INTERPOLACIÓN

INDICE

¿Qué es?

La interpolación es un método que nos permite saber valores de una gráfica que desconocemos. La interpolación se basa en una serie de puntos, cuya posición conocemos. Con esos puntos, mediante una serie de operaciones, hacemos una gráfica que nos da un valor aproximado del punto que queramos.

Hay muchos tipos diferentes de interpolación, aunque todos llevan a lo mismo; a una función. Algunos de los tipos de interpolación son la polinómica, la lineal, la polinómica de Hermite…

Sin embargo, nosotros nos vamos a centrar en la interpolación de Lagrange, la de Newton y la interpolación a trozos.

Hay que tener en cuenta que cada método de interpolación es usado con diferentes fines, ya que dependiendo de los valores que busquemos y la interpolación que usamos, los valores estarán mejor o peor aproximados.

Interpolación de Lagrange

La interpolación de Lagrange es un método usado para aproximar una función desconocida mediante puntos conocidos. Esto nos permite "predecir" la posición de un punto cualquiera. La función de Lagrange se define mediante la siguiente operación:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_1)$$

Donde:

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

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

Así de primeras parece algo muy complejo, pero si lo miramos detalladamente podemos dividirlo en partes más sencillas. Vemos que $\mathcal{L}$[i] es un productorio, donde x es el punto en el que buscamos la función y, x[i] y x[j] son los puntos del soporte. $\mathcal{F}$[n] es un sumatorio de $\mathcal{L}$[i] multiplicado por el valor de la función en cada [i].

En el código, primero tendremos que formar un productorio, que almacenaremos en un vector. Esto lo conseguimos mediante un bucle "for".

```
for (i in 1:n){
    L[i] = 1
    for (j in 1:n){
      if (i!=j){
        L[i] = L[i] * (t-s[j])/(s[i]-s[j])
      }
    }
  }
```

En este código, n es la longitud del vector s; que es la cantidad de veces que se repetirá el bucle for. El vector s es un vector que contiene los puntos de interpolación, es decir, las "x" de una función. Por último, t es el valor interpolado, es decir, el valor de "x" del cual queremos saber la "y". El productorio se guarda en el vector L[i]. Como se trata de un productorio, L[i] tiene que ser igualado a uno, para que funcione bien

Una vez obtenido este vector, tan solo nos queda multiplicar cada valor del vector por los puntos del vector f(x) (los valores "y" asociados a cada valor del soporte, "x"). En código, esto se vería de la siguiente manera:

```
P = 0
  for (i in 1:n){
    P = P + B[i]*L[i]
  }
```

Esta parte es la formación del polinomio, que se trata de un sumatorio. Al ser un sumatorio, hay que marcar un P = 0, para que el sumatorio funcione. El valor de n es el mismo que antes, la longitud del vector del soporte. El vector B[i] es el vector que contiene los valores "y" de la función. L[i] es el productorio que calculamos en el apartado anterior.

El valor obtenido se almacena en P, siendo este el valor que buscábamos para nuestra t.

**Para este código, los datos de entrada serían t, B, s, y n (n = length(s)).

Interpolación de Newton

La interpolación de Newton es otro tipo de interpolación. Similar a la de Lagrange, esto nos permite "predecir" valores que no conocemos mediante una serie de operaciones. La fórmula completa es la siguiente.

$$P = f[S_1] + \sum_{I=2}^{n}(f[S_1, S_2, \ldots, S_i] * \prod_{J=1}^{i-1}(t-s_j))$$

Como se puede observar, la fórmula es similar a la de Lagrange. Presenta un productorio dentro de un sumatorio.

Lo primero que hay que hacer en esta interpolación es crear una matriz, en la cual contendremos unos valores que serán la pendiente entre puntos. La matriz corresponde a la primera parte del sumatorio. La matriz tendrá una estructura similar a la tabla siguiente, siendo una matriz diagonal superior invertida.

x_k	$f[x_k]$	$f[x_k, x_{k+1}]$	$f[x_k, x_{k+1}, x_{k+2}]$	$f[x_k, x_{k+1}, x_{k+2}, x_{k+3}]$
1	$f[1] = 2$	$f[1,3] = \frac{3-2}{3-1} = \frac{1}{2}$	$f[1,3,4] = \frac{-1-\frac{1}{2}}{4-1} = -\frac{1}{2}$	$f[1,3,4,8] = \frac{-\frac{1}{2}-\frac{3}{5}}{8-1} = \frac{11}{70}$
3	$f[3] = 3$	$f[3,4] = \frac{2-3}{4-3} = -1$	$f[3,4,8] = \frac{2+1}{8-3} = \frac{3}{5}$	
4	$f[4] = 2$	$f[4,8] = \frac{10-2}{8-4} = 2$		
8	$f[8] = 10$			

Como se puede observar, en cada casilla, restamos los valores de "y" (f[x]) y los dividimos por la resta de los valores de "x". De manera más representativa, estas son las operaciones que se están llevando a cabo:

- $f_1(x_0, x_1) = \dfrac{f_0(x_1) - f_0(x_0)}{x_1 - x_0}$
- $f_2(x_0, x_1, x_2) = \dfrac{f_1(x_1, x_2) - f_1(x_0, x_1)}{x_2 - x_0}$

Para el último valor, el enésimo, la formula sería la siguiente:

$$f_i(x_0, x_1, \ldots, x_{i-1}, x_i) = \frac{f_{i-1}(x_1, \ldots, x_{i-1}, x_i) - f_{i-1}(x_0, x_1, \ldots, x_{i-1})}{x_i - x_0},$$

En código, esto lo conseguimos mediante la realización de una matriz. La matriz que usaremos constará de n filas y n columnas. La cantidad (n) de columnas y filas está determinada por la longitud del vector de puntos del soporte. El código de la matriz sería el siguiente:

```
A=matrix( c(0), nrow=n, ncol=n)
```

Una vez obtenida la matriz, tan solo tenemos que rellenarla tal y como vimos antes. La primera fila, contiene los valores del vector f(x), y el resto de los valores son las diferencias, como visto antes. El código para esta matriz es muy sencillo, siendo el siguiente:

```
for (j in 1:n){
        for (i in 1:(n-j+1)){
                if (j == 1){
                        A[i,j] = y[i]
                } else{
                A[i,j] = (A[i+1,j-1]-A[i,j-1])/(s[i+j-1]-s[i])
                }
        }
}
```

Como se puede observar en el código, este consta de un bucle "if" dentro de un bucle "for". En el bucle "if", ponemos la condición de j==1 (se usa doble "=", ya que si no se estaría otorgando un valor), con la finalidad de que la primera columna sean los valores de "y". En la parte "else" del bucle, otorgamos al resto de valores las diferencias. Así construimos la matriz A, que contiene toda la información.

Una vez obtenida esta matriz, procedemos a construir el productorio y el sumatorio. Esto lo podemos hacer con una serie de bucles anidados. El código es el siguiente:

```
q = 1
g = 0
for (i in 2:n){
        for (j in 1:(i-1)){
                q = q*(t-s[j])
        }
        g = g + A[1,i]*q
}
```

En este código, necesitamos asignar esos valores a q y g, ya que son necesarios para formar el productorio y el sumatorio. Asimismo, hay que tener en cuenta que i va a comenzar en 2, ya que el valor A[1,1] se añade a parte. El productorio se guarda en q, y el sumatorio se guarda en g. Revisando la fórmula general, lo único que nos falta es sumar el valor de f[S_1], y obtendríamos finalmente en polinomio completo.

P = A[1,1] + g

**Los datos de entrada de esta interpolación serían y, s, t y n (n=length(s)).

Interpolación por tramos

La interpolación por tramos se utiliza para determinadas funciones, que, al presentar un grado elevado, su aproximación mediante Lagrange o Newton sería muy inexacta. Si no usáramos la interpolación por tramos, el error sería muy grande, y los valores no podrían ser tomados.

La interpolación por tramos consiste en dividir la función en diferentes puntos, y crear una aproximación de grado bajo (1) entre ellos. Dependiendo del valor que busquemos (t), la aproximación que usaríamos sería diferente. Para ello, creamos un conjunto de funciones, y utilizamos la necesaria dependiendo del punto.

$$U(x) \begin{cases} P_1(x) \text{ si } x \in [S_0, S_1] \\ P_2(x) \text{ si } x \in [S_1, S_2] \\ P_3(X) \text{ si } x \in [S_2, S_3] \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ P_i(x) \text{ si } x \in [S_{i-1}, S_i] \end{cases}$$

Para interpolar cada uno de estos polinomios, se usa el método de funciones de base, que es el siguiente:

$$U(x)=\sum f_i \varphi_i(x)$$

En cada uno de los subintervalos (φ_i), la fórmula general que lo define es la siguiente: x_i

$$\varphi_i \begin{cases} \dfrac{t-x[i-1]}{x[i]-x[i-1]} \text{ si } x \in [S_{i-1}, S_i] \\ \dfrac{t-x[i+1]}{x[i]-x[i+1]} \text{ si } x \in [S_i, S_{i+1}] \\ 0 \text{ si } x \in [S_0, S_{i-1}] \cup [S_{i+1}, S_n] \end{cases}$$

En el caso concreto de φ_0 y φ_n, la fórmula varía, ya que la primera posición no tiene punto anterior; y la posición final no tiene punto siguiente.

φ_0 $\begin{cases} \frac{t-x[1]}{x[0]-x[1]} & \text{si x ε } [S_0, S_1] \\ 0 & \text{si x ε } [S_1, S_n] \end{cases}$ φ_n $\begin{cases} \frac{t-x[n-1]}{x[n]-x[n-1]} & \text{si x ε } [S_{n-1}, S_n] \\ 0 & \text{si x ε } [S_0, S_{n-1}] \end{cases}$

Como cada subintervalo consta de 2 operaciones distintas de 0, lo mas sencillo es colocar cada valor en una casilla de una matriz con 2 columnas y n filas.

```
f=matrix( c(0), nrow=n, ncol=2)
```

En este caso, el valor de n es igual que siempre, la longitud del vector con los puntos de soporte.

Una vez obtenida esta matriz, le colocamos los valores que no siguen la fórmula general. Estos valores son el primero y el último.

```
f[1,1]= (t-s[2])/(s[1]-s[2])

f[n,1]= (t-s[n-1])/(s[n]-s[n-1])
```

Tras esto, tan solo nos falta colocar el resto de valores, y así obtendremos la matriz con todos los valores que necesitamos para nuestros cálculos.

```
for (i in 2:m){
        f[i,1]= (t-s[i-1])/(s[i]-s[i-1])
        f[i,2]= (t-s[i+1])/(s[i]-s[i+1])
}
```

Como podemos ver, recurrimos a un bucle “for”, donde la i varia de 2 a m. Esto se debe a que la primera posición ya está rellenada, así como la n. El valor de m es de n-1.

Una vez obtenemos la matriz, tan solo nos falta ver a que intervalo corresponde nuestro punto t para ver que valores de la matriz tendremos que utilizar. Esto se hace mediante un sencillo código de bucles “if”, tal que así.

```
for (i in 3:m){
        if (t < s[1]){
                u = "No se"
        } else{
                if (t < s[2]){
                        u = y[1]*f[1,1] + y[2]*f[2,1]
                } else{
                        if (t < s[i]){
                                u = y[i-1]*f[i-1,2] + y[i]*f[i,1]
                        } else{
                                u = y[m]*f[m,2] + y[n]*f[n,1]
                        }
                }
        }
}
```

La i la ponemos que varíe de 3 a m, poniendo nosotros los primeros 2 valores de manera manual. Esto lo hice con el fin de que se entendiera el proceso que está siguiendo el programa. Al llegar al i=m, el programa se pregunta si t<s[m]. Si esto es falso, significa que tiene que ser mayor q este, por lo que directamente procede a hacer el último cálculo.

**Los datos de entrada de esta interpolación son y, s, t, n (n = length(s)) y m (m = n-1).