Explicación del Código de Interpolación con el método de Newton en R

Este documento explica en detalle las funciones definidas en el código para calcular y evaluar polinomios interpoladores de Newton usando diferencias divididas. Se describen los objetivos y pasos de cada función.

1. "diferencias_divididas": función que crea la tabla de diferencias divididas

La tabla de diferencias divididas en el método de Newton es una herramienta clave para calcular los coeficientes del polinomio de interpolación de manera sistemática y organizada. Además, es muy útil para agregar nuevos puntos y analizar cómo los coeficientes del polinomio cambian con los datos.

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	
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	$i=4$

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

El siguiente código permite su obtención en R:

Código:

```
diferencias_divididas <- function(x, y) {
	n <- length(x) - 1
	# Inicializar la matriz para almacenar las diferencias divididas
	tabla <- matrix(0, nrow = n + 1, ncol = n + 1)
	# La primera columna son los valores de y
	tabla[, 1] <- y

	# Calcular las diferencias divididas
	for (j in 2:(n + 1)) {
		for (i in 1:(n + 2 - j)) {
			tabla[i, j] <- (tabla[i + 1, j - 1] - tabla[i, j - 1]) / (x[i + j - 1] - x[i])
			}
		}
	# Nombrar las filas y columnas
	rownames(tabla) <- x
	colnames(tabla) <- paste0("Orden ", 0:n)

	return(tabla)
}
```

Pasos principales:

1. Inicializa una matriz "tabla" para almacenar las diferencias divididas.
2. La primera columna se llena con los valores de "y" (valores de la función en los nodos).
3. Utiliza bucles anidados para calcular las diferencias divididas según la fórmula recursiva:
   f[x_i, ..., x_{i+j}] = (f[x_{i+1}, ..., x_{i+j}] - f[x_i, ..., x_{i+j-1}]) / (x_{i+j} - x_i).
4. Nombra las filas y columnas para mejorar la interpretación.

Salida: La tabla de diferencias divididas.

2.Funciones calculadoras del polinomio interpolador haciendo uso de la tabla de diferencias divididas.

A continuación se detallan diferentes funciones que se pueden emplear para obtener el polinomio interpolador y calcular su valor en cualquier punto x. La tabla de diferencias divididas es clave ya que permite obtener los **coeficientes canónicos**

Los coeficientes canónicos son los valores de las diferencias divididas que aparecen en la fórmula del polinomio de interpolación de Newton. Estos se calculan a partir de los datos iniciales (xi) y f[xi] y se extraen directamente de la tabla de diferencias divididas.

En la forma del polinomio de Newton:

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

Los coeficientes son:

- $f[x_0]$,
- $f[x_0, x_1]$,
- $f[x_0, x_1, x_2]$,
- $\ldots$,
- $f[x_0, x_1, \ldots, x_n]$.

Estos valores se encuentran en la primera fila de cada columna de la tabla de diferencias divididas.

Gracias a los coeficientes canónicos obtenidos mediante la tabla de diferencias divididas, es posible estimar el valor en un punto x

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

$$\begin{aligned} p_n(x) = \; & f[s_0] + f[s_0, s_1](x - s_0) + \\ & + f[s_0, s_1, s_2](x - s_0)\,(x - s_1) \\ & + f[s_0, s_1, s_2, s_3](x - s_0)\,(x - s_1)(x - s_2) + \cdots \\ & + f[s_0, s_1, s_2, s_3, s_4](x - s_0)\,(x - s_1)(x - s_2)(x - s_3) + \end{aligned}$$

2.1.“polinomio_newton_sin_funcion_interna”:

Código:

```
polinomio_newton_sin_funcion_interna <- function(tabla_dif_div, x_orig, x_eval) {
    n <- ncol(tabla_dif_div) - 1
    coef <- as.numeric(tabla_dif_div[1, ])

    # Inicializar un vector para almacenar los resultados
    resultados <- numeric(length(x_eval))

    for (k in seq_along(x_eval)) {
        x <- x_eval[k]
        resultado <- coef[1] # Término constante

        for (i in 1:n) {
            producto <- 1

            for (j in 1:i) {
                producto <- producto * (x - x_orig[j])
                }
        resultado <- resultado + coef[i + 1] * producto
            }
    resultados[k] <- resultado
    }
return(resultados)
}
```

Descripción: Evalúa el polinomio interpolador de Newton para un conjunto de puntos, sin utilizar funciones anidadas.

Pasos principales:

1. Extrae los coeficientes del polinomio desde la primera fila de la tabla de diferencias divididas.
2. Evalúa el polinomio punto a punto usando la fórmula del polinomio de Newton:
   $P(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + ...$
3. Los bucles anidados calculan los productos acumulados $(x - x_0)$, $(x - x_1)$, ...

Salida: Un vector con los valores evaluados.

2.2. "polinomio_newton_con_funcion_interna"

Código:

```
polinomio_newton_con_funcion_interna <- function(tabla_dif_div, x_orig, x_eval) {
        n <- ncol(tabla_dif_div) - 1
        coef <- as.numeric(tabla_dif_div[1, ])
        p <- function(x) {
                resultado <- coef[1]
                for (i in 1:n) {
                        producto <- 1
                        for (j in 1:i) {
                                producto <- producto * (x - x_orig[j])
                                }
                        resultado <- resultado + coef[i + 1] * producto
                }
return(resultado)
}
return(sapply(x_eval, p))
}
```

Descripción: Similar a la función anterior, pero encapsula el cálculo en una función interna “p(x)” para facilitar la reutilización.

Pasos principales:

1. Define una función interna "p(x)" que evalúa el polinomio en un punto "x".
2. Usa "sapply" para aplicar la función "p(x)" a todos los puntos de evaluación.

Ventaja: Mejora la modularidad y legibilidad del código.

Salida: Un vector con los valores evaluados.

2.3. “polinomio_newton”: función más optimizada

Código:

```
polinomio_newton <- function(tabla_dif_div, x_orig, x_eval) {
    n <- ncol(tabla_dif_div) - 1
    coef <- as.numeric(tabla_dif_div[1, ])
    p <- function(x) {
        resultado <- coef[1]
        producto <- 1
        for (i in 2:(n + 1)) {
            producto <- producto * (x - x_orig[i - 1])
            resultado <- resultado + coef[i] * producto
            }
    return(resultado)
    }
return(sapply(x_eval, p))
}
```

Descripción: Optimiza aún más la evaluación del polinomio al reducir la lógica de cálculo del producto acumulado. Es un código más complejo pero más optimizado

Pasos principales:

1. Utiliza una única variable "producto" para llevar el cálculo acumulado de (x - x_i).
2. Simplifica el bucle principal para mejorar el rendimiento.

Salida: Un vector con los valores evaluados.

2.4. "polinomio_newton_canonico": Función que devuelve el polinomio en forma canónica.

Código:

```
polinomio_newton_canonico <- function(tabla_dif_div, x_orig) {

    # Determinar el grado del polinomio

    n <- length(tabla_dif_div[1, ]) - 1


    # Extraer los coeficientes de Newton desde la primera

    #fila de la tabla de diferencias divididas

    coef_newton <- as.numeric(tabla_dif_div[1, ])


    # Inicializar el polinomio con el término constante

    pol <- c(coef_newton[1])

    prod_pol <- 1

    for (i in 2:(n+1)) {

        prod_pol = c(0,prod_pol)-c(prod_pol,0)*x_orig[i-1]

        pol=c(pol,0)+coef_newton[i]*prod_pol

    }

return(pol)

}
```

Descripción: Devuelve el polinomio interpolador de Newton en su forma canónica, como un vector de coeficientes.

Pasos principales:

1. Extrae los coeficientes desde la tabla de diferencias divididas.
2. Convierte el polinomio de la forma de Newton a la forma estándar (canónica):
   P(x) = a_0 + a_1x + a_2x^2 + ...
3. Usa aritmética de polinomios para construir el resultado paso a paso.

Salida: Un vector que representa el polinomio en su forma estándar.

- Los polinomios en forma canónica pueden estar representados mediante vectores, tales que:

a | a_0 | a_1 | ... | ... | a_n | Representa: $p_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$

- Multiplicar $p_n(x)$ por cx^k es equivalente **a desplazar el vector k veces a la izquierda** y **multiplicar todos sus coeficientes por c**:

k ceros

0	0	0	...	a_0c	...	ca_n

Representa: $p_n(x)cx^k = ca_0x^k + ca_1xx^k + c\,a_2x^2x^k + \cdots + c\,a_nx^n\,x^n = ca_0x^k + ca_1x^{k+1} + ca_2x^{k+2} + \cdots + ca_nx^{n+k}$

Después de esta operación, el grado del polinomio resultante es k + n , por lo que el vector tendrá k + n + 1 elementos.

- A su vez si $q_n(x)$ está representado por el vector d y $r_n(x) = p_n(x) + q_n(x)$ está representado por el vector e, entonces:

e | $a_0 + d_0$ | $a_1 + d_1$ | ... | ... | $a_n + d_n$ | Representa: $r_n(x)$