Preguntas de R

Juan Cardona

Abril 2020

1 Aproximación por sumas parciales

$$S = \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6} \approx 1.64493$$

Definimos la función que queremos sumar, pero como la queremos aproximar por sumas parciales, decidimos el límite de cada suma parcial S_x el cual al hacerlo tender a infinito convergerá al valor arriba expuesto, por ejemplo:

```
f <- function(i){ 1/(i**2)}
x<-10
sum(f(1:(x)))

## [1] 1.549768
```

```
f <- function(i){ 1/(i**2)}
x<-100
sum(f(1:(x)))

## [1] 1.634984
```

```
f <- function(i){ 1/(i**2)}
x<-1000
sum(f(1:(x)))

## [1] 1.643935
```

2 Valor aproximado de la medida usando una partición regular

$$\int_0^{\infty} e^{-x^2} dx \approx$$

Inicialmente definimos la función que vamos a integrar, luego para aplicarle dicho operador definimos la función que integra en RStudio, donde únicamente reemplazamos los límites de la integral y obtenemos como resultado:

```
f <- function(x){ exp(-x^2)}
lower <- 0
upper <- 1
integrate(f, lower, upper)

## 0.7468241 with absolute error < 8.3e-15
```

Por otro lado, en la integral de Riemann, hay una división de intervalos, al parecer esta función ya la tiene implícita, pues si escribimos los intervalos nos da siempre lo mismo

```
f <- function(x){ exp(-x^2)}
lower <- 0
upper <- 1
integrate(f, lower, upper, subdivisions = 10L)

## 0.7468241 with absolute error < 8.3e-15
```

3 Criba de Eratóstenes

Debido a las dificultades con este ejercicio, me enfoqué en entender el pseudo código detrás del algoritmo. Para mayor información revisar *Implementación de algoritmos de teoría de números/Criba de Eratóstenes* [1]. Este algoritmo permite generar una lista de números primos menores a un número natural dado, y funciona quitando los números que no son primos, así pues, si hay un número que no está eliminado, se eliminan todos sus múltiplos.
Inicialmente definimos una función de posibles números primos, que va desde 2 hasta n y los recorre uno a uno. Entonces el primer número primo es 2, luego se eliminan todos sus múltiplos hasta la parte entera piso de raíz de n. Luego continuamos con el 3 y así sucesivamente

```
primos<-function(n){
  posibles<-seq(1,n,by=2)
  posibles[1]<-2
  for(i in 2:round(sqrt(n))){
    if(posibles[i]!=0){
      for(j in seq(i + posibles[i], n/2, by=posibles[i])){
        posibles[j]<-0
      }
    }
  }
```

```
  return(posibles[posibles!=0])
}
primos(100)
```

```
##  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

4 Discriminante diferente de cero

```
cat("Raíces de a*x**2+b*x +c.\n")
```

```
## Raíces de a*x**2+b*x +c.
```

```
# Lectura de coeficientes desde el teclado (ENTER)
a = 1  #as.numeric(readline("a = ")) # Leer desde el teclado (en la consola)
b = 2  #as.numeric(readline("b = "))
c = 1  #as.numeric(readline("c = "))
# Cálculos y criterio de parada
if(a==0) stop("No es cuadrática")
# cálculo del Discriminante
d = b**2-4*a*c
# ---
if(d==0){
  x1 = -b/(2*a)
  cat("Una raíz real x1 = ", x1)
}
```

```
## Una raíz real x1 =  -1
```

```
if(d>0){
  x1 = (-b+sqrt(d))/(2*a)
  x2 = (-b-sqrt(d))/(2*a)
  cat("Dos raíces reales \n", "x1 = ", x1, "\n", "x2 = ", x2)
}
if(d<0) cat("Las raíces son complejas")
```

Gracias a mi compañero Santiago Orjuela, para solucionar el problema del discriminante igual a cero descargamos el paquete de R llamado "Rmpfr", el cual si encuentra una cadena de dígitos en donde hay ceros, muestra los números hasta el numero anterior al cero.

```
install.packages("Rmpfr")
```

```
## Installing package into '/usr/local/lib/R/site-library'
## (as 'lib' is unspecified)
```

```
## Warning in install.packages("Rmpfr"):  'lib = "/usr/local/lib/R/site-library"'
is not writable
## Error in install.packages("Rmpfr"):  unable to install packages

library(Rmpfr)

## Error in library(Rmpfr):  there is no package called 'Rmpfr'

cat("Raíces de a*x**2+b*x +c.\n")

## Raíces de a*x**2+b*x +c.

# Lectura de coeficientes desde el teclado (ENTER)
a = 1  #as.numeric(readline("a = ")) # Leer desde el teclado (en la consola)
b = 2  #as.numeric(readline("b = "))
c = 1  #as.numeric(readline("c = "))
# Cálculos y criterio de parada
if(a==0) stop("No es cuadrática")
# cálculo del Discriminante
d = mpfr(b**2-4*a*c,53)

## Error in mpfr(b^2 - 4 * a * c, 53):  could not find function "mpfr"

# ---
if(d==0){
  x1 = -b/(2*a)
  cat("Una raíz real x1 = ", x1)
}

## Una raíz real x1 =  -1

if(d>0){
  x1 = (-b+sqrt(d))/(2*a)
  x2 = (-b-sqrt(d))/(2*a)
  cat("Dos raíces reales \n", "x1 = ", x1, "\n", "x2 = ", x2)
}
if(d<0) cat("Las raíces son complejas")
```

5 Método de Jacobi

```
A <- matrix( c(5, 1, 0,
               3,-1, 2,
               4, 0,-1), nrow=3, byrow=TRUE)

B <- matrix( c(5, 1, 0), nrow=3)
```

```
U <- triu(A, 1)

## Error in triu(A, 1):  could not find function "triu"

L <- tril(A, -1)

## Error in tril(A, -1):  could not find function "tril"

C <- L + U

## Error in eval(expr, envir, enclos):  object 'L' not found

D <- A-U-L

## Error in eval(expr, envir, enclos):  object 'U' not found

D1 <- solve(D)

## Error in as.vector(x, mode):  cannot coerce type 'closure' to vector
of type 'any'

Tj <- D1%*%C

## Error in eval(expr, envir, enclos):  object 'D1' not found

a <- eigen(A)

b <- eigen(Tj)

## Error in as.matrix(x):  object 'Tj' not found

solve(A,B)

##       [,1]
## [1,] 0.375
## [2,] 3.125
## [3,] 1.500
```

En cuanto a la solución, podemos ver en las siguientes imágenes los resultados en las figuras 1 y 2

Finalmente para la solución del sistema, también es posible usar la función "lsolve.jacobi" que viene del paquete "Rlinsolve" y como resultado nos da una solución más exacta pero parecida a la función "solve(A,B)". Todo lo anterior en la figura 3.

```
lsolve.jacobi(A, B, xinit = NA, reltol = 1e-05, maxiter = 1000,
              weight = 2/3, adjsym = TRUE, verbose = TRUE)

## Error in lsolve.jacobi(A, B, xinit = NA, reltol = 1e-05, maxiter
= 1000, :  could not find function "lsolve.jacobi"
```

```
> A
     [,1] [,2] [,3]
[1,]    5    1    0
[2,]    3   -1    2
[3,]    4    0   -1
> B
     [,1]
[1,]    5
[2,]    1
[3,]    0
> U
3 x 3 Matrix of class "dtrMatrix"
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    .    0    2
[3,]    .    .    0
> L
3 x 3 Matrix of class "dtrMatrix"
     [,1] [,2] [,3]
[1,]    0    .    .
[2,]    3    0    .
[3,]    4    0    0
> C
3 x 3 Matrix of class "dgeMatrix"
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    3    0    2
[3,]    4    0    0
> D
3 x 3 Matrix of class "dgeMatrix"
     [,1] [,2] [,3]
[1,]    5    0    0
[2,]    0   -1    0
[3,]    0    0   -1
> D1
     [,1] [,2] [,3]
[1,]  0.2    0    0
[2,]  0.0   -1    0
[3,]  0.0    0   -1
```

Figure 1: Matrices

```
> Tj
3 x 3 Matrix of class "dgeMatrix"
     [,1] [,2] [,3]
[1,]    0  0.2    0
[2,]   -3  0.0   -2
[3,]   -4  0.0    0
> a
eigen() decomposition
$values
[1]  5.633994+0.000000i -1.316997+1.051391i -1.316997-1.051391i

$vectors
             [,1]                  [,2]                  [,3]
[1,] 0.7526027+0i  0.1326755+0.0220823i  0.1326755-0.0220823i
[2,] 0.4771454+0i -0.8613281+0.0000000i -0.8613281+0.0000000i
[3,] 0.4537856+0i -0.0624942-0.4859198i -0.0624942+0.4859198i

> b
eigen() decomposition
$values
[1] -0.5+1.161895i -0.5-1.161895i  1.0+0.000000i

$vectors
                       [,1]                  [,2]          [,3]
[1,] -0.0553509-0.1286239i -0.0553509+0.1286239i -0.1543033+0i
[2,]  0.8856149+0.0000000i  0.8856149+0.0000000i -0.7715167+0i
[3,]  0.3044301-0.3215598i  0.3044301+0.3215598i  0.6172134+0i

> solve(A,B)
      [,1]
[1,] 0.375
[2,] 3.125
[3,] 1.500
```

Figure 2: Matrices 2

```
$x
          [,1]
[1,] 0.3749713
[2,] 3.1253141
[3,] 1.5001937

$iter
[1] 32

$errors
              [,1]
 [1,] 2.774531e-01
 [2,] 8.385980e-02
 [3,] 2.684954e-02
 [4,] 1.031659e-02
 [5,] 5.264788e-03
 [6,] 3.452820e-03
 [7,] 2.572661e-03
 [8,] 2.014697e-03
 [9,] 1.606388e-03
[10,] 1.289066e-03
[11,] 1.036787e-03
[12,] 8.345561e-04
[13,] 6.719645e-04
[14,] 5.411049e-04
[15,] 4.357450e-04
[16,] 3.509045e-04
[17,] 2.825839e-04
[18,] 2.275656e-04
[19,] 1.832593e-04
[20,] 1.475793e-04
[21,] 1.188461e-04
[22,] 9.570721e-05
[23,] 7.707335e-05
[24,] 6.206744e-05
[25,] 4.998312e-05
[26,] 4.025158e-05
[27,] 3.241474e-05
[28,] 2.610370e-05
[29,] 2.102140e-05
[30,] 1.692861e-05
[31,] 1.363267e-05
[32,] 1.097844e-05
[33,] 8.840972e-06
```

Figure 3: Jacobi

References

[1] Wikilibros. *Implementación de algoritmos de teoría de números/Criba de Eratóstenes*. Wikibooks, 2020.