Ejercicios

Uriel Paluch

16/10/2021

Guía Práctica 4: Interpolación

```
PolinomioLagrange <- function(x, fx, y){

  n <- length(x)

  l <- rep("", times = n)

  resultado <- 0

  for (i in 1:n) {
    l[i] <- fx[i]
    for (j in 1:n) {
      if (j != i){
        l[i] <- l[i] + glue::glue("*(x-",x[j],")/(",x[i],"-",x[j],")")
      }
    }
  }

  for(i in 1:n){
    resultado <- resultado + eval(parse(text=l[i]), y)
  }
  return(paste("El resultado es: ", resultado))
}
```

```
# Metodo de Neville -------------------------------------------------------

Neville <- function(x, y, interpolar){
  #cantidad de iteraciones que voy a hacer
  n <- length(x)-1

  #Hago un vector vacio para llenar el df
  empty_vec <- rep(0, times = length(x))

  df <- data.frame(x, y)

  for (i in 1:n) {
    df[glue::glue("Q",i)] <- empty_vec

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

      df[j, (i+2)] <- ( (interpolar-x[(j-i)]) * df[j,(i+1)] - (interpolar-x[j]) * df[(j-1),(i+1)] )  /
```

```
    }
  }

  return(df)
}
```

```
# Metodo de diferencias divididas ----------------------------------------
DiferenciasDivididas <- function(x, y){
  n <- length(x)

  #Hago un vector vacio para llenar el df
  empty_vec <- rep(0, times = n)

  df <- data.frame(x, y)

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

    df[glue::glue("Q",i)] <- empty_vec

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

      df[j, (i+2)] <- ( df[j,(i+1)] - df[(j-1),(i+1)])/(x[j]-x[j-i])
    }
  }

  return(df)
}
```

```
PolinomioInterpolanteNewton <- function(x, y){
  df <- DiferenciasDivididas(x = x, y = y)

  #Saco la primer columna del df
  df[,1] <- NULL

  n <- ncol(df)

  polinomio <- df[1,1]

  for (i in 2:n) {
    polinomio <- polinomio + glue::glue(" + ", df[i,i])
    for (j in 1:(i-1)) {
      polinomio <- polinomio + glue::glue(" * ( x - ", x[j], " )")
    }
  }
  return(polinomio)
}
```

```
TrazadorCubicoNatural = function(x,y){
  n = length(y)
  j = n - 1

  a = y
  b = c(rep(NA,n))
  c = c(rep(NA,n))
```

```r
d = c(rep(NA,n))

A = c(rep(NA,n))
h = c(rep(NA,n))
l = c(rep(NA,n))
u = c(rep(NA,n))
z = c(rep(NA,n))

#Paso 1
for (i in 1:j) {
  h[i] = x[i + 1] - x[i]
}


#Paso 2
for (i in 1:j) {
  if(i != 1){
    A[i] = (3 * (a[i + 1] - a[i])/(h[i])) - (3 * (a[i] - a[i - 1]) /h[i - 1])
  }
}


#Paso 3
l[1] = 1
u[1] = 0
z[1] = 0

#Paso 4
for (i in 2:j) {
  l[i] = 2 * (x[i + 1] - x[i - 1]) - h[i - 1] * u[i - 1]
  u[i] = h[i]/l[i]
  z[i] = (A[i] - h[i - 1] * z[i - 1])/l[i]
}

#Paso 5
l[n] = 1
z[n] = 0
c[n] = 0

#Paso 6
for (i in j:1) {
  c[i] = z[i] - u[i] * c[i + 1]
  b[i] = (a[i + 1] - a[i])/h[i] - h[i] * (c[i + 1] + 2*c[i])/3
  d[i] = (c[i + 1] - c[i])/(3*h[i])
}

#Paso 7
results = matrix(rep(NA, 4*j), nrow = j, ncol = 4, byrow = F)
for (k in 1:j) {
  results[k, 1] = a[k]
  results[k, 2] = b[k]
  results[k, 3] = c[k]
  results[k, 4] = d[k]
```

```
  }

  #Construyo el polinomio
  polinomios <- rep(NA, times = nrow(results))
  for (i in 1:nrow(results)) {
    polinomios[i] <- glue::glue(results[i,1])
    for(j in 2:ncol(results)){
      polinomios[i] <- polinomios[i] + glue::glue(" + ", results[i,j], " * (x - ", x[i], ")^", (j-1))
    }
  }

  return(polinomios)

}
```

Ejercicio 1:

Use los polinomios interpolantes de Lagrange apropiados de grado uno, dos y tres para aproximar lo siguiente:

```
# x es una lista con todos los valores
# fx es la funcion que hay que aproximar
# y es el valor donde se desea aproximar la función
polinomioLagrangeGrado3 <- PolinomioLagrange(x = c(0, 0.25, 0.5, 0.75), fx = c(1, 1.64872, 2.71828, 4.48
print(polinomioLagrangeGrado3)
```

a. f (0.43) si f (0) = 1, f (0.25) = 1.64872, f (0.5) = 2.71828, f (0.75) = 4.48169

```
## [1] "El resultado es:  2.36060473408"
```

```
polinomioLagrangeGrado2 <- PolinomioLagrange(x = c(0, 0.25, 0.5), fx = c(1, 1.64872, 2.71828), y = list
print(polinomioLagrangeGrado2)
```

```
## [1] "El resultado es:  2.376382528"
```

```
polinomioLagrangeGrado2 <- PolinomioLagrange(x = c(0.25, 0.5, 0.75), fx = c(1.64872, 2.71828, 4.48169),
print(polinomioLagrangeGrado2)
```

```
## [1] "El resultado es:  2.34886312"
```

```
#Este es el que va
#Creo que es porque es el intervalo mas pequeño
polinomioLagrangeGrado1 <- PolinomioLagrange(x = c(0.25, 0.5), fx = c(1.64872, 2.71828), y = list(x = 0
print(polinomioLagrangeGrado1)
```

```
## [1] "El resultado es:  2.4188032"
```

```
polinomioLagrangeGrado3 <- PolinomioLagrange(x = c(-0.5, -0.25, 0.25, 0.5), fx = c(1.93750, 1.33203, 0.8
print(polinomioLagrangeGrado3)
```

b. f (0) si f (-0.5) = 1.93750, f (-0.25) = 1.33203, f (0.25) = 0.800781, f (0.5) = 0.687500

```
## [1] "El resultado es:  0.984374"
```

```
polinomioLagrangeGrado2 <- PolinomioLagrange(x = c(-0.5, -0.25, 0.25), fx = c(1.93750, 1.33203, 0.80078
print(polinomioLagrangeGrado2)
```

```
## [1] "El resultado es:  0.953123666666667"
```

```
polinomioLagrangeGrado2 <- PolinomioLagrange(x = c(-0.25, 0.25, 0.5), fx = c(1.33203, 0.800781, 0.687500
print(polinomioLagrangeGrado2)
```

```
## [1] "El resultado es:  1.01562433333333"
```

```
polinomioLagrangeGrado1 <- PolinomioLagrange(x = c(-0.25, 0.25), fx = c(1.33203, 0.800781), y = list(x =
print(polinomioLagrangeGrado1)
```

```
## [1] "El resultado es:  1.0664055"
```

```
polinomioLagrangeGrado3 <- PolinomioLagrange(x = c(0.1, 0.2, 0.3, 0.4), fx = c(-0.29004986, -0.56079734
print(polinomioLagrangeGrado3)
```

c. f (0.18) sif (0.1) = -0.29004986, f (0.2) = -0.56079734, f (0.3) = -0.81401972, f (0.4) =-1.0526302

```
## [1] "El resultado es:  -0.5081430744"
```

```
polinomioLagrangeGrado2 <- PolinomioLagrange(x = c(0.1, 0.2, 0.3), fx = c(-0.29004986, -0.56079734, -0.8
print(polinomioLagrangeGrado2)
```

```
## [1] "El resultado es:  -0.508049852"
```

```
polinomioLagrangeGrado1 <- PolinomioLagrange(x = c(0.1, 0.2), fx = c(-0.29004986, -0.56079734), y = list
print(polinomioLagrangeGrado1)
```

```
## [1] "El resultado es:  -0.506647844"
```

Ejercicio 2:

Escribir las tablas de diferencias divididas asociadas al Ejercicio 1.

```
print(DiferenciasDivididas(x = c(0, 0.25, 0.5, 0.75), y = c(1, 1.64872, 2.71828, 4.48169)))
```

Ejercicio a:

```
##      x       y      Q1      Q2       Q3
## 1 0.00 1.00000 0.00000 0.00000 0.000000
## 2 0.25 1.64872 2.59488 0.00000 0.000000
## 3 0.50 2.71828 4.27824 3.36672 0.000000
## 4 0.75 4.48169 7.05364 5.55080 2.912107
```

```
print(DiferenciasDivididas(x = c(-0.5, -0.25, 0.25, 0.5), y = c(1.93750, 1.33203, 0.800781, 0.687500)))
```

Ejercicio b:

```
##       x        y        Q1        Q2        Q3
## 1 -0.50 1.937500  0.000000 0.0000000  0.000000
## 2 -0.25 1.332030 -2.421880 0.0000000  0.000000
## 3  0.25 0.800781 -1.062498 1.8125093  0.000000
## 4  0.50 0.687500 -0.453124 0.8124987 -1.000011
```

```
print(DiferenciasDivididas(x = c(0.1, 0.2, 0.3, 0.4), y = c(-0.29004986, -0.56079734, -0.81401972, -1.05
```

Ejercicio c:

```
##     x          y        Q1       Q2         Q3
## 1 0.1 -0.2900499  0.000000 0.000000  0.0000000
## 2 0.2 -0.5607973 -2.707475 0.000000  0.0000000
## 3 0.3 -0.8140197 -2.532224 0.876255  0.0000000
## 4 0.4 -1.0526302 -2.386105 0.730595 -0.4855333
```

Ejercicio 3:

Escriba el polinomio Pn(x) del ejercicio 1 utilizando la "Fórmula de diferencias divididas Interpolantes de Newton", siendo "n" el grado másgrande de aproximación.

```r
print(PolinomioInterpolanteNewton(x = c(0, 0.25, 0.5, 0.75), y = c(1, 1.64872, 2.71828, 4.48169)))
```

a:

```
## 1 + 2.59488 * ( x - 0 ) + 3.36672 * ( x - 0 ) * ( x - 0.25 ) + 2.91210666666667 * ( x - 0 ) * ( x - 
```

```r
print(PolinomioInterpolanteNewton(x = c(-0.5, -0.25, 0.25, 0.5), y = c(1.93750, 1.33203, 0.800781, 0.68
```

b:

```
## 1.9375 + -2.42188 * ( x - -0.5 ) + 1.81250933333333 * ( x - -0.5 ) * ( x - -0.25 ) + -1.00001066666
```

```r
print(PolinomioInterpolanteNewton(x = c(0.1, 0.2, 0.3, 0.4), y = c(-0.29004986, -0.56079734, -0.8140197
```

c:

```
## -0.29004986 + -2.7074748 * ( x - 0.1 ) + 0.876255000000001 * ( x - 0.1 ) * ( x - 0.2 ) + -0.48553333
```

Ejercicio 4:

Aplique el método de Neville para obtener las aproximaciones del ejercicio 1.

```r
# x es la preimagen
# y es la imagen
# interpolar es el número que se desea interpolar
print(Neville(x = c(0, 0.25, 0.5, 0.75), y = c(1, 1.64872, 2.71828, 4.48169), interpolar = 0.43))
```

Ejercicio a:

```
##      x       y       Q1       Q2       Q3
## 1 0.00 1.00000 0.000000 0.000000 0.000000
## 2 0.25 1.64872 2.115798 0.000000 0.000000
## 3 0.50 2.71828 2.418803 2.376383 0.000000
## 4 0.75 4.48169 2.224525 2.348863 2.360605
```

```r
# x es la preimagen
# y es la imagen
# interpolar es el número que se desea interpolar
print(Neville(x = c(-0.5, -0.25, 0.25, 0.5), y = c(1.93750, 1.33203, 0.800781, 0.687500), interpolar = 
```

Ejercicio b:

```
##       x        y       Q1        Q2       Q3
## 1 -0.50 1.937500 0.000000 0.0000000 0.000000
## 2 -0.25 1.332030 0.726560 0.0000000 0.000000
## 3  0.25 0.800781 1.066406 0.9531237 0.000000
## 4  0.50 0.687500 0.914062 1.0156243 0.984374
```

```
# x es la preimagen
# y es la imagen
# interpolar es el número que se desea interpolar
print(Neville(x = c(0.1, 0.2, 0.3, 0.4), y = c(-0.29004986, -0.56079734, -0.81401972, -1.0526302), inte
```

Ejercicio c:

```
##     x          y         Q1         Q2         Q3
## 1 0.1 -0.2900499  0.0000000  0.0000000  0.0000000
## 2 0.2 -0.5607973 -0.5066478  0.0000000  0.0000000
## 3 0.3 -0.8140197 -0.5101529 -0.5080499  0.0000000
## 4 0.4 -1.0526302 -0.5276871 -0.5083994 -0.5081431
```

Ejercicio 5:

Escriba los trazadores Si(x) mediante el método de Trazadores Cúbicos Naturalespara los datos del ejercicio 1.b., indicando claramente los intervalos de x en los cuales se debe utilizar cada uno.

```
print(TrazadorCubicoNatural(x = c(-0.5, -0.25, 0.25, 0.5), y = c(1.93750, 1.33203, 0.800781, 0.687500)))
```

```
## [1] "1.9375 + -2.63867825 * (x - -0.5)^1 + 0 * (x - -0.5)^2 + 3.468772 * (x - -0.5)^3"
## [2] "1.33203 + -1.9882835 * (x - -0.25)^1 + 2.601579 * (x - -0.25)^2 + -1.500016 * (x - -0.25)^3"
## [3] "0.800781 + -0.5117165 * (x - 0.25)^1 + 0.351555000000001 * (x - 0.25)^2 + -0.468740000000001 *
```

1: (-0.5;-0.25)
2: (-0.25;0.25)
3: (0.25;0.5) s