Analisis Numerico - Taller Interpolación

Juan Francisco Hamon, Diego Mauricio Bulla, Juan Diego Campos Neira

27/10/2019

1)Dado los $n+1$ puntos distintos (x_i, y_i) el polinomio interpolante que incluye a todos los puntos es unico.

Demostracion del teorema de existencia y unicidad de un polinomio interpolante:

Sean $x_1, \ldots, x_n$ numeros diferentes por pares y $y_1, \ldots, y_n$ algunos numeros. Entonces existe un unico polinomio P de grado $\leq n-1$ tal que $P(x_j) = y_j$ para cada j en $\{1, \ldots, n\}$.

Los coeficientes del polinomio P se denotan como $c_1, .., c_{n-1}$, obteniendo:

$P(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{n-1} x^{n-1}$

Ahora, sustituyedo los x por $x_1, x_2, \ldots, x_n$, se obtiene un sistema de ecuaciones lineales para las incognitas $c_0, c_1, c_2, \ldots, c_{n-1}$

$c_0 + c_1 x_1 + c_2 x_1^2 + \ldots + c_{n-1} x_1^{n-1}$

$c_0 + c_1 x_2 + c_2 x_2^2 + \ldots + c_{n-1} x_2^{n-1}$

. ..

$c_0 + c_1 x_n + c_2 x_n^2 + \ldots + c_{n-1} x_n^{n-1}$

La matriz que genera este sistema de ecuaciones se le conoce como la matriz de Valdermonde, y esta asociada a los puntos $x_1, \ldots, x_n$, y el sistema se escribe brevemente en la forma:

$V(x_1, \ldots, x_n)c = y$

Donde $c = [c_{k-1}]^n$ es el vector de los coeficientes incognitos. El determinante de este sistema es el determinante de Valdermonde y se calcula como el producto de todas las diferencias $x_j - x_i$ con $i < j$.

$$\prod_{j,k \epsilon (1,\ldots,n); j<k} (x_k - x_j)$$

Como los puntos $x_1, \ldots, x_n$ son diferentes por pares, todas las diferencias $x_k - x_j$ son diferentes de cero y, al mismo tiempo, el determinante de la matriz de Valdermonde es diferente de cero. Por lo tanto, el sistema de ecuaciones lineales tiene una unica solocion, es decir, existe un unico polinomio que cumple con dichas propiedades.

2)Construya un polinomio de grado 3 que pase por: (0,10), (1,15), (2,5) y que la tangente sea igual a 1 en x_0.

```
require(pracma)

## Loading required package: pracma

#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```
divided.differences <- function(x, y, x0) {
  require(rSymPy)
  n <- length(x)
  q <- matrix(data = 0, n, n)
  q[,1] <- y
  f <- as.character(round(q[1,1], 5))
  fi <- ''

  for (i in 2:n) {
    for (j in i:n) {
      q[j,i] <- (q[j,i-1] - q[j-1,i-1]) / (x[j] - x[j-i+1])
    }
    fi <- paste(fi, '*(x - ', x[i-1], ')', sep = '', collapse = '')

    f <- paste(f, ' + ', round(q[i,i], 5), fi, sep = '', collapse = '')
  }

  x <- Var('x')
  sympy(paste('e = ', f, collapse = '', sep = ''))
  approx <- sympy(paste('e.subs(x, ', as.character(x0), ')', sep = '',
collapse = ''))

  return(list('Approximation from Interpolation'=as.numeric(approx),
              'Interpolated Function'=f,
              'Divided Differences Table'=q))
}

x = c(0,1,2)
y = c(10,15,5)

resultados <- divided.differences(x,y,1)

## Loading required package: rSymPy

## Loading required package: rJython

## Loading required package: rJava
```

```
## Loading required package: rjson

print(resultados$`Interpolated Function`)

## [1] "10 + 5*(x - 0) + -7.5*(x - 0)*(x - 1)"
```

4)Con la funcion $f(x) = ln(x)$ construya la interpolación de diferencias divididas en $x_0 = 1; x_1 = 2$ y estime el error en [1,2].

```
require(pracma)
#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```
f <- function(x){
  log(x)
}

divided.differences <- function(x, y, x0) {
  require(rSymPy)
  n <- length(x)
  q <- matrix(data = 0, n, n)
  q[,1] <- y
  f <- as.character(round(q[1,1], 5))
  fi <- ''

  for (i in 2:n) {
    for (j in i:n) {
      q[j,i] <- (q[j,i-1] - q[j-1,i-1]) / (x[j] - x[j-i+1])
    }
    fi <- paste(fi, '*(x - ', x[i-1], ')', sep = '', collapse = '')

    f <- paste(f, ' + ', round(q[i,i], 5), fi, sep = '', collapse = '')
  }

  x <- Var('x')
  sympy(paste('e = ', f, collapse = '', sep = ''))
  approx <- sympy(paste('e.subs(x, ', as.character(x0), ')', sep = '',
collapse = ''))

  return(list('Approximation from Interpolation'=as.numeric(approx),
              'Interpolated Function'=f,
              'Divided Differences Table'=q))
}

xs = seq(1,2)
y = f(xs)
```

```r
resultados <- divided.differences(xs,y,1)
resultados2 <- divided.differences(xs,y,2)

cat("Ln(x) en x = 1 \n")

## Ln(x) en x = 1

print(resultados$`Approximation from Interpolation`)

## [1] 0

print(resultados$`Interpolated Function`)

## [1] "0 + 0.69315*(x - 1)"

print(resultados$`Divided Differences Table`)

##           [,1]      [,2]
## [1,] 0.0000000 0.0000000
## [2,] 0.6931472 0.6931472

cat("Ln(x) en x = 2 \n")

## Ln(x) en x = 2

print(resultados2$`Approximation from Interpolation`)

## [1] 0.69315

print(resultados2$`Interpolated Function`)

## [1] "0 + 0.69315*(x - 1)"

print(resultados2$`Divided Differences Table`)

##           [,1]      [,2]
## [1,] 0.0000000 0.0000000
## [2,] 0.6931472 0.6931472
```

5)Utilice la interpolación de splines cúbicos para el problema de la mano y del perrito.

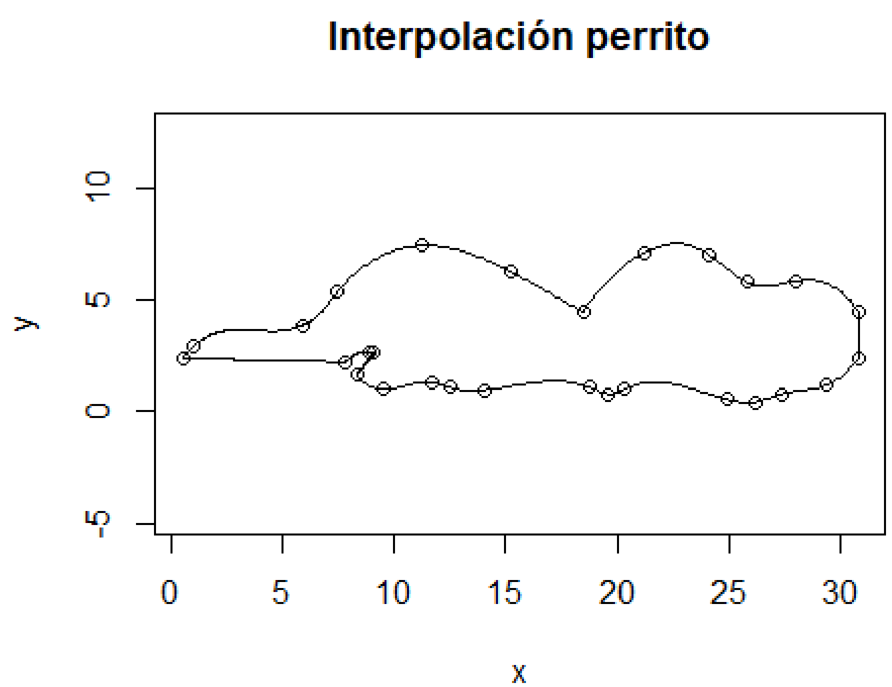
Para desarrollar el problema, se tomaron los puntos al importar la imágen del perrito a Adobe Illustrator CC 2019.

Perrito

```r
library(stats)
#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```
x = c(00.50, 01.01, 05.85, 07.46, 11.28, 15.20, 18.46, 21.25, 24.15,
25.80, 28.00, 30.80, 30.81, 29.40, 27.40, 26.21, 24.97, 20.32, 19.54,
18.80, 14.04, 12.54, 11.68, 09.55, 08.30, 09.10, 08.85, 07.80, 00.50)
y = c(02.40, 02.95, 03.86, 05.41, 07.45, 06.30, 04.49, 07.15, 07.05,
05.80, 05.85, 04.50, 02.40, 01.20, 00.80, 00.44, 00.54, 01.01, 00.80,
01.08, 00.98, 01.08, 01.33, 01.00, 01.64, 02.65, 02.70, 02.24, 02.40)

plot(x,y,main = "Interpolación perrito", asp = 1)

vx1 = c(x[1:4])
vy1 = c(y[1:4])

splines = splinefun(vx1,vy1, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx1[1], to =
vx1[length(vx1)])

vx2 = c(x[4:7])
vy2 = c(y[4:7])

splines = splinefun(vx2,vy2, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx2[1], to =
vx2[length(vx2)])

vx3 = c(x[7:12])
vy3 = c(y[7:12])

splines = splinefun(vx3,vy3, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx3[1], to =
vx3[length(vx3)])

vx4 = c(x[12:13])
vy4 = c(y[12:13])

splines = splinefun(vx4,vy4, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx4[1], to =
vx4[length(vx4)])

vx5 = c(x[13:18])
vy5 = c(y[13:18])

splines = splinefun(vx5,vy5, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx5[1], to =
vx5[length(vx5)])

vx6 = c(x[18:25])
vy6 = c(y[18:25])

splines = splinefun(vx6,vy6, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx6[1], to =
```

```
vx6[length(vx6)])

vx7 = c(x[25:26])
vy7 = c(y[25:26])

splines = splinefun(vx7,vy7, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx7[1], to =
vx7[length(vx7)])

vx8 = c(x[26:28])
vy8 = c(y[26:28])

splines = splinefun(vx8,vy8, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx8[1], to =
vx8[length(vx8)])

vx9 = c(x[28:29])
vy9 = c(y[28:29])

splines = splinefun(vx9,vy9, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx9[1], to =
vx9[length(vx9)])
```



Mano

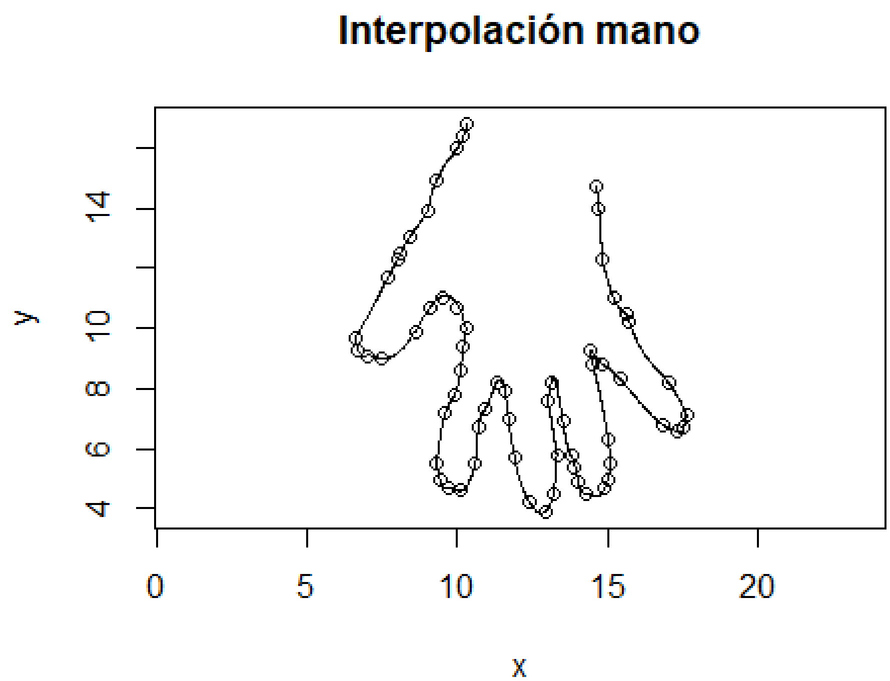
```
library(stats)
#Se limpian los elementos creados con anterioridad
```

```r
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```r
x=c(14.6, 14.7, 14.8, 15.2, 15.6, 15.7, 17.0, 17.6, 17.5, 17.3, 16.8,
15.4, 14.8, 14.4, 14.5, 15.0, 15.1, 15.0, 14.9, 14.3, 14.0, 13.9, 13.8,
13.5, 13.1, 13.0, 13.3, 13.2, 12.9, 12.4, 11.9, 11.7, 11.6, 11.3, 10.9,
10.7, 10.6, 10.1, 9.7, 9.4, 9.3, 9.6, 9.9, 10.1, 10.2, 10.3, 9.10, 8.6,
7.5, 7.0, 6.7, 6.6, 7.70, 8.00, 8.10, 8.40, 9.00, 9.30, 10, 10.2, 10.3,
10.0, 9.50)
y=c(14.7, 14.0, 12.3, 11.0, 10.5, 10.2, 8.20, 7.10, 6.70, 6.60, 6.80,
8.30, 8.80, 9.30, 8.80, 6.30, 5.50, 5.00, 4.70, 4.50, 4.90, 5.40, 5.80,
6.90, 8.20, 7.60, 5.80, 4.50, 3.90, 4.20, 5.70, 7.00, 7.90, 8.20, 7.30,
6.70, 5.50,  4.60, 4.7, 5.0, 5.5, 7.2, 7.8, 8.60, 9.40, 10.0, 10.7, 9.9,
9.0, 9.1, 9.3, 9.7, 11.7, 12.3, 12.5, 13.0, 13.9, 14.9, 16, 16.4, 16.8,
10.7, 11.0)

plot(x,y,main = "Interpolación mano", asp = 1)

vx1 = c(x[1:3])
vy1 = c(y[1:3])

splines = splinefun(vx1,vy1, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx1[1], to =
vx1[length(vx1)])

vx2 = c(x[3:5])
vy2 = c(y[3:5])

splines = splinefun(vx2,vy2, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx2[1], to =
vx2[length(vx2)])

vx3 = c(x[5:8])
vy3 = c(y[5:8])

splines = splinefun(vx3,vy3, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx3[1], to =
vx3[length(vx3)])

vx4 = c(x[8:10])
vy4 = c(y[8:10])

splines = splinefun(vx4,vy4, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx4[1], to =
vx4[length(vx4)])
```

```
vx5 = c(x[10:13])
vy5 = c(y[10:13])

splines = splinefun(vx5,vy5, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx5[1], to =
vx5[length(vx5)])

vx6 = c(x[13:14])
vy6 = c(y[13:14])

splines = splinefun(vx6,vy6, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx6[1], to =
vx6[length(vx6)])

vx7 = c(x[14:17])
vy7 = c(y[14:17])

splines = splinefun(vx7,vy7, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx7[1], to =
vx7[length(vx7)])

vx8 = c(x[17:23])
vy8 = c(y[17:23])

splines = splinefun(vx8,vy8, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx8[1], to =
vx8[length(vx8)])

vx9 = c(x[23:26])
vy9 = c(y[23:26])

splines = splinefun(vx9,vy9, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx9[1], to =
vx9[length(vx9)])

vx10 = c(x[26:27])
vy10 = c(y[26:27])

splines = splinefun(vx10,vy10, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx10[1], to =
vx10[length(vx10)])

vx11 = c(x[27:28])
vy11 = c(y[27:28])

splines = splinefun(vx11,vy11, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx11[1], to =
vx11[length(vx11)])
```

```r
vx12 = c(x[28:33])
vy12 = c(y[28:33])

splines = splinefun(vx12,vy12, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx12[1], to =
vx12[length(vx12)])

vx13 = c(x[33:37])
vy13 = c(y[33:37])

splines = splinefun(vx13,vy13, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx13[1], to =
vx13[length(vx13)])

vx14 = c(x[37:41])
vy14 = c(y[37:41])

splines = splinefun(vx14,vy14, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx14[1], to =
vx14[length(vx14)])

vx15 = c(x[41:46])
vy15 = c(y[41:46])

splines = splinefun(vx15,vy15, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx15[1], to =
vx15[length(vx15)])

vx16 = c(x[46:52])
vy16 = c(y[46:52])

splines = splinefun(vx16,vy16, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx16[1], to =
vx16[length(vx16)])

vx17 = c(x[52:57])
vy17 = c(y[52:57])

splines = splinefun(vx17,vy17, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx17[1], to =
vx17[length(vx17)])

vx18 = c(x[57:61])
vy18 = c(y[57:61])

splines = splinefun(vx18,vy18, method = "fmm")
curve(splines(x), add = TRUE, col = 1, from = vx18[1], to =
vx18[length(vx18)])
```



8)Considere el comportamiento de gases no ideales se describe a menudo con la ecuación viral de estado. Los siguientes datos para el nitrogeno N_2:

```
#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```
datos = matrix(c(100, 200,  300, 400,  500,  600,
                 -160, -35, -4.2,  9 , 16.9, 21.3),nrow = 2,ncol = 6,
byrow = TRUE)

rownames(datos)<-c("T(K)","B(cm^3)/mol")
datos <- as.table(datos)
datos

##                  A      B      C      D      E      F
## T(K)          100.0  200.0  300.0  400.0  500.0  600.0
## B(cm^3)/mol -160.0  -35.0   -4.2    9.0   16.9   21.3
```

Donde T es la temperatura $[K]$ y B es el segundo coeficiente viral. El comportamiento de gases no ideales se describe a menudo con la ecuación viral de estado:

$$\frac{PV}{RT} = 1 + \frac{B}{V} + \frac{C}{C^2} + \ldots$$

Donde P es la presión, V el volumen molar del gas, T la temperatura Kelvin y R es la constante de gas ideal. Los coeficientes $B = B(T)$, $C = C(T)$, son el segundo y tercer coeficiente viral, respectivamente. En la práctica se usa la serie truncada para aproximar:

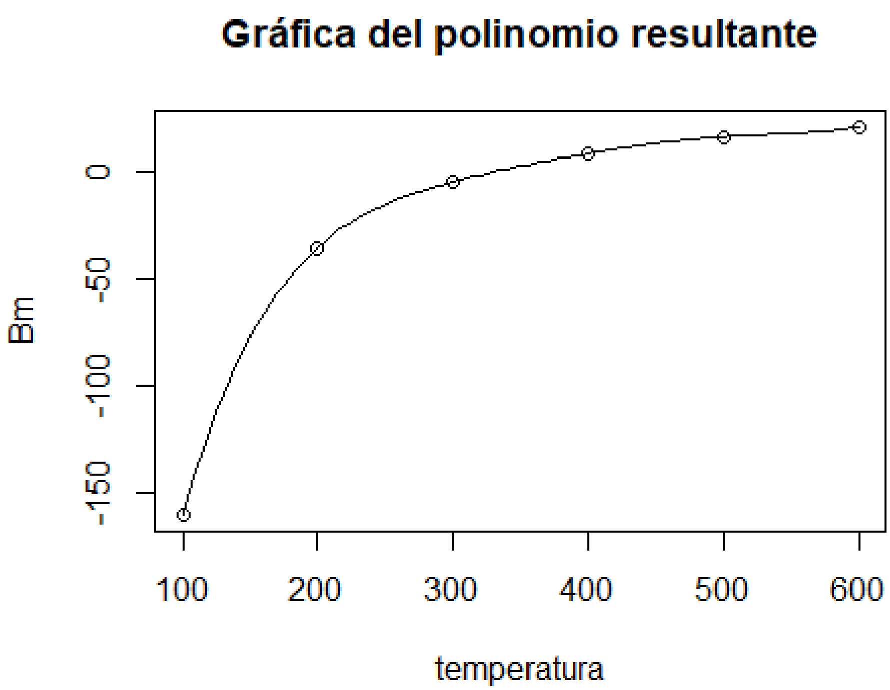
$$\frac{PV}{RT} = 1 + \frac{B}{V}$$

a)Determine un polinomio interpolante para este caso b)Utilizando el resultado anterior calcule el segundo y tercer coeficiente viral a $450K$ c)Grafique los puntos y el polinomio que se ajusta d)Utilice la interpolación de Lagrange y escriba el polinomio interpolante

```
library(PolynomF)

##
## Attaching package: 'PolynomF'

## The following object is masked from 'package:pracma':
##
##     integral

#Se limpian los elementos creados con anterioridad
rm(list=ls())
#Se limpia la consola para una mejor visualización
cat("\014")
```

```
#Punto a

temperatura = c(100,200,300,400,500,600)
Bm = c(-160, -35, -4.2, 9, 16.9, 21.3)

A = matrix(c(1, temperatura[1],temperatura[1]^2, temperatura[1]^3,
temperatura[1]^4, temperatura[1]^5
            ,1, temperatura[2],temperatura[2]^2, temperatura[2]^3,
temperatura[2]^4, temperatura[2]^5
            ,1, temperatura[3],temperatura[3]^2, temperatura[3]^3,
temperatura[3]^4, temperatura[3]^5
            ,1, temperatura[4],temperatura[4]^2, temperatura[4]^3,
temperatura[4]^4, temperatura[4]^5
            ,1, temperatura[5],temperatura[5]^2, temperatura[5]^3,
temperatura[5]^4, temperatura[5]^5
            ,1, temperatura[6],temperatura[6]^2, temperatura[6]^3,
temperatura[6]^4, temperatura[6]^5
            ),nrow = 6, ncol = 6, byrow = TRUE)

sol = solve(A,Bm)

polinomio = function(x){
```

```r
  sol[1]+sol[2]*x+sol[3]*x^2+sol[4]*x^3+sol[5]*x^4+sol[6]*x^5
}

#Punto b

B = polinomio(450)
cat("El segundo coeficiente viral a 450K es",B,"\n")

## El segundo coeficiente viral a 450K es 13.88437

#Punto c

plot(temperatura, Bm, main = "Gráfica del polinomio resultante")
curve(polinomio, add = TRUE)
```




```r
#Punto d

PolinomioLagrange = poly_calc(temperatura, Bm)
cat("Polinomio de Lagrange resultante: \n")

## Polinomio de Lagrange resultante:

print(PolinomioLagrange)

## -573.9 + 6.63535*x - 0.03183458*x^2 + 7.766667e-05*x^3 - 9.404167e-
08*x^4

cat("\n")
```