ANALISIS NÚMERICO

TALLER INTERPOLACIÓN 2

Autores

CAMILO EDUARDO CRUZ GUTIERREZ
JOHAN FERNEY GARCIA PACHON
ESTEBAN CASAS GARZON

Profesora
EDDY HERREA

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE MATEMATICAS
BOGOTA D.C.

1. Dados n+1 nodos distintos, demuestre que el polinomio interpolante es único:

Sean x1, . . ., xn algunos números diferentes por pares y sean y1, . . ., yn algunos números.
Entonces existe un único polinomio P de grado ≤ n − 1 tal que:
$P(xj) = yj \ para \ cada \ j \ en$ {1, . . . , n}.
Demostración:
Denotemos por c0, . . ., cn−1 a los coeficientes del polinomio:
$P(x)$ = $c0 + c1x + c2x0$
2 + . . . + cn − $1x \ n$ − 1 .
Sustituyendo x = x1, luego x = x2, etc., hasta x = xn, obtenemos el siguiente sistema de ecuaciones
lineales para las incógnitas
$c0 + x1c1 + x$ 2 1 $c2$ + . . . + $x \ n$ − 1 1 cn − 1 = $y1$
$c0 + xnc1 + x$ 2 $n \ c2$ + . . . + $x \ n$ − 1 $n \ cn$ − 1 = yn.
La matriz de este sistema es la matriz de Vandermonde asociada a los puntos x1, . . ., xn, y el
sistema se escribe brevemente en la forma
V ($x1$, . . . , xn)c = y,
donde c = ck−1 n k=1 es el vector de los coeficientes incognitos.
El determinante de este sistema es el determinante de Vandermonde y se calcula como el
producto de todas las diferencias xj − xi con i < j:
$det \ V(x1$, . . . , xn) = ∏ (xk − xj).
j,k∈{1,...,n} j<k
Como los puntos x1, . . ., xn son diferentes por pares, todas estas diferencias xk − xj son distintas
de cero, y el determinante es distinto de cero. Por lo tanto, el sistema de ecuaciones lineales tiene
una solución única, esto es, existe un único polinomio que cumple con dichas propiedades.

2. Considere el comportamiento de gases no ideales se describe a menudo con la ecuación viral de estado. los siguientes datos para el nitrógeno:

T(K) 100 200 300 400 450 500 600
B(cm3/mol) 160 35 4.2 9.0 16.9 21.3
Donde:
El comportamiento de gases no ideales se describe a menudo con la ecuación virial de estado:

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

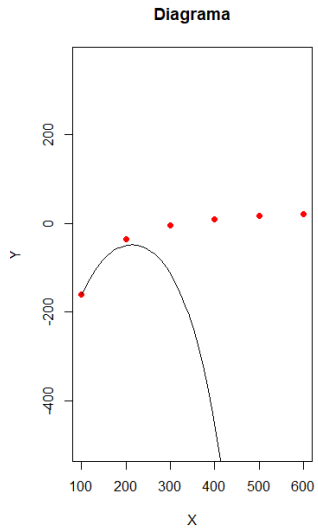
Donde p es la presión V, el volumen molar del gas, T es la temperatura Kelvin y R es la constante de gas ideal. Los coeficientes B, C(T)… son el segundo y tercer coeficiente viral, Respectivamente. En la práctica se usa la serie truncada

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

a. Determine un polinomio interpolante para este caso (escriba el polinomio).
b. Utilizando el resultado anterior calcule el segundo y tercer coeficiente virial a 450K.
c. Grafique los puntos y el polinomio que ajusta.
d. Utilice la interpolación de Lagrange y escriba el polinomio interpolante.
e. Grafique los puntos y el polinomio interpolante de Lagrange.
f. ¿Cuál es el segundo y tercer coeficiente virial a 450K?. con el método de Lagrange.
g. Compare su resultado con la serie truncada (modelo teórico), ¿cuál de las tres aproximaciones es mejor por qué?

```
library(PolynomF)
library(rSymPy)
#Se debe instalar RSymPy
lagrange.poly <- function(x, y) {

  l <- list() # List to store Lagrangian polynomials L_{1,2,3,4}
  k <- 1

  for (i in x) {
    # Set the numerator and denominator of the Lagrangian polynomials to 1 and
build them up
    num <- 1
    denom <- 1

    # Remove the current x value from the iterated list
    p <- x[! x %in% i]

    # For the remaining points, construct the Lagrangian polynomial by
successively
    # appending each x value
    for (j in p) {
      num <- paste(num, "*", "(", 'x', " - ", as.character(j), ")", sep = "",
collapse = "")
      denom <- paste(denom, "*", "(", as.character(i)," - ", as.character(j),
")", sep = "", collapse = "")
    }

    # Set each Lagrangian polynomial in rSymPy to simplify later.
    l[k] <- paste("(", num, ")", "/", "(", denom, ")", sep = "", collapse = "")
    k <- k + 1
  }
```

```
  # Similar to before, we construct the final Lagrangian polynomial by
successively building
  # up the equation by iterating through the polynomials L_{1,2,3,4} and the y
values
  # corresponding to the x values.
  eq <- 0

  for (i in 1:length(y)) {
    eq <- paste(eq, '+', as.character(y[i]), "*", l[[i]], sep = "", collapse =
"")
  }

  # Define x variable for rSymPy to simplify
  x <- Var('x')

  # Simplify the result with rSymPy and return the polynomial
  return(sympy(paste("simplify(", eq, ")")))
}
c<-c(100,200,300,400,500,600)
y<-c(-160,-35,-4.2,9,16.9,21.3)
plot(c,y, pch=19, cex=1, col = "red", asp=1,xlab="X", ylab="Y",
main="Diagrama")
re=lagrange.poly(c,y)
Ajuste_Polinomio = poly.calc(c,y)
curve(Ajuste_Polinomio,add=T,from = 100, to = 700)
f <- as.function(alist(x=, eval(parse(text=re))))
curve(f,add=T,from = 100, to = 700,col="Blue")
cat("Ajuste polinomico: en negro. Lagrange: Azul \n")
cat("Con Lagrange: \nEl polinomio interpolante es: ",re,"\n")
cat("Cuando T=450K, B tiene un valor de: ",f(450),"\n")
cat("Con ajuste Polinomico: \nEl polinomio interpolante es:
",as.character(Ajuste_Polinomio),"\n")
cat("Cuando T=450K, B tiene un valor de: ",Ajuste_Polinomio(450),"\n")
```

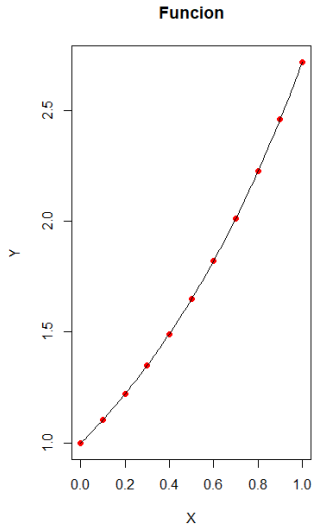


3. Sea $f(x) = e^x$ el intervalo [0,1].
   a. Tabular varios puntos y grafíquelos.
   b. Interpolar con el método de Lagrange.
   c. Utilizando 8 cifras decimales o más, en cada entrada, determine el tamaño del paso que me produzca un error por debajo de 10^{-6}.

```
f<-
function(x
)
              {
                return (exp(x))
              }
              x = c(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)
              y =
              c(f(0),f(0.1),f(0.2),f(0.3),f(0.4),f(0.5),f(0.6),f(0.7),f(0.8),f(0.9),f(
              1))
              plot(x,y, pch=19, cex=1, col = "red", asp=1,xlab="X", ylab="Y",
              main="Diagrama ")


              lagrange.poly <- function(x, y)
              {
                require(rSymPy)

                if (length(x) != length(y)) {
                  stop('x and y must be of equal length')
                }

                l <- list() # List to store Lagrangian polynomials L_{1,2,3,4}
                k <- 1
```

```
  for (i in x) {
    # Set the numerator and denominator of the Lagrangian polynomials to
1 and build them up
    num <- 1
    denom <- 1

    # Remove the current x value from the iterated list
    p <- x[! x %in% i]

    # For the remaining points, construct the Lagrangian polynomial by
successively
    # appending each x value
    for (j in p) {
      num <- paste(num, "*", "(", 'x', " - ", as.character(j), ")", sep
= "", collapse = "")
      denom <- paste(denom, "*", "(", as.character(i)," - ",
as.character(j), ")", sep = "", collapse = "")
    }

    # Set each Lagrangian polynomial in rSymPy to simplify later.
    l[k] <- paste("(", num, ")", "/", "(", denom, ")", sep = "",
collapse = "")
    k <- k + 1
  }

  # Similar to before, we construct the final Lagrangian polynomial by
successively building
  # up the equation by iterating through the polynomials L_{1,2,3,4} and
the y values
  # corresponding to the x values.
  eq <- 0

  for (i in 1:length(y)) {
    eq <- paste(eq, '+', as.character(y[i]), "*", l[[i]], sep = "",
collapse = "")
  }

  # Define x variable for rSymPy to simplify
  x <- Var('x')

  # Simplify the result with rSymPy and return the polynomial
  return(sympy(paste("simplify(", eq, ")")))
```

```
}
DatosX = x[1:11]; DatosY = y[1:11]
Ajuste_Polinomio = poly.calc(DatosX,DatosY)
plot(x,y, pch=19, cex=1, col = "red", asp=1,xlab="X", ylab="Y",
main="Funcion")
points(DatosX,DatosY, pch=19, cex=1, col = "red", asp=1,xlab="X",
ylab="Y", main="Funcion")
curve(Ajuste_Polinomio,add=T,from =0,to =1)
aux =lagrange.poly(x,y)
f<-as.function(alist(a = , eval (parse((text=aux)))))
curve(Ajuste_Polinomio,add=T,from =0,to =1)
```



4. En la tabla que sigue aparece las estadísticas de un curso con la cantidad de estudiantes en cada rango de notas.

Rango de notas	30-40	40-50	50-60	60-70	70-80
No. Estudiantes	35	48	70	40	22

```
install.packages("PolynomF")#instalar
paquete
                                    library(PolynomF)


                                    #Hallar frecuencia acumulada
                                    x <- c(40,50,60,70,80)
                                    y <- c(35,83,153,193,215)


                                    datx = x[1:5]; daty = y[1:5]
```

```r
polyAjuste = poly.calc(datx,daty)
polyAjuste
plot(datx,daty,pch=19, cex=1, col = "red",
asp=1)
curve(polyAjuste,add=T)


evaluarfuncion <- function(f, a){
  f(a)
}




polinomio <- function(x) 3343-
239.3667*x+6.183333*x**2-
0.06733333*x**3+0.0002666667*x**4




num <- evaluarfuncion(polinomio, 55)
print(num)
e = 120 - num
cat("Error:",round(e,3))


lagrange = function(x,y,a){
  n = length(x)
  if(a < min(x) || max(x) < a) stop("No está
interpolando")
  X = matrix(rep(x, times=n), n, n, byrow=T)
  mN = a - X; diag(mN) = 1
  mD = X - t(X); diag(mD) = 1
  Lnk = apply(mN, 1, prod)/apply(mD, 2, prod)
  sum(y*Lnk)
}
cat("Resultado con Lagrange:
",lagrange(x,y,55))
print("Error nulo")
```

```
> num <- evaluarfuncion(polinomio, 55)
> print(num)
[1] 119.998
> e = 120 - num
> cat("Error:",round(e,3))
Error: 0.002>
> lagrange = function(x,y,a){
+   n = length(x)
+   if(a < min(x) || max(x) < a) stop("No está int
erpolando")
+   X = matrix(rep(x, times=n), n, n, byrow=T)
+   mN = a - X; diag(mN) = 1
+   mD = X - t(X); diag(mD) = 1
+   Lnk = apply(mN, 1, prod)/apply(mD, 2, prod)
+   sum(y*Lnk)
+ }
> cat("Resultado con Lagrange: ",lagrange(x,y,55))
Resultado con Lagrange:  120> print("Error nulo")
```

5. Considere la función de valor real dada por $f(x) = \frac{1}{1+x^2}$ conocida en el intervalo $[-1,1]$ y una partición de la forma $x_i = \cos\left(\frac{2(n-i)+1}{2n+2}\pi\right)$ $i = 0,1,2 \dots, n$. Demuestre que para un valor en el intervalo se tiene que:

$$|f(x^*) - P_n(x^*)| \leq \frac{M}{(n+1)!}\frac{1}{2^n} \quad \text{si} \quad |f^{(n+1)}(x)| \leq M \quad \text{para todo} \quad x \in [a,b].$$

5. De acuerdo a la función dada consideramos a $f(x)$ como la función a interpolar, donde además se conocen todas sus derivadas, adicionalmente se considera $p(x)$ como el polinomio interpolante de $f(x)$. Además se tiene que el error absoluto del polinomio interpolante es: $\varepsilon = f(x) - p_n(x)$ donde $a \leq n \leq b$.

Según la definición de error para la interpolación de Lagrange tenemos que:

$$\varepsilon = \frac{f^{n+1}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n)$$

De acuerdo a esto podemos decir que:

$$f(x) - p_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n)$$

Además consideramos que $f^{n+1}\xi(x)$ tiene una cota máxima que se denominara M:

$$max_{a\leq x\leq b}|f^{n+1}\xi(x)| = M$$

Teniendo en cuenta lo anterior podemos concluir que:

$$f(x) - p_n(x) \leq \frac{M}{(n+1)!}(x-x_0)(x-x_1)\dots(x-x_n) \quad \text{donde } a \leq x \leq b$$

Ahora teniendo en cuenta que $(x-x_0)(x-x_1)\dots(x-x_n)$ es un polinomio Mónico (el mayor de sus coeficientes es acompañado por un 1) de grado $(n+1)$. Se tiene que:

$$(x-x_0)(x-x_1)\dots(x-x_n) = \bar{T}_{n+1}(x) \quad \text{donde } a \leq x \leq b$$

Donde $\bar{T}_{n+1}(x)$ es un polinomio de Chebyshev que es derivado de los polinomios de Chebyshev $T_{n+1}(x)$por medio de la división del coeficiente 2^{n-1}, donde además este es el valor mínimo. Lo que nos genera:

$$f(x) - p_n(x) \leq \frac{M}{(n+1)!}\bar{T}_{n+1}(x) \quad \text{donde } a \leq x \leq b$$

Ahora, debido a que $max_{a\leq x\leq b}|\bar{T}_{n+1}(x)| = 2^{-n}$. esto también implica que:

$$f(x) - p_n(x) \leq \frac{M}{(n+1)!}\frac{1}{2^n}$$

Del mismo modo consideramos un $\bar{x}_{k+1}$ que nos ayudara a encontrar el máximo valor de $|(x-x_0)(x-x_1)\dots(x-x_n)|$.

$$\bar{x}_{k+1} = cos\left(\frac{2(n-i)+1}{2n+2}\pi\right) donde\ i = 1, 2, \dots, n$$

Por lo que se puede concluir que:

$$f(x^*) - p_n(x^*) \leq \frac{M}{(n+1)!}\frac{1}{2^n}\ si\ \left|f^{n+1}(x)\right| \leq M\ para\ a \leq x \leq b$$

6. Utilice el polinomio de Taylor para interpolar $f(x) = e^x,\ x_0$=0 y $f(x) = \frac{1}{2}$.

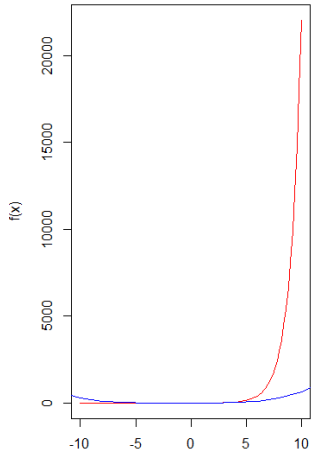
a. Implemente un código en R para la solución del problema con 5 cifras.

```
require(pracm
a)
```

```
f = function(x) exp(x)
g= function (x) 1/x
p = taylor(f, 0, 4) # Polinomio de Taylor de orden 4, alrededor
dex0=0.
pa= taylor(g, 5, 4) # esta funcion no se puede aproximar en x0=0
debido a que al evaluar este punto la funcion es indeterminada
pa1=taylor(g,-5,4)
curve(f, col= "red", from = -10, to= 10)
curve(p[1]*x^(4)+p[2]*x^(3)+p[3]*x^(2)+p[4]*x+p[5],add=TRUE,
col="blue", from = -100, to= 100)
curve(g, col= "green", from = -10, to= 10)
curve(pa[1]*x^(4)+pa[2]*x^(3)+pa[3]*x^(2)+pa[4]*x+pa[5],add=TRUE,
col="red", from = -100, to= 100 )
curve(pa1[1]*x^(4)+pa1[2]*x^(3)+pa1[3]*x^(2)+pa1[4]*x+pa1[5],add=TRUE
, col="red", from = -100, to= 100 )


cat(p[1],"x^(4)+",p[2],"*x^(3)+",p[3],"*x^(2)+",p[4],"x+",p[5],"/n")
#polinomio para la parte positiva
cat(pa[1],"x^(4)+",pa[2],"*x^(3)+",pa[3],"*x^(2)+",pa[4],"x+",pa[5],"
/n")
#polinomio para la parte negativa
cat(pa1[1],"x^(4)+",pa1[2],"*x^(3)+",pa1[3],"*x^(2)+",pa1[4],"x+",pa1
[5],"/n")
```

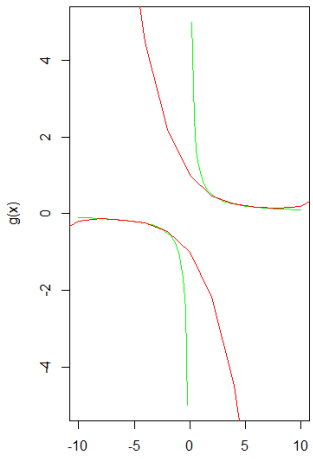
b. Escriba el polinomio resultante en cada caso

- para $f(x) = e^x$: $0.04166657x^4 + 0.1666667\,x^3 + 0.5x^2 + 1x + 1$



- para $g(x) = \frac{1}{x}$ $(es\ necesario\ dividir\ la\ aproximacion\ en\ 2, debido\ a\ que\ el\ punto\ es\ centrado\ en\ 0\ lo\ cual\ no\ genera\ una\ division\ por\ 0\ lo\ que\ genera\ un\ valor\ indeterindo)$:

1. en su parte positiva:
   $0.0003202768\ x^4$-$0.008005558x^3$+ $0.08004186x^2$ -$0.4001401x$+1.000176
2. En su parte negativa:
   -$0.0003203398\ x^4$+ -$0.00800682\ x^3$-$0.08005132\ x^2$+ -$0.4001716\ x$-1.000215

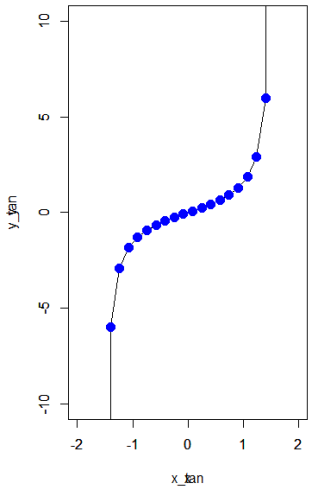


c. Debido a que el polinomio es de grado 4 da una aproximacion acertada hasta cierto punto, ademas hay que tener en cuenta el punto en el que va a estar centrada el polinomio.

7. Se desea aproximar la función $tan(x)$ en el intervalo $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$.

```
require(PolynomF)
                    require(pracma)
                    rm(list=ls())
```

```r
ft <- function(x)
{
  y0 = c()
  it = 1
  while (it <= length(x))
  {
    y0[it] = tan(x[it])
    it = it + 1
  }
  return(y0)
}


x = seq(-pi/2, pi/2,length = 20 )
y = ft(x)


polinomio = poly.calc(x,y)
polinomio




x_tan = seq(-pi/2,pi/2,length = 20)
y_tan = tan(x_tan)


plot(x_tan,y_tan,type="l",xlim=c(-2,2),ylim=c(-10,10))
par(new=TRUE)


plot(x,y, pch = 19, cex=1.5, col= "blue",xlim=c(-2,2),ylim=c(-
10,10))
par(new=TRUE)
```


y_tan
x_tan
10
5
0
-5
-10
-2
-1
0
1
2