros.

Esta técnica resulta de gran utilidad en muchos sectores, y entre sus principales aplicaciones destacan:

- **Gráficos computacionales**: para dibujar curvas suaves a partir de puntos discretos.
- **Análisis de datos**: para estimar valores entre puntos de una tabla o serie temporal.
- **Procesamiento de imágenes**: para la ampliación o reducción de imágenes.
- **Cálculo numérico**: para resolver ecuaciones diferenciales o problemas de optimización.

Además, existen diferentes métodos de interpolación, a destacar tres: la forma matricial, la interpolación de Lagrange y la interpolación de Newton.

MÉTODOS DE INTERPOLACIÓN:

1. **MATRICES O COEFICIENTES INDETERMINADOS:**

Este método consiste en la elaboración de matrices para obtener los coeficientes del polinomio interpolador que se desea hallar, apenas con un par de puntos.

Rellenando la matriz siguiendo la fórmula: s[i]^(j - 1)

$$\begin{pmatrix} 1 & s_1 & \cdots & s_1^{(J-1)} & \cdots & s_1^{(n-1)} \\ 1 & s_2 & \cdots & s_2^{(J-1)} & \cdots & s_2^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & s_I & \cdots & s_I^{(J-1)} & \cdots & s_I^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & s_n & \cdots & s_n^{(J-1)} & \cdots & s_n^{(n-1)} \end{pmatrix} \cdot \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_I \\ \vdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} f(s_1) \\ f(s_2) \\ \vdots \\ f(s_I) \\ \vdots \\ f(s_n) \end{Bmatrix}$$

Tras resolver el sistema, utilizamos los coeficientes a[i] para rellenar el polinomio interpolador:

p(x)=a1+a2·x+a3·x^2+⋯+an·x^(n-1)

Código: Resolver sistema utilizando funciones:

```
Resolucion_ecuaciones = function(s, fS){

  n = length(s)              # Número de puntos
  M = matrix(0, n, n)        # Matriz vacía
  b = numeric(n)             # Vector vacío

  # Construir la matriz del sistema

  for(i in 1:n){
    for(j in 1:n){
      M[i, j] = s[i]^(j - 1)    # Llenar matriz con potencias de los puntos
    }
    b[i] = fS[i]              # Llenar el vector con los valores dados
  }
  a = solve(M) %*% b          # Resolver el sistema M * a = b
  return(a)                   # Devolver los coeficientes
}
```

✏️ **¿Qué hace esta función?**

1. **Entrada**:
   - `s`: vector con valores de x (los puntos en el eje horizontal).
   - `fS`: vector con valores de f(x) (los puntos en el eje vertical).
2. **Proceso**:
   - Crea una matriz `M` usando las potencias de los valores de s.
   - Llena el vector `b` con los valores de fS.
   - Resuelve el sistema de ecuaciones $M \cdot a = b$ para encontrar los coeficientes del polinomio.
3. **Salida**:
   - Un vector `a` con los coeficientes del polinomio.

Crear polinomio:

```
Poly_Ecu = function(x, a){
  n = length(a)        # Número de coeficientes
  Px = 0               # Iniciar el valor del polinomio

 for (j in 1:n) {
   Px = Px + a[j] * (x^(j - 1))  # Sumar cada término del polinomio
 }
 return(Px)           # Devolver el valor del polinomio
}
```

✏️ **¿Qué hace esta función?**

1. **Entrada**:
   - x: valor donde queremos evaluar el polinomio.
   - a: vector con los coeficientes del polinomio.
2. **Proceso**:
   - Calcula el polinomio sumando cada término a[j] · xj−1.
3. **Salida**:
   - El valor del polinomio P(x) en el punto dado.

2. La fórmula del polinomio de Lagrange

El polinomio P(x)P(x) se define como:

$$P(x) = \sum_{i=1}^{n} f(s_i) \cdot L_i(x)$$

Donde:

- f(si): Es el valor de la función en el punto sisi.
- Li(x): Son los **polinomios base de Lagrange**, que se calculan así:

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - s_j}{s_i - s_j}$$

Esto significa que tomamos todos los puntos excepto **si** y los usamos para construir el polinomio base **Li(x)**.

Por ejemplo:

$L_1(x) = \frac{(x - s_2)(x - s_3)}{(s_1 - s_2)(s_1 - s_3)}$ Nos saltamos $(x - s1)/(s1 - s1)$, ya que en este caso i = j

Ahora solo queda multiplicar **L[i]** por **f(si)** para formar el polinomio

Código: Calculo de L(i) utilizando funciones

```
Polimonio_Lag = function(x, s) {
  n = length(s)            # Número de puntos de soporte
  L = numeric(n)           # Inicializamos un vector para almacenar los polinomios de base

  for (i in 1:n) {         # Para cada punto de soporte
    L[i] = 1               # Inicializamos el valor del polinomio L[i] en 1

   for (j in 1:n) {        # Iteramos sobre los demás puntos de soporte
     if (j != i) {         # Excluimos el caso donde i = j

       L[i] = L[i] * (x - s[j]) / (s[i] - s[j])  # Calculamos el producto de diferencias

     } } }
return(L)                  # Devolvemos el vector L con los polinomios de base
}
```

✏️ **¿Qué hace esta función?**

- **Toma:** Un punto `x` donde evaluar y un conjunto de soportes `s` (los puntos `s1, s2, ..., sn`).
- **Hace:** Calcula los polinomios base de Lagrange, que son las partes individuales necesarias para construir el polinomio interpolador.
- **Devuelve:** Un vector `L` con los valores de cada polinomio evaluados en `x`.

Crear Polinomio

```
Aplicar_Lag = function(Fs, L) {
  Px = 0                  # Inicializamos el polinomio en 0
  n = length(Fs)          # Obtenemos el número de puntos de soporte

  for (i in 1:n) {        # Para cada valor de s
    Px = Px + Fs[i] * L[i]  # Sumamos el producto de Fs[i] y L[i]
 }
return(Px)                # Devolvemos el valor del polinomio interpolador en
}
```

✏️ **¿Qué hace esta función?**

- **Toma:** Los valores de la función `Fx` en los puntos de soporte y el vector de polinomios base `L`.
- **Hace:** Combina los polinomios base con sus valores correspondientes de `Fx` para calcular el polinomio interpolador completo.
- **Devuelve:** El valor del polinomio interpolador `P(x)` en el punto evaluado.

3) FÓRMULA DE NEWTON:

$$p_2(x) = f[s_0] + f[s_0, s_1](x - s_0) + f[s_0, s_1, s_2](x - s_0)\,(x - s_1)$$

Para resolver este polinomio solo es necesario saber los puntos y saber cómo resolver las diferencias divididas (f[s0,s1]), lo cual tiene esta pinta:

$$f[s_0, s_1, s_2, \cdots, s_n] = \frac{f[s_1, s_2, \cdots, s_n] - f[s_0, s_1, s_2, \cdots, s_{n-1}]}{s_n - s_0}$$

Una manera de visualizar esto mejor, y ser capaz de encontrar los valores de esta ecuación es mediante la **Tabla de diferencias divididas**:

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	
s_0	$f[s_0]$	$f[s_0,s_1]$	$f[s_0,s_1,s_2]$	$f[s_0,s_1,s_2,s_3]$	$f[s_0,s_1,s_2,s_3,s_4]$	$i=0$
s_1	$f[s_1]$	$f[s_1,s_2]$	$f[s_1,s_2,s_3]$	$f[s_1,s_2,s_3,s_4]$	0	$i=1$
s_2	$f[s_2]$	$f[s_2,s_3]$	$f[s_2,s_3,s_4]$	0	0	$i=2$
s_3	$f[s_3]$	$f[s_3,s_4]$	0	0	0	$i=3$
s_4	$f[s_4]$	0	0	0	0	$i=4$

Donde se rellena la tabla en columnas , siempre utilizando los valores inferiores para calcular la próxima diferencia dividida.
Como en este ejemplo:

$s_0 = 0$	$f[s_0] = 0$	$f[s_0,s_1] = \frac{f[s_1]-f[s_0]}{s_1-s_0} = \frac{\frac{1}{\sqrt{2}}-0}{\frac{\pi}{4}-0} = \frac{2\sqrt{2}}{\pi}$	$f[s_0,s_1,s_2] = \frac{f[s_1,s_2]-f[s_0,s_1]}{s_2-s_0} = \frac{8(1-\sqrt{2})}{\pi^2}$
$s_1 = \frac{\pi}{4}$	$f[s_1] = \frac{1}{\sqrt{2}}$	$f[s_1,s_2] = \frac{f[s_2]-f[s_1]}{s_2-s_1} = \frac{1-\frac{1}{\sqrt{2}}}{\frac{\pi}{2}-\frac{\pi}{4}} = \frac{2(2-\sqrt{2})}{\pi}$	0
$s_2 = \frac{\pi}{2}$	$f[s_2] = 1$	0	0

Finalmente rellenamos el polinomio, multiplicando la diferencia dividida de la tabla por su productorio ((x-s1)(x-s2)(x-s3)).

$$p_n(x) = \sum_{i=0}^{n} f[s_0, s_1, \ldots, s_i] \prod_{j=0}^{i-1} (x - s_j)$$

Código: Crear tabla de diferencia divididas:

```
diferencias_divididas <- function(x, y) {
 n <- length(x) - 1                                  # Número de puntos - 1
 tabla <- matrix(0, nrow = n + 1, ncol = n + 1)  # Crear matriz para la tabla

  tabla[, 1] <- y                    # Llenar la primera columna con los valores de y

  # Calcular diferencias divididas

  for (j in 2:(n + 1)) {              # Iterar sobre columnas (orden creciente)
    for (i in 1:(n + 2 - j)) {        # Iterar sobre las filas restantes

      tabla[i, j] <- (tabla[i + 1, j - 1] - tabla[i, j - 1]) / (x[i + j - 1] - x[i])
# Recomendamos seguir los índices i/j con números en un papel, para entender el proceso.
    }
  }
  return(tabla)                       # Devolver la tabla completa
}
```

✏️ ¿Qué hace esta función?

- **Toma:** Dos vectores:
  - `x`: Puntos de soporte (coordenadas en el eje X).
  - `y`: Valores de la función en esos puntos (coordenadas en el eje Y).
- **Hace:** Calcula una **tabla de diferencias divididas**, que es una herramienta útil para construir un polinomio interpolador de Newton.
  - Comienza llenando la primera columna con los valores de `y` (Orden 0).
  - Luego, calcula las diferencias divididas sucesivas (Orden 1, Orden 2, ...) usando las fórmulas iterativas.
- **Devuelve:** Una matriz llamada `tabla` donde:
  - Las filas corresponden a los puntos `x`.
  - Las columnas contienen las diferencias divididas para cada orden.

Multiplicar coeficientes por su productorio:

```
polinomio_newton <- function(tabla_dif_div, x_orig, x_eval) {

  n <- ncol(tabla_dif_div) - 1                  # Número de términos del polinomio (grados)
  coef <- as.numeric(tabla_dif_div[1, ])        # coef = primera fila de tabla_dif_div
  resultados <- numeric(length(x_eval))         # Vector para almacenar los resultados

  for (k in seq_along(x_eval)) {                # Evaluar el polinomio en cada valor de x_eval
   x <- x_eval[k]
    resultado <- coef[1]                        # Iniciar con el término constante

    for (i in 1:n) {                            # Construir el polinomio sumando los términos
      producto <- 1                             # Producto de las diferencias (x - x_i)
      for (j in 1:i) {
        producto <- producto * (x - x_orig[j])  # Actualizar el producto
      }
      resultado <- resultado + coef[i + 1] * producto
    }
    resultados[k] <- resultado                  # Almacenar el resultado del polinomio en x
  }
  return(resultados)                            # Devolver los valores calculados
}
```

✏️ **¿Qué hace esta función?**

- **Toma:**
  - `tabla_dif_div`: Una tabla de diferencias divididas (creada previamente con la función `diferencias_divididas`).
  - `x_orig`: Puntos de soporte (coordenadas `x` originales donde se basa el polinomio).
  - `x_eval`: Valores de `x` donde quieres evaluar el polinomio.
- **Hace:**
  - Extrae los coeficientes del polinomio de Newton (primera fila de la tabla de diferencias divididas).
  - Evalúa el polinomio de Newton en cada valor de `x_eval` usando los coeficientes y la fórmula:

    $$P(x)=c0+c1(x-x1)+c2(x-x1)(x-x2)+\ldots$$

    - Se utilizan dos bucles:
      - Uno para iterar sobre los coeficientes (`i`).
      - Otro para calcular el producto acumulativo de las diferencias $(x-xj)(x-xj)$.
- **Devuelve:**
  - Un vector con los valores del polinomio evaluados en cada punto de `x_eval`.