moodle_02_02-06-16-52

Alan Coila Bustinza

2022-06-02

```
library(knitr)     # For knitting document and include_graphics function
library(ggplot2)   # For plotting
library('png')
```

pregunta 1

```
img1_path <- "p1_2022-06-02_163951.png"
include_graphics(img1_path)
```

Sabemos que cierta función pasa por los siguientes puntos:

$$\begin{bmatrix} x_i & 0 & 1 & 3 & 4 & 6 \\ f(x_i) & -1 & 1.143 & 1.272 & -0.86 & -1.519 \end{bmatrix}$$

Usando el polinomio interpolador de Lagrange, da el valor aproximado de la función en el punto $x = 7$

```
x1 <- c(0,1,3,4,6)
y1 <- c(-1,1.143,1.272,-0.86,-1.519)
n <- length(x1)-1
myPhi <- function(x, n) {
  Phi <- matrix(1, length(x), n + 1)
  for (i in 1:n) {
    Phi[, i + 1] <- x^i
  }
  return(Phi)
}

mylssolve <- function(A, y) {
  AT <- t(A)
  return((solve(AT %*% A)) %*% AT %*% y)
}

mypolyfit <- function(x, y, n) {
  Phi <- myPhi(x, n)
  c <- mylssolve(Phi, y)
  return(c)
}
```

```
coef <- mypolyfit(x1,y1,n)

myeval <- function(x, c) {
  f <- 0
  for (i in 1:length(c)) {
    f <- f + c[i] * x^(i - 1)
  }
  return(f)
}
myeval(7,coef)
```

```
## [1] 6.4704
```

pregunta 2

```
img1_path <- "p2_2022-06-02_164145.png"
include_graphics(img1_path)
```

Usando el método de Romberg, queremos calcular

$$\int_0^1 \frac{dx}{1+x}.$$

Una parte del esquema triangular toma la siguiente forma:

Nivel 0	*Nivel* 1	...
0.75000000		
0.70833333	0.69444444	
0.69702381	0.69325397	α
0.69412185	0.69315453	0.69314790
0.69339120	0.69314765	0.69314719

Entonces, el valor de α y la aproximación de la integral es:

```
# primero definir la funcion
f2 <- function(x){
  return(1/(1+x))
}
# luego formula del trapecio compuesta para funcion f
trap=function(f,a,b,m)
{
  x=seq(a,b,length.out=m+1)
  y=f(x)
  p.area=sum((y[2:(m+1)]+y[1:m]))
  p.area=p.area*abs(b-a)/(2*m)
  return(p.area)
```

```r
}
# para generar el primer nivel 0 ver el numero de valores en nivel 0:

n0 <- function(valores,f,a,b){

  n=c()
  for(i in 2^(0:(valores-1))){
    n=c(n,trap(f2,0,1,i))
  }
  return(n)
}
v0 <- n0(5,f2,0,1)
k=length(v0)
nk <- function(ncero){

  mat <- matrix(0, length(ncero), k)
  mat[,1] <- ncero
  # ((4*n1[i+1]-n1[i])/(4-1))
  # for(i in 2:k){
  #   a=i
  #   b=i+1
  #   mat[,k] <- 4**(k-1)*mat[k,k-1]
  # }
  return (mat)
}
mat <- nk(v0)

for(i in 1:k){
  vect=c()
  for(j in 1:(k-i)){
    if(j>0){
      a=mat[j+1,i]
      b=mat[j,i]
      pot= 4**i
      vect <- c(vect ,(pot*a-b)/(pot-1))
    }
  }
  if(i<k){
     mat[,i+1] <- vect[1:k]
  }
}

mat
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.7500000 0.6944444 0.6931746 0.6931475 0.6931472
## [2,] 0.7083333 0.6932540 0.6931479 0.6931472        NA
## [3,] 0.6970238 0.6931545 0.6931472        NA        NA
## [4,] 0.6941219 0.6931477        NA        NA        NA
## [5,] 0.6933912        NA        NA        NA        NA
```

pregunta 3

```r
img1_path <- "p3_2022-06-02_164349.png"
include_graphics(img1_path)
```

Si queremos calcular un polinomio que pase por una sucesión de puntos, pero sabemos que en el futuro obtendremos más puntos por los que tendrá que pasar el polinomio, ¿qué método nos conviene utilizar para inteprolar?

Seleccione una:

- a. El método de Lagrange.
- b. Es indiferente usar un método o el otro.
- c. El método de Newton.

pregunta 4

```r
img1_path <- "p4_2022-06-02_164432.png"
include_graphics(img1_path)
```

Un río tiene una anchura de $\frac{25}{36}$m. En función de la distancia hasta la orilla, la profundidad del río, medida en sección recta, viene dada por la tabla siguiente:

$$\begin{bmatrix} \mathit{Distancia} & 0 & \frac{1}{36} & \frac{1}{4} & \frac{7}{18} & \frac{5}{9} & \frac{11}{18} & \frac{25}{36} \\ \mathit{Profundidad} & 1 & \frac{180}{179} & \frac{20}{19} & \frac{90}{83} & \frac{9}{8} & \frac{90}{79} & \frac{36}{31} \end{bmatrix}$$

Usando la regla del trapecio compuesta, dad una aproximación del area de la sección.

```r
x4 <- c(0,1/36,1/4,7/18,5/9,11/18,25/36)
y4 <- c(1,180/179,20/19,90/83,9/8,90/79,36/31)

polyinterp=function(x,y)
{
  # comprobamos que la longitud de los vectores sea la misma
  if(length(x)!=length(y))
    stop ("La longitud de los vectores x e y debe ser la misma" )
  # calculamos el valor de n que es el grado del polinomio
  # a partir del numero de puntos menos 1
  n=length(x)-1
  # creamos la primera columna de la matriz de vandermonde
  vandermonde=rep(1,length(x))
```

```r
  # iteramos para ir agregando las columnas siguientes según
  # el grado del polinomio
  for(i in 1:n)
  {
    # la matriz contiene columnas sucesivas de los valores de x
    # elevados a la nth potencia
    xi=x^i
    vandermonde=cbind(vandermonde,xi)
  }
  # resolvemos el sistema de ecuaciones
  beta=solve(vandermonde,y, tol=1e-22)
  # borramos los nombres de las columnas (xi)
  names(beta)=NULL
  # nos retorna los coeficientes del polinomio
  return(beta)
}


myeval <- function(x, c) {
  f <- 0
  for (i in 1:length(c)) {
    f <- f + c[i] * x^(i - 1)
  }
  return(f)
}

coef <- polyinterp(x4,y4)
m=9
a=x4[1]
b=x4[length(x4)]

trap=function(a,b,m)
{

  x=seq(a,b,length.out=m+1)
  y=myeval(x, coef)
  p.area=sum((y[2:(m+1)]+y[1:m]))
  p.area=p.area*abs(b-a)/(2*m)
  return(p.area)
}
trap(a,b,m)
```

```
## [1] 0.7476933
```