MA0501 – Tarea 4

Diego Alberto Vega Víquez - C38367 José Carlos Quintero Cedeño - C26152
Gabriel Valverde Guzmán - C38060

2025-09-23

Tabla de contenidos

Ejercicio 1

Instrucción del ejercicio 1

Complete las demostraciones que quedaron pendientes en la clase.

Solución

Teorema 1 [Weierstrass]

Sea f continua en $[a, b]$; entonces dado $\varepsilon > 0$, existe $n \in \mathbb{N}$ y $P_n(x) \in P_n$ tal que

$$|f(x) - P_n(x)| < \varepsilon, \quad \forall x \in [a, b].$$

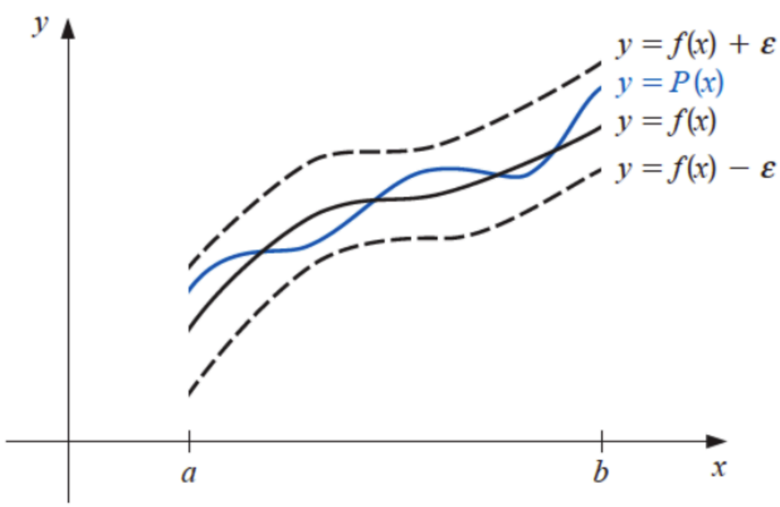


Figura 1: Polinomio de Bernstein

Prueba

Sin pérdida de generalidad, suponga que $a = 0$ y $b = 1$. Sea:

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k} f\left(\frac{k}{n}\right),$$

se puede probar que $B_n(x) \to f(x)$ uniformemente en $[0, 1]$:
Vea que

$$|B_n(x) - f(x)| \leq \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k} \left| f\left(\frac{k}{n}\right) - f(x) \right|$$

Note que se tiene una esperanza con respecto a una distribución binomial $\mathrm{Bin}(n, x)$

$$X_n \sim \mathrm{Bin}(n, x) \implies \mathbb{E}\left[f\left(\frac{X_n}{n}\right)\right] = B_n(x)$$

Entonces:

$$|B_n(x) - f(x)| = \left| \mathbb{E}\left[f\left(\tfrac{X_n}{n}\right)\right] - f(x) \right|$$

Dado que f es continua en $[0, 1]$, por el Teorema de Heine–Cantor, también es uniformemente continua.
Entonces, para todo $\varepsilon > 0$, existe $\delta > 0$ tal que:

$$|x - y| < \delta \quad \Rightarrow \quad |f(x) - f(y)| < \varepsilon$$

Considere dividir el dominio del índice k en dos regiones:

- Cuando $\left|\frac{k}{n} - x\right| < \delta$
- Cuando $\left|\frac{k}{n} - x\right| \geq \delta$

Entonces:

$$|B_n(x) - f(x)| \leq \varepsilon \cdot P\left(\left|\tfrac{X_n}{n} - x\right| < \delta\right) + 2\|f\|_\infty \cdot P\left(\left|\tfrac{X_n}{n} - x\right| \geq \delta\right)$$

Usando la *desigualdad de Chebyshev* aplicada a $\frac{X_n}{n}$, cuya varianza es $\frac{x(1-x)}{n}$, el segundo término tiende a 0 uniformemente en $x \in [0, 1]$.
Como el error puede hacerse arbitrariamente pequeño uniformemente en x, se concluye:

$$\sup_{x \in [0,1]} |B_n(x) - f(x)| \longrightarrow 0 \qquad \text{cuando } n \to \infty.$$

■

Teorema 3

Sea $f \in C[a,b]$ y $f \in C^n]a,b[$, si f se anula en $n+1$ puntos distintos, $x_0, x_1, \dots, x_n$ en $[a,b]$. Entonces $\exists\, c \in]a,b[$ tal que $f^{(n)}(c) = 0$.

Prueba

Por inducción sobre n

Caso base: $n = 1$.

Si $f(x_0) = f(x_1) = 0$ con $x_0 < x_1$, entonces por el Teorema de Rolle clásico, existe $c \in (x_0, x_1)$ tal que:

$$f'(c) = 0.$$

Hipótesis de inducción:

Supongamos que el resultado es cierto para $n = k$, es decir, si $f \in C^k]a,b[$ y se anula en $k+1$ puntos distintos en $[a,b]$, entonces existe $c \in (a,b)$ tal que $f^{(k)}(c) = 0$.

Paso inductivo:

Sea ahora $f \in C^{k+1}]a,b[$ y supongamos que $f(x_0) = f(x_1) = \cdots = f(x_{k+1}) = 0$, con $x_0 < x_1 < \cdots < x_{k+1}$.

Por el Teorema de Rolle, para cada par consecutivo (x_i, x_{i+1}), existe $c_i \in (x_i, x_{i+1})$ tal que:

$$f'(c_i) = 0.$$

Eso da una nueva colección de $k+1$ puntos $c_0 < c_1 < \cdots < c_k$ donde f' se anula. Entonces, por la hipótesis de inducción aplicada a f', existe un punto $c \in (a,b)$ tal que:

$$f^{(k+1)}(c) = 0.$$

■

Teorema 6

Si $P_n(x)$ es el polinomio de Lagrange que coincide con $f(x)$ en $x_0, x_1, \dots, x_n$, entonces:

$$
\begin{aligned}
P_n(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\
&\quad + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\
&= f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k](x - x_0)\cdots(x - x_{k-1}).
\end{aligned}
$$

Prueba

Si $P_n(x)$ se escribe de la forma

$$\begin{aligned}P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots \\ + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}),\end{aligned}$$

entonces $P_n(x_0) = a_0$. Como $P_n(x_0) = f(x_0)$ se sigue que
$a_0 = f(x_0) = f[x_0]$.
Además:

$$\begin{aligned}P_n(x_1) &= a_0 + a_1(x_1 - x_0), \\ f(x_1) &= f[x_0] + a_1(x_1 - x_0) \quad \Rightarrow \quad a_1 = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1].\end{aligned}$$

Luego, por inducción se puede probar fácilmente que $a_k = f[x_0, x_1, \ldots, x_k]$.
Paso base:
Para $k = 0$, se tiene:

$$P_n(x_0) = a_0 = f(x_0),$$

por lo tanto:

$$a_0 = f[x_0].$$

Hipótesis de inducción:
Supongamos que para algún $k \geq 1$ se cumple que:

$$a_j = f[x_0, x_1, \ldots, x_j] \quad \text{para todo } j = 0, 1, \ldots, k-1.$$

Paso inductivo:
Queremos probar que:

$$a_k = f[x_0, x_1, \ldots, x_k].$$

Evaluamos el polinomio $P_n(x)$ en $x = x_k$. Por construcción, se cumple:

$$\begin{aligned}P_n(x_k) = a_0 + a_1(x_k - x_0) + a_2(x_k - x_0)(x_k - x_1) + \cdots \\ + a_k(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1}).\end{aligned}$$

Pero también sabemos que $P_n(x_k) = f(x_k)$, y por la hipótesis de inducción:

Teorema 7

- Sea $f \in C[a,b]$.
- Sean $x_0, x_1, \ldots, x_n$, $(n+1)$ nodos distintos en $[a,b]$.

Entonces el polinomio de grado menor que coincide con f y f' en $x_0, x_1, \ldots, x_n$:

- Tiene grado $2n+1$.
- Está dado por

$$H_{2n+1}(x) = \sum_{j=0}^{n} f(x_j) H_{nj}(x) + \sum_{j=0}^{n} f'(x_j) \widetilde{H}_{nj}(x),$$

donde

$$H_{nj}(x) = [1 - 2(x - x_j) L'_{nj}(x_j)] L^2_{nj}(x),$$

y

$$\widetilde{H}_{nj}(x) = (x - x_j) L^2_{nj}(x).$$

- Además, el error absoluto es:

$$|f(x) - H_{2n+1}(x)| = \left| \frac{(x-x_0)^2 \cdots (x-x_n)^2}{(2n+2)!} f^{(2n+2)}(\xi) \right|, \quad \text{con } \xi \in]a,b[.$$

Prueba

- Se debe demostrar que $H_{2n+1}(x_i) = f(x_i)$ para todo $i = 0, 1, \ldots, n$. Para ver esto, recordemos que:

$$L_{nj}(x_i) = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

de donde, cuando $i \neq j$:

$$H_{nj}(x_i) = 0 \quad \text{y} \quad \widetilde{H}_{nj}(x_i) = 0.$$

Mientras que:

$$H_{ni}(x_i) = [1 - 2(x_i - x_i)L'_{ni}(x_i)] \cdot 1 = 1,$$

$$\widetilde{H}_{ni}(x_i) = (x_i - x_i) \cdot 1^2$$

Luego:

$$H_{2n+1}(x_i) = \sum_{\substack{j=0 \\ j\neq i}}^{n} f(x_j) \cdot 0 + f(x_i) \cdot 1 + \sum_{j=0}^{n} f'(x_j) \cdot 0 = f(x_i).$$

Por lo tanto:

- $H_{2n+1}(x_i) = f(x_i)$ para $i = 0, 1, 2, \ldots, n$.
- Se debe demostrar que $H'_{2n+1}(x_i) = f'(x_i)$ para todo $i = 0, 1, \ldots, n$.

Nótese que $L_{nj}(x)$ es un factor de $H'_{nj}(x)$, lo cual implica que $H'_{nj}(x_i) = 0$ cuando $i \neq j$. Además, si $i = j$:

$$\begin{aligned} H'_{ni}(x_i) &= -2L'_{ni}(x_i)L^2_{ni}(x_i) + [1 - 2(x_i - x_i)L'_{ni}(x_i)] \cdot 2 \cdot L_{ni}(x_i)L'_{ni}(x_i) \\ &= -2L'_{ni}(x_i) + 2L'_{ni}(x_i) \\ &= 0. \end{aligned}$$

Por lo tanto, $H'_{nj}(x_i) = 0$ para todo $i = 0, 1, 2, \ldots, n$ y para todo $j = 0, 1, 2, \ldots, n$. Además:

$$\widetilde{H}_{nj}(x_i) = L^2_{nj}(x_i) + (x_i - x_j)L'_{nj}(x_j) \cdot 2 \cdot L_{nj}(x_i)L'_{nj}(x_i),$$

de donde:

$$\widetilde{H}'_{nj}(x_i) = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

Lema 1

Si $f \in C^n[a,b]$ y $x_0, x_1, \ldots, x_n$ son los $(n+1)$ nodos distintos en $[a,b]$, entonces:
existe $\xi \in]a,b[$ tal que:

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

Prueba

Consideremos el polinomio interpolador de Newton de grado n, que interpola a f en los nodos $x_0, \dots, x_n$:

$$P_n(x) = \sum_{k=0}^{n} f[x_0, \dots, x_k] \prod_{j=0}^{k-1} (x - x_j).$$

Sea el **polinomio del error**:

$$R_n(x) = f(x) - P_n(x).$$

Sabemos por teoría del error de interpolación que:

$$R_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j),$$

para algún ξ_x entre el menor y el mayor de los puntos x_j y x.
Ahora consideremos la función:

$$F(x) = f(x) - Q(x),$$

donde $Q(x)$ es el polinomio de grado $n-1$ tal que $Q(x_i) = f(x_i)$ para $i = 0, \dots, n-1$. Entonces $F(x)$ es una función que coincide con f en los primeros n nodos, pero no necesariamente en el último.
Aplicamos el **Teorema de Rolle generalizado** a la función:

$$g(x) = f(x) - P_n(x),$$

la cual se anula en los puntos $x_0, \dots, x_n$, ya que $P_n(x_j) = f(x_j)$. Como $f \in C^n[a, b]$, entonces $g \in C^n[a, b]$ y se anula en $n + 1$ puntos.
Por el teorema, existe $\xi \in (a, b)$ tal que:

$$g^{(n)}(\xi) = 0.$$

Pero como $P_n(x)$ es de grado n, su derivada de orden n es constante:

$$P_n^{(n)}(x) = n! \cdot f[x_0, x_1, \dots, x_n],$$

y:

$$g^{(n)}(x) = f^{(n)}(x) - P_n^{(n)}(x) = f^{(n)}(x) - n! \cdot f[x_0, \dots, x_n].$$

Entonces, si $g^{(n)}(\xi) = 0$:

$$f^{(n)}(\xi) = n! \cdot f[x_0, \dots, x_n] \quad \Rightarrow \quad f[x_0, \dots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

Ejercicio 2

Instrucción del ejercicio 2

Implemente en R los algoritmos de interpolación polinómica vistos en clase.

Los métodos que se deben implementar son:

a) Interpolar $f(x)$ usando el polinomio de Lagrange con el algoritmo de Neville.
b) Interpolar $f(x)$ usando el polinomio de Lagrange con el algoritmo de diferencias divididas de Newton.
c) Interpolar $f(x)$ usando el polinomio de Hermite con el algoritmo de diferencias divididas de Newton.
d) Interpolar $f(x)$ usando el "Splines" cúbicos naturales y sujetos.

Luego en general programe una función que permita graficar el polinomio de interpolación y la función correspondiente (si la hay).

Solución Neville

```r
# Neville: interpolacion en un punto x a partir de un vector de nodos (xi) y otro de valores (f

# Retorna una lista con el valor interpolado y la matriz completa Q (la que posee todos los pol

neville <- function(nodos, valores, x) {
  stopifnot(is.numeric(nodos),
            is.numeric(valores),
            length(nodos) == length(valores))
  n <- length(nodos)
  Q <- matrix(NA_real_, nrow = n, ncol = n)

  # Columna inicial con valores de Y
  Q[, 1] <-  valores

  # Construccion de la tabla de Neville
  for (i in 2:n) {
    for (j in 2:i) {
      numerador <- ((x - nodos[i - j + 1]) * Q[i, j - 1] - (x - nodos[i]) * Q[i -
                                                                               1, j - 1])
      denominador <- nodos[i] - nodos[i - j + 1]
      Q[i, j] <-  numerador / denominador
```

```
    }
  }
  return(list(valor = Q[n, n], tabla = Q))
}
```

Lagrange por diferencias divididas de Newton

```
# Lagrange.Newton: interpolacion en un punto x function# Lagrange.Newton: interpolacion en un p

# Retorna una lista con el valor interpolado, la matriz para la construccion de los coeficiente

lagrange.newton <- function(nodos, valores, x) {
  stopifnot(is.numeric(nodos),
            is.numeric(valores),
            length(nodos) == length(valores))
  n <- length(nodos)
  Q <- matrix(NA_real_, nrow = n, ncol = n)

  Q[, 1] <- valores

  # Construccion de la tabla de diferencias de Newton
  for (i in 2:n) {
    for (j in 2:i) {
      numerador <- Q[i, j - 1] - Q[i - 1, j - 1]
      denominador <- nodos[i] - nodos[i - j + 1]
      Q[i, j] <-  numerador / denominador
    }
  }
  coeficientes <- diag(Q)

  valor = Q[1, 1]
  producto = 1

  for (i in 2:n) {
    producto <- producto * (x - nodos[i - 1])
    valor <- valor + coeficientes[i] * producto
  }
  return(list(
```

```
    valor = valor,
    tabla = Q,
    coeficientes = coeficientes
  ))
}
```

Hermite por diferencias divididas de Newton

```
# Hermite.newton: interpolacion en un punto x a partir de un vector de nodos, otro de valores y

# Retorna una lista con el valor interpolado, la matriz para la construccion de los  y un vecto

hermite.newton <- function(nodos, valores, derivadas, x) {
  stopifnot(
    is.numeric(nodos),
    is.numeric(valores),
    is.numeric(derivadas),
    length(nodos) == length(valores),
    length(nodos) == length(derivadas)
  )
  n <- length(nodos)

  Z <- numeric(2 * n)
  Q <- matrix(0, nrow = 2 * n, ncol = 2 * n)



  # Set-up inicial de la matriz
  for (i in 1:n) {
    z0 <- 2 * i - 1
    z1 <- 2 * i
    Z[z0] <- nodos[i]
    Z[z1] <- nodos[i]
    Q[z0, 1] <- valores[i]
    Q[z1, 1] <- valores[i]
    Q[z1, 2] <- derivadas[i]
    if (i != 1) {
      Q[z0, 2] <- (Q[z0, 1] - Q[z0 - 1, 1]) / (Z[z0] - Z[z0 - 1])
    }
```

```
  }
  # Rellenar el resto de la matriz a partir de estos valores
  for (i in 3:(2 * n)) {
    for (j in 3:i) {
      Q[i, j] <- (Q[i, j - 1] - Q[i - 1, j - 1]) / (Z[i] - Z[i - j + 1])
    }
  }

  coeficientes <- diag(Q)

  valor <- Q[1, 1]
  producto <- 1

  for (i in 2:(2 * n)) {
    producto <- producto * (x - Z[i - 1])
    valor <- valor + coeficientes[i] * producto
  }
  return(list(
    valor = valor,
    tabla = Q,
    coeficientes = coeficientes
  ))
}
```

Splines cubicos naturales

```
# spline.natural: funcion que calcula la interpolacion por splines a partir de unos nodos y sus

# Retorna los valores de los coeficientes a, b, c, d de cada una de las n-1 ecuaciones generada
spline.natural <- function(nodos, valores, x) {
  # Note que los valores de a corresponden a los valores de los nodos en la funcion, por lo que
  n <- length(nodos)

  h <- numeric(n - 1)
  alfa <- numeric(n - 1)
  alfa[1] <- 0

  # Paso 1 y 2: definir los h's y alfas
```

```
for (i in 1:(n - 1)) {
  h[i] <- nodos[i + 1] - nodos[i]
  if (i != 1) {
    alfa[i] <- (3 / h[i]) * (valores[i + 1] - valores[i]) - (3 / h[i - 1]) *
      (valores[i] - valores[i - 1])
  }
}

#Paso 3: definir valores iniciales de l, m, y z
l <- numeric(n) # creo que el tamano de esto puede ser n-1
l[1] <- 1
m <- numeric(n - 1)
m[1] <- 0
z <- numeric(n) # creo que el tamano de esto puede ser n-1
z[1] <- 0

# Paso 4: rellenar vectores l, m, z
for (i in 2:(n - 1)) {
  l[i] <- 2 * (nodos[i + 1] - nodos[i - 1]) - h[i - 1] * m[i - 1]
  m[i] <- h[i] / l[i]
  z[i] <- (alfa[i] - h[i - 1] * z[i - 1]) / l[i]
}

# Paso 5: definir valores finales
l[n] <- 1 #creo que esto no hace falta
z[n] <- 0 #esto tampoco
c <- numeric(n)
c[n] <- 0
b <- numeric(n)
d <- numeric(n)

# Paso 6: sustitucion hacia atras
for (j in (n - 1):1) {
  c[j] <- z[j] - m[j] * c[j + 1]
  b[j] <- (valores[j + 1] - valores[j]) / h[j] - (h[j] / 3) * (c[j + 1] +
                                                                2 * c[j])
```

```
    d[j] <- (c[j + 1] - c[j]) / (3 * h[j])
  }

  # Paso extra: evaluar la interpolacion en el punto x especificado

  ## Encontramos los dos nodos que estan prensando al intervalo
  indice <- NULL
  for (i in 1:(n - 1)) {
    if (x >= nodos[i] && x < nodos[i + 1]) {
      indice <- i
    }
  }
  if(x == nodos[n]){
    indice <- n
  }
  if (is.null(indice)) {
    return("El valor de interpolacion debe estar entre dos nodos")
  }
  ## evaluamos en la funcion asociada

  valor <- valores[indice] + b[indice] * (x - nodos[indice]) + c[indice] * (x - nodos[indice])^

  return(list(
    a = valores,
    b = b,
    c = c,
    d = d,
    valor = valor
  ))
}
```

Splines cubicos sujetos

```
# spline.sujeto: funcion que calcula la interpolacion por splines a partir de unos nodos y sus

# Retorna los valores de los coeficientes a, b, c, d de cada una de las n-1 ecuaciones generada
spline.sujeto <- function(nodos, valores, derivadas, x) {
  # Note que los valores de a corresponden a los valores de los nodos en la funcion, por lo que
```

```r
stopifnot(length(derivadas) == 2)
n <- length(nodos)

h <- numeric(n - 1)
alfa <- numeric(n)

# Paso 1 y 2: definir los h's y alfas
for (i in 1:(n - 1)) {
  h[i] <- nodos[i + 1] - nodos[i]
  if (i != 1) {
    alfa[i] <- (3 / h[i]) * (valores[i + 1] - valores[i]) - (3 / h[i - 1]) *
      (valores[i] - valores[i - 1])
  }
}

alfa[1] <- 3 * ((valores[2] - valores[1]) / h[1] - derivadas[1])
alfa[n] <- 3 * (derivadas[2] - (valores[n] - valores[n - 1]) / h[n - 1])

#Paso 3: definir valores iniciales de l, m, y z
l <- numeric(n) # creo que el tamano de esto puede ser n-1
l[1] <- 2 * h[1]
m <- numeric(n - 1)
m[1] <- 1 / 2
z <- numeric(n) # creo que el tamano de esto puede ser n-1
z[1] <- alfa[1] / l[1]

# Paso 4: rellenar vectores l, m, z
for (i in 2:(n - 1)) {
  l[i] <- 2 * (nodos[i + 1] - nodos[i - 1]) - h[i - 1] * m[i - 1]
  m[i] <- h[i] / l[i]
  z[i] <- (alfa[i] - h[i - 1] * z[i - 1]) / l[i]
}

# Paso 5: definir valores finales
l[n] <- h[n - 1] * (2 - m[n - 1])
z[n] <- (alfa[n] - h[n - 1] * z[n - 1]) / l[n]
```

```
c <- numeric(n)
c[n] <- z[n]
b <- numeric(n)
d <- numeric(n)

# Paso 6: sustitucion hacia atras
for (j in (n - 1):1) {
  c[j] <- z[j] - m[j] * c[j + 1]
  b[j] <- (valores[j + 1] - valores[j]) / h[j] - (h[j] / 3) * (c[j + 1] + 2 * c[j])
  d[j] <- (c[j + 1] - c[j]) / (3 * h[j])
}

# Paso extra: evaluar la interpolacion en el punto x especificado

## Encontramos los dos nodos que estan prensando al intervalo
indice <- NULL
for (i in 1:(n - 1)) {
  if (x >= nodos[i] && x < nodos[i + 1]) {
    indice <- i
  }
}
if (x == nodos[n]) {
  indice <- n
}
if (is.null(indice)) {
  return("El valor de interpolacion debe estar entre dos nodos")
}
## evaluamos en la funcion asociada

valor <- valores[indice] + b[indice] * (x - nodos[indice]) + c[indice] * (x - nodos[indice])^

return(list(
  a = valores,
  b = b,
  c = c,
  d = d,
```

```
    valor = valor
  ))
}
```

Funcion de graficacion

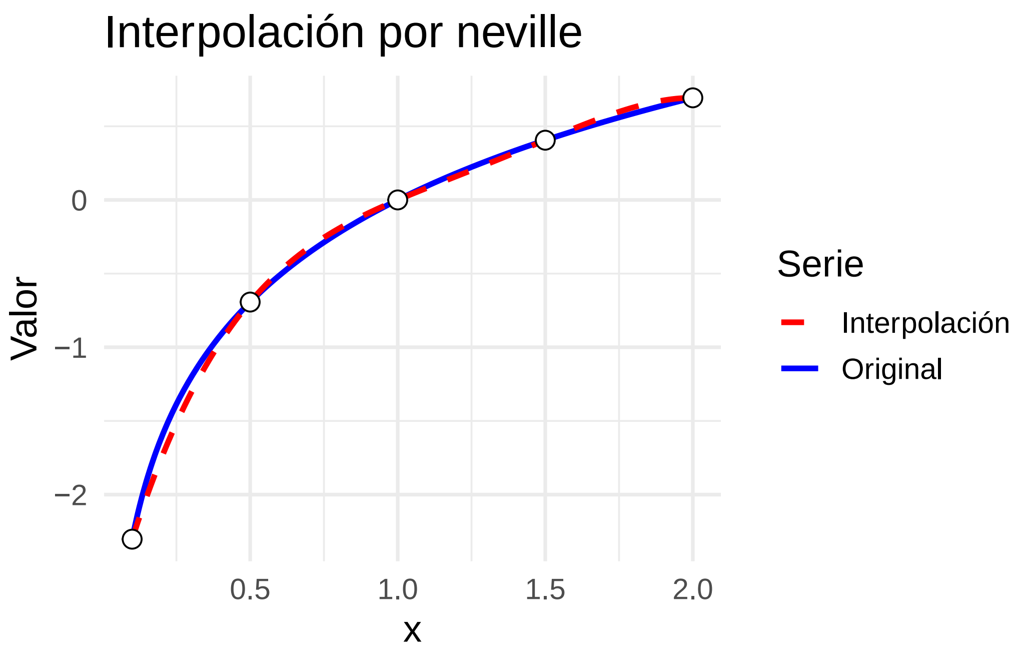
```
library(tidyverse)
graficar.polinomio <- function(nodos,
                               a,
                               b,
                               metodo,
                               f = NULL,
                               valores = NULL,
                               df = NULL,
                               derivadas.clamped = NULL) {
  stopifnot(is.numeric(nodos), length(nodos) >= 2)
  # Validación: exactamente uno de f o valores
  if (is.null(f) == is.null(valores)) {
    stop("Debe proveer exactamente uno: 'f' (función) o 'valores' (numérico).")
  }
  # Obtener valores en nodos según el caso
  if (!is.null(f)) {
    stopifnot(is.function(f))
    valores_nodos <- f(nodos)
  } else {
    stopifnot(is.numeric(valores), length(valores) == length(nodos))
    valores_nodos <- valores
  }

  # Derivadas (opcional). Acepta función o vector numérico.
  derivadas_nodos <- NULL
  if (!is.null(df)) {
    if (is.function(df)) {
      derivadas_nodos <- df(nodos)
    } else if (is.numeric(df)) {
      stopifnot(length(df) == length(nodos))
      derivadas_nodos <- df
    } else {
```

```r
    stop("`df` debe ser función o vector numérico de derivadas en los nodos.")
  }
}

# Wrapper vectorizado para el método (con o sin derivadas)
if (!is.null(derivadas_nodos)) {
  H <- function(x)
    vapply(x, function(xx)
      metodo(nodos, valores_nodos, derivadas_nodos, xx)$valor, numeric(1))
} else {
  if(is.null(derivadas.clamped)){
    H <- function(x)
      vapply(x, function(xx)
        metodo(nodos, valores_nodos, xx)$valor, numeric(1))
  } else{
    H <- function(x)
      vapply(x, function(xx)
        spline.sujeto(nodos, valores_nodos, derivadas.clamped, xx)$valor, numeric(1))
  }
}

# Malla y data frames
xi <- seq(a, b, length.out = 400)
df_plot <- data.frame(x  = xi,
                      Hx = H(xi),
                      fx = if (!is.null(f))
                        f(xi)
                      else
                        NA_real_)
df_nodos <- data.frame(x = nodos, y = valores_nodos)

# Gráfico: con f (dos curvas) o solo interpolación
p <- ggplot(df_plot, aes(x = x))
if (!is.null(f)) {
  p <- p +
    geom_line(aes(y = fx, color = "Original"), linewidth = 1) +
```

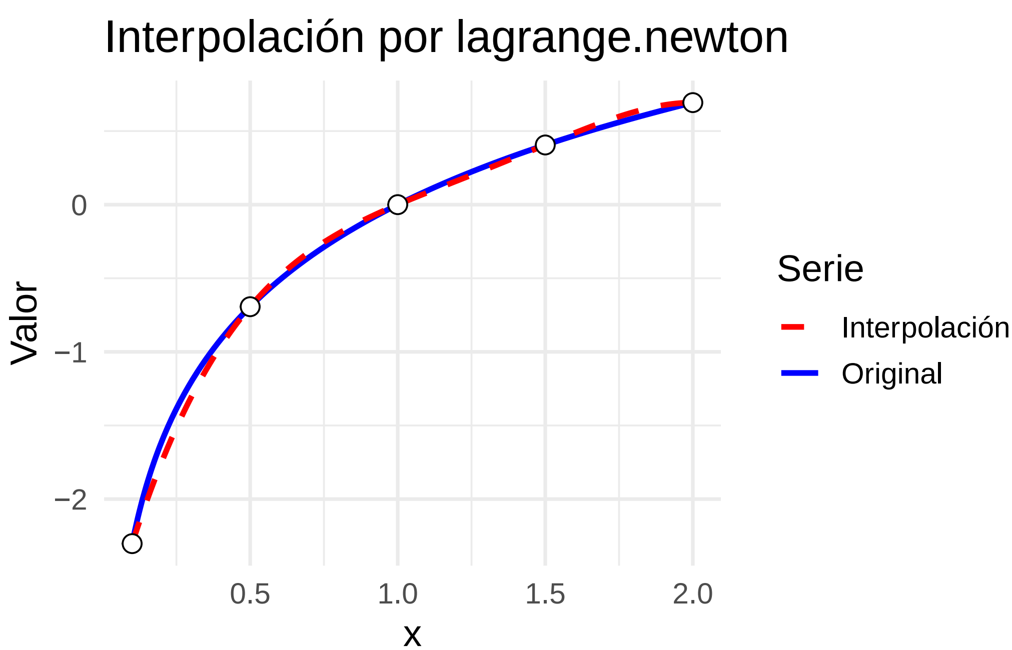
```r
      geom_line(aes(y = Hx, color = "Interpolación"),
                linewidth = 1,
                linetype = "dashed") +
      scale_color_manual(values = c(
        "Original" = "blue",
        "Interpolación" = "red"
      ))
  } else {
    p <- p +
      geom_line(aes(y = Hx, color = "Interpolación"), linewidth = 1) +
      scale_color_manual(values = c("Interpolación" = "red"))
  }
  p +
    geom_point(
      data = df_nodos,
      aes(x = x, y = y),
      shape = 21,
      size = 3,
      fill = "white"
    ) +
    labs(title = paste0("Interpolación por ", deparse(substitute(metodo))), y = "Valor", color
    theme_minimal(base_size = 14)
}
```

Pruebas

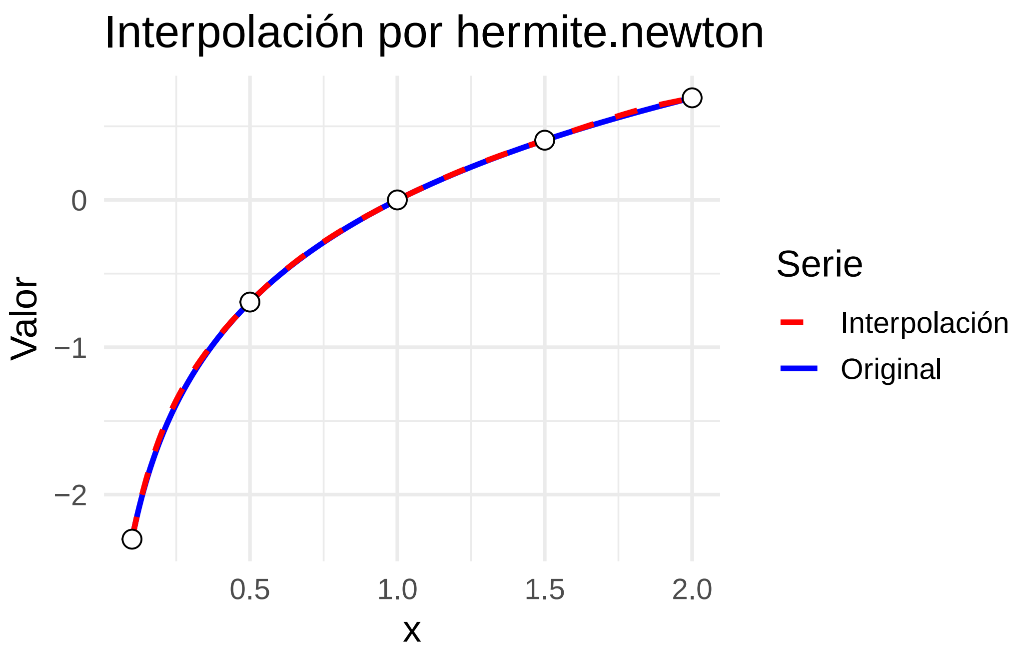
```r
f <- function(x) log(x)
df <- function(x) 1/x
nodos <- c(0.1, 0.5, 1, 1.5, 2)
graficar.polinomio(nodos, nodos[1], nodos[length(nodos)], neville, f)
```



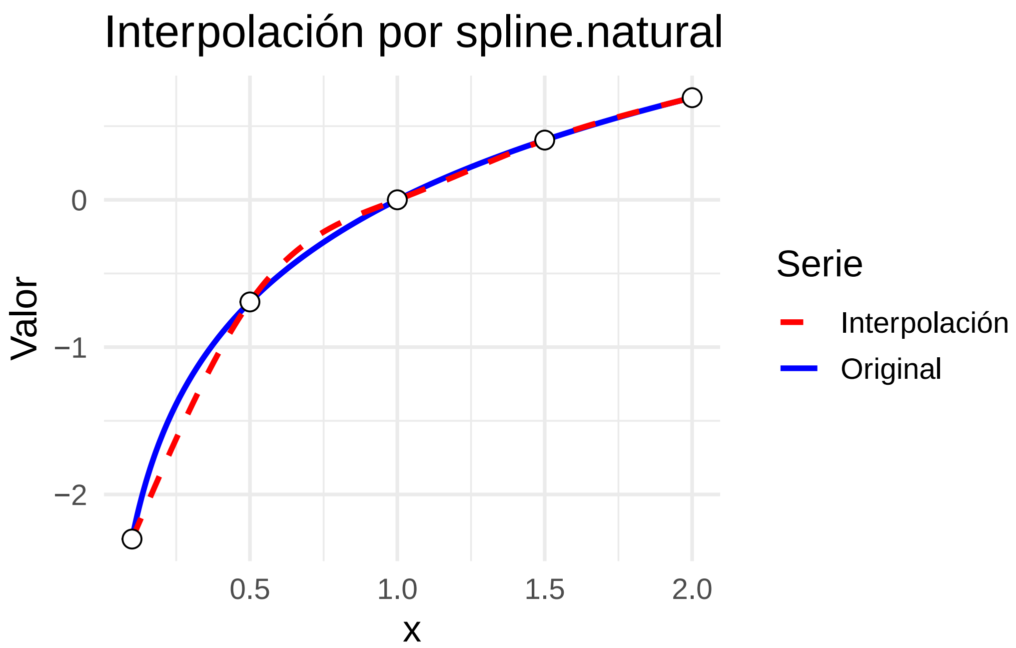

```
graficar.polinomio(nodos, nodos[1], nodos[length(nodos)], lagrange.newton, f)
```



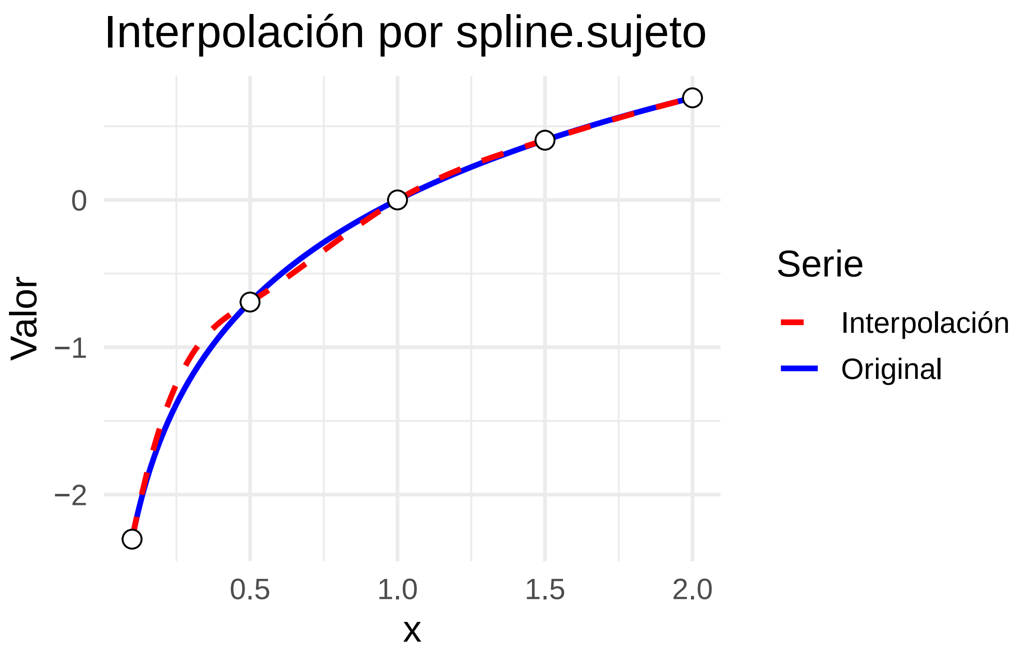

```
graficar.polinomio(nodos, nodos[1], nodos[length(nodos)], hermite.newton, f = f, df = df)
```




```
graficar.polinomio(nodos, nodos[1], nodos[length(nodos)], spline.natural, f)
```




```
graficar.polinomio(nodos, nodos[1], nodos[length(nodos)], spline.sujeto, f = f, derivadas.clamp
```



Ejercicio 3

Instrucción del ejercicio 3

Para el polinomio de Bernstein $B_n(x)$ hacer lo siguiente:

a) Demostrar que para $k \leq n$ se tiene

$$\binom{n-1}{k-1} = \frac{k}{n}\binom{n}{k}.$$

b) Pruebe que, para todo $n \in \mathbb{N}$

$$1 = \sum_{k=0}^{n} \binom{n}{k} x^k (1-x)^{n-k}.$$

c) Use (b) y (c) para probar que para $f(x) = x^2$

$$B_n(x) = \binom{n-1}{n} x^2 + \frac{1}{n} x.$$

Solución

Ejercicio 4

Instrucción del ejercicio 4

Dada la siguiente tabla de datos para $f(x)$:

x	$f(x)$
0.2	0.9798652
0.4	0.9177710
0.6	0.8080348
0.8	0.6386093
1.0	0.3843735

Aproxime $f(0.5)$ usando el procedimiento **Neville**.

Solución

Ejercicio 5

Instrucción del ejercicio 5

a) Use el algoritmo de Neville para aproximar $f(1.03)$ con $P_{0,1,2}$ para la función

$$f(x) = 3xe^x - e^{2x}$$

usando $x_0 = 1$, $x_1 = 1.05$ y $x_2 = 1.07$.

b) Suponga que la aproximación en (a) no es suficientemente exacta. Calcule $P_{0,1,2,3}$ donde $x_3 = 1.04$.

c) Compare el error real en (a) y (b) con la cota del error teórica según los teoremas vistos en clase.

Solución

Ejercicio 6

Instrucción del ejercicio 6

Repita el ejercicio anterior usando el polinomio de interpolación de Hermite, compare resultados.

Solución

Ejercicio 7

Instrucción del ejercicio 7

Use el algoritmo de Diferencias Divididas para construir el polinomio interpolante de grado 4 según la siguiente tabla:

x	$f(x)$
0.0	-7.00000
0.1	-5.89483
0.3	-5.65014
0.6	-5.17788
1.0	-4.28172

Grafique este polinomio.

Solución

Ejercicio 8

Instrucción del ejercicio 8

Use el algoritmo de Hermite para construir el polinomio interpolante de Hermite dada la siguiente tabla:

x	$f(x)$	$f'(x)$
0.2	0.9798652	0.20271
0.4	0.9177710	0.42279
0.6	0.8080348	0.68414
0.8	0.6386093	1.02964

$$1.0 \quad 0.3843735 \quad 1.55741$$

Grafique este polinomio.

Solución

Ejercicio 9

Instrucción del ejercicio 9

Use el algoritmo de Diferencias Divididas para calcular el polinomio de interpolación de Lagrange $p(x)$ de cuarto grado para:

$$f(x) = x^3 \sin(x)$$

con nodos $x_0 = 1$, $x_1 = 2$, $x_2 = 3$, $x_3 = 4$ y $x_4 = 5$.
Grafique en un mismo plano $f(x)$ y $p(x)$ y luego imprima.

Solución

Ejercicio 10

Instrucción del ejercicio 10

Probar que los polinomios $L_k(x)$ vistos en clase se pueden expresar de la forma:

$$L_k(x) = \frac{\psi(x)}{(x - x_k)\psi'(x_k)}$$

donde:

$$\psi(x) = \prod_{j=0}^{n} (x - x_j)$$

y que por lo tanto el polinomio interpolante de Lagrange se puede expresar como:

$$p(x) = \psi(x) \sum_{k=0}^{n} \frac{f(x_k)}{(x - x_k)\psi'(x_k)}.$$

Solución

Ejercicio 11

Instrucción del ejercicio 11

Demostrar que si $f(x)$ es un polinomio de grado menor o igual a n, entonces el polinomio de grado menor o igual a n que interpola $f(x)$ en $x_0, x_1, \dots, x_n$ es el mismo $f(x)$.

Solución

Ejercicio 12

Instrucción del ejercicio 12

Usar el ejercicio anterior para probar que:

$$\sum_{i=0}^{n} L_i(x) = 1$$

Solución

Ejercicio 13

Instrucción del ejercicio 13

Para las siguientes funciones:

- $f(x) = 3x^2 \ln(x) + 2x$ con nodos $x_0 = 1$, $x_1 = 1.5$, $x_2 = 2$, $x_3 = 2.5$ y $x_4 = 3$.
- $f(x) = x^2 \sin(x) - 3\cos(x)$ con nodos $x_0 = 1$, $x_1 = 2$, $x_2 = 3$, $x_3 = 4$ y $x_4 = 5$.
- $f(x) = x\cos(x) - 2x^2 + 3x - 1$ con nodos $x_0 = 1$, $x_1 = 2$, $x_2 = 3$, $x_3 = 4$ y $x_4 = 5$.

a) Encuentre el polinomio de interpolación de Lagrange $P^\star(x)$ en los nodos indicados, grafique $f(x)$ y $P^\star(x)$ en el mismo plano.

b) Encuentre el polinomio de interpolación usando Splines cúbicos $P^{\star\star}(x)$ en los nodos indicados, grafique $f(x)$ y $P^{\star\star}(x)$ en el mismo plano, luego imprima.

c) Encuentre el polinomio de interpolación de Hermite $P^{\star\star\star}(x)$ en los nodos indicados, grafique $f(x)$ y $P^{\star\star\star}(x)$ en el mismo plano.

d) Grafique $f(x)$, $P^\star(x)$, $P^{\star\star}(x)$ y $P^{\star\star\star}(x)$ en el mismo plano. ¿Qué se puede concluir?

Solución

Ejercicio 14

Instrucción del ejercicio 14

¿Existen a, b, c, d y e tal que la función:

$$S(x) = \begin{cases} ax^3 + x^2 + cx, & -1 \leq x \leq 0, \\ bx^3 + x^2 + dx, & 0 \leq x \leq 1 \end{cases}$$

sea el spline cúbico natural que coincide con la función $f(x) = |x|$ en los nodos $-1, 0, 1$?

Solución

Ejercicio 15

Instrucción del ejercicio 15

Encuentre los valores de a, b, c, d y e tal que la función $S(x)$ es un spline cúbico natural:

$$S(x) = \begin{cases} a + b(x-1) + c(x-1)^2 + d(x-1)^3, & 0 \leq x \leq 1, \\ (x-1)^3 + ex^2 - 1, & 1 \leq x \leq 2. \end{cases}$$

Solución

Ejercicio 16

Instrucción del ejercicio 16

Encuentre los valores de a, b, c y d tal que la función $S(x)$ es un spline cúbico y cumple que $\int_0^2 \left[S''(x)\right]^2 dx$ es mínimo (esta condición sustituye a la condición para ser spline natural o sujeto):

$$S(x) = \begin{cases} 3 + x - 9x^3, & 0 \leq x \leq 1, \\ a + b(x-1) + c(x-1)^2 + d(x-1)^3, & 1 \leq x \leq 2. \end{cases}$$

Solución

Ejercicio 17

> **i** Instrucción del ejercicio 17

El objetivo de este ejercicio es estudiar e implementar un algoritmo para **Aproximación discreta por mínimos cuadrados**. Para esto:

a) El problema en general es aproximar una tabla de datos $\{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ por un polinomio de grado $n < m - 1$ denotado por

$$P_n(x) = \sum_{k=0}^{n} a_k x^k.$$

La idea es encontrar constantes $\{a_k\}_{k=0}^{n}$ tal que se minimice el error:

$$E = \sum_{i=1}^{m} \left(y_i - P_n(x_i)\right)^2.$$

Pruebe que este mínimo se alcanza en la solución del sistema de **ecuaciones normales** (n+1)×(n+1) para las incógnitas $\{a_k\}_{k=0}^{n}$ dado por:

$$\sum_{k=0}^{n} a_k \sum_{i=1}^{m} x_i^{j+k} = \sum_{i=1}^{m} y_i x_i^j, \qquad j = 0, 1, 2, \ldots, n.$$

b) Dada una tabla de datos para f, escriba una función en **R** que permita generar el sistema de ecuaciones normales del inciso (a).

c) Luego escriba una función en **R** que encuentre los coeficientes del polinomio de mínimos cuadrados y luego lo grafique.

d) Construir la **aproximación de mínimos cuadrados de grado 3** para la siguiente tabla y **construir el gráfico**.

x_i	y_i
4.0	102.56
4.2	113.18
4.5	130.11
4.7	142.05
5.1	167.53
5.5	195.14
5.9	224.87
6.3	256.73
6.8	299.50
7.1	326.72

Solución

Ejercicio 18

Instrucción del ejercicio 18

El objetivo de este ejercicio es **generalizar la aproximación discreta por mínimos cuadrados**.

Dada una función $f \in C[a,b]$, se requiere un polinomio $\tilde{P}_n(x) = \sum_{k=0}^{n} a_k x^k$ de manera tal que las constantes $\{a_k\}_{k=0}^{n}$ minimicen el error:

$$E = \int_a^b \left(f(x) - \tilde{P}_n(x)\right)^2 dx.$$

Pruebe que este mínimo se alcanza en la solución del sistema de $(n+1)$ **ecuaciones normales** y $(n+1)$ incógnitas $\{a_k\}_{k=0}^{n}$ dado por:

$$\sum_{k=0}^{n} a_k \int_a^b x^{j+k}\, dx \;=\; \int_a^b x^j f(x)\, dx, \qquad j = 0, 1, 2, \dots, n. \tag{2}$$

a) Dada una función f escriba una función en **R** que permita **generar el sistema de ecuaciones (2)**.

b) Luego escriba una función en **R** que **encuentre los coeficientes** del polinomio $\tilde{P}_n(x)$ y luego lo grafique.

c) Encuentre la **aproximación polinómica** $\tilde{P}_n(x)$ de grado 2, 4 y 6 para $f(x) = \cos(\pi x)$ en el intervalo $[-1, 1]$. Además, **construya los gráficos**.

Solución

Ejercicio 19

Instrucción del ejercicio 19

a) Demuestre que el **Polinomio de Hermite** visto en clase $H_{2n+1}(x)$ es **único**.
Sugerencia: Suponga que existe otro polinomio $P(x)$ que cumple las condiciones de interpolación de Hermite y considere $D = H_{2n+1}(x) - P(x)$ y D' en $x_0, x_1, \dots, x_n$.

b) Demuestre que el **error absoluto** en este caso está dado por:

$$\left|f(x) - H_{2n+1}(x)\right| = \left|\frac{(x-x_0)^2 \cdots (x-x_n)^2}{(2n+2)!} f^{(2n+2)}(\xi)\right|, \quad \text{con } \xi \in (a,b).$$

Sugerencia: Use el mismo método que usamos para demostrar la fórmula del error absoluto en el caso de Lagrange, pero con:

$$g(t) = f(t) - H_{2n+1}(t) - \frac{(t-x_0)^2 \cdots (t-x_n)^2}{(x-x_0)^2 \cdots (x-x_n)^2}\Big[f(x) - H_{2n+1}(x)\Big].$$

Solución