Tarea 1: R

Catalina Núñez
Gabriela Barbosa

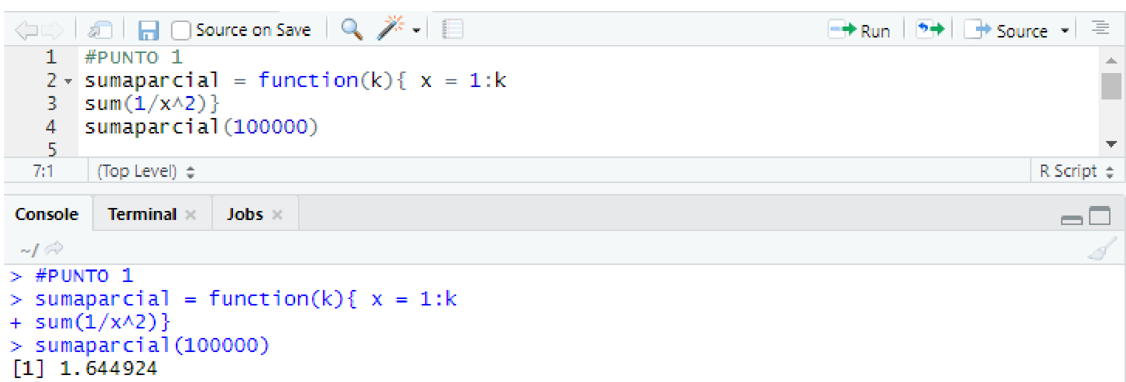
13 de Abril - 2020

1. Aproximación por suma parcial

Para éste problema vamos a utilizar las sumas parciales esto es

$$\sum_{i=1}^{k} \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + ... + \frac{1}{k^2}$$

Ahora, para implementar el código en Rstudio, se plantea de la siguiente manera utilizando vectores y crearemos una función:



2. Valor aproximado de la medida

Para encontrar el valor de la medida podemos construir una partición regular, utilizando la suma de los rectangulos bajo la curva de la siguiente manera:

```r
area1 = function(dx){
  xi = seq(0,1, by = dx) # = (0, dx, 2dx,...,1)
  sum(exp(-xi^2))*dx
}
area1(1/10000000)
```

```
7:1   (Top Level)

Console   Terminal   Jobs

~/
> area1 = function(dx){
+   xi = seq(0,1, by = dx) # = (0, dx, 2dx,...,1)
+   sum(exp(-xi^2))*dx
+ }
> area1(1/10000000)
[1] 0.7468242
>
>
```

Donde el área del rectángulo está dada por la base y la altura, descrita por esta ecuación.

3. Función que devuelve numeros primos

Es un algoritmo que permite hallar todos los números primos menores que un número natural n, mediante el siguiente proceso:

- Listar los valores de 2 a n
- Tomar el primer número no tachado o marcado como número primo y se marca como primo
- Tachar los múltiplos del número que se acaban de marcar como primo
- Si el cuadrado del número que se indicó como primo es igual o menor que n, se vuelve al paso 2. De lo contrario, termina el proceso.

Ahora el algoritmo se presenta a continuación:

```r
primos = function(x){
  numeros = 2:x
  d = 1
  while(numeros[d]^2 <= x){
    for(n in numeros){
      if(numeros[d] != n){
        if(n%%numeros[d]==0){
          numeros = numeros[-which(numeros==n)]
        }
      }
    }
    d=d+1
  }
  numeros
}

primos(200)
```

Lo cual da el resultado.

```
> primos(200)
 [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73
[22]  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181
[43] 191 193 197 199
>
```

4. Discriminante de una función cuadrática

Para este problema utilizaremos el paquete **Rmpfr** que nos permite ampliar los decimales para tomar en cuenta en las raíces de la ecuación cuadrática como en el siguiente ejemplo:

```
install.packages("Rmpfr")
require(Rmpfr)
aa = mpfr(45, precBits = 256)
bb = mpfr(100, precBits = 256)
cc = mpfr(2, precBits = 256)
x2= (-bb+sqrt(bb^2-4*aa*cc))/(2*aa);x2
x2= (-bb-sqrt(bb^2-4*aa*cc))/(2*aa);x2
```

```
> require(Rmpfr)
> aa = mpfr(45, precBits = 256)
> bb = mpfr(100, precBits = 256)
> cc = mpfr(2, precBits = 256)
> x2= (-bb+sqrt(bb^2-4*aa*cc))/(2*aa);x2
1 'mpfr' number of precision  256   bits
[1] -2.018331478812647213239891237366396231674257893481999622762218982320005892033204e-2
> x2= (-bb-sqrt(bb^2-4*aa*cc))/(2*aa);x2
1 'mpfr' number of precision  256   bits
[1] -2.202038907434095750089823309848558259905479643287402225994600032399022163301891
>
```

5. Dado un sistema $AX = B$, resolverlo por el método de Jacobi

Primero tomaremos una matriz A o simplemente un sistema de ecuaciones lineales.

```r
rm(list=ls())
n = as.numeric(readline("n = "))
x = scan()
x
A = matrix(x, nrow = n, ncol = n, byrow=T )
A
```

```
> rm(list=ls())
> n = as.numeric(readline("n = "))
n = 4
> x = scan()
1: 3
2: -2
3: -1
4: -1
5: -2
6: -1
7: 2
8: 1
9: 5
10: -2
11: -2
12: -1
13: 1
14: 1
15: 2
16: 5
17: 
Read 16 items
> x
 [1]  3 -2 -1 -1 -2 -1  2  1  5 -2 -2 -1  1  1  2  5
> A = matrix(x, nrow = n, ncol = n, byrow=T )
> A
     [,1] [,2] [,3] [,4]
[1,]    3   -2   -1   -1
[2,]   -2   -1    2    1
[3,]    5   -2   -2   -1
[4,]    1    1    2    5
> 
```

- Luego desagregamos la matriz A como la suma de una matriz D (diagonal de A) + una matriz triangular superior U + una matriz triangular inferior L

```r
D = diag(diag(A))
D
# M es la matriz A, restandole la diagonal D
M = A - D
M
# U es la matriz triangular superior
U = {
  lower.tri(M)
  M[lower.tri(M)] <- 0
  M}
print(U)

M = A - D
# L es la matriz triangular inferior
L = {
  upper.tri(M)
  M[upper.tri(M)] <- 0
  M}
print(L)
```

```
> D = diag(diag(A))
> D
     [,1] [,2] [,3] [,4]
[1,]    3    0    0    0
[2,]    0   -1    0    0
[3,]    0    0   -2    0
[4,]    0    0    0    5
> M = A - D
> M
     [,1] [,2] [,3] [,4]
[1,]    0   -2   -1   -1
[2,]   -2    0    2    1
[3,]    5   -2    0   -1
[4,]    1    1    2    0
> U = {
+   lower.tri(M)
+   M[lower.tri(M)] <- 0
+   M}
> print(U)
     [,1] [,2] [,3] [,4]
[1,]    0   -2   -1   -1
[2,]    0    0    2    1
[3,]    0    0    0   -1
[4,]    0    0    0    0
> M = A - D
> L = {
+   upper.tri(M)
+   M[upper.tri(M)] <- 0
+   M}
> print(L)
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]   -2    0    0    0
[3,]    5   -2    0    0
[4,]    1    1    2    0
>
```

- Determinamos la matriz de transición Tj, para ello necesitamos de la inversa de la diagonal I y las matrices triangulares U y L, respectivamente.
- Luego generamos una lista de autovalores y autovectores.

```
# I es la inversa de la diagonal D
I = 1/diag(D)
r = matrix(I ,n , n, byrow=T)
I = diag(diag(r))
I

#Matriz de transición
Tj = I %*% (L + U)
Tj


e <- eigen(A)
b = e$values
b

v = e$vectors
v
```

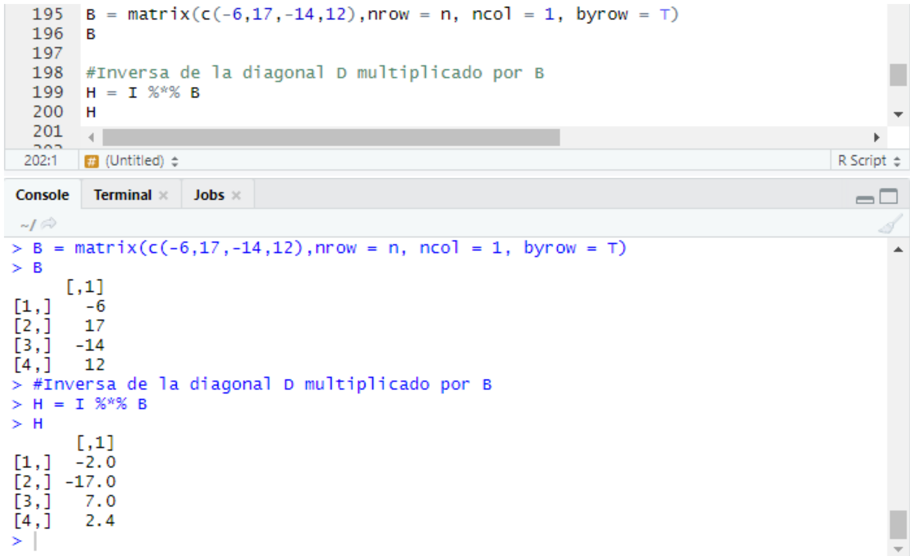
192:1 (Untitled) R Script

Console Terminal Jobs

~/

```
> # I es la inversa de la diagonal D
> I = 1/diag(D)
> r = matrix(I ,n , n, byrow=T)
> I = diag(diag(r))
> I
          [,1] [,2] [,3] [,4]
[1,] 0.3333333    0  0.0  0.0
[2,] 0.0000000   -1  0.0  0.0
[3,] 0.0000000    0 -0.5  0.0
[4,] 0.0000000    0  0.0  0.2
> #Matriz de transición
> Tj = I %*% (L + U)
> Tj
     [,1]       [,2]       [,3]       [,4]
[1,]  0.0 -0.6666667 -0.3333333 -0.3333333
[2,]  2.0  0.0000000 -2.0000000 -1.0000000
[3,] -2.5  1.0000000  0.0000000  0.5000000
[4,]  0.2  0.2000000  0.4000000  0.0000000
> e <- eigen(A)
> b = e$values
> b
[1]  3.5292512+1.365591i  3.5292512-1.365591i -1.9511338+0.000000i
[4] -0.1073686+0.000000i
> v = e$vectors
> v
                      [,1]                  [,2]           [,3]          [,4]
[1,]  0.3525983-0.2718491i  0.3525983+0.2718491i -0.33475904+0i  0.3473235+0i
[2,] -0.1299274+0.0053778i -0.1299274-0.0053778i -0.92549450+0i  0.3875501+0i
[3,]  0.4073717-0.3483843i  0.4073717+0.3483843i  0.01719884+0i  0.7364243+0i
[4,] -0.7053646+0.0000000i -0.7053646+0.0000000i  0.17635337+0i -0.4322622+0i
> 
```

- Ahora resolvemos el sistema, entonces tomamos un B particular, para hallar H, con $H = I * B$.



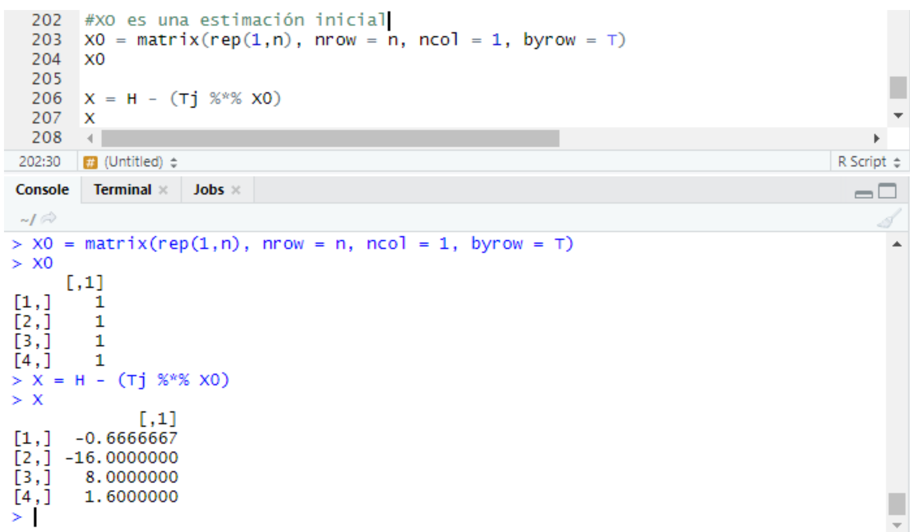
```
B = matrix(c(-6,17,-14,12),nrow = n, ncol = 1, byrow = T)
B

#Inversa de la diagonal D multiplicado por B
H = I %*% B
H
```

202:1 (Untitled) R Script

Console Terminal Jobs

```
> B = matrix(c(-6,17,-14,12),nrow = n, ncol = 1, byrow = T)
> B
     [,1]
[1,]   -6
[2,]   17
[3,]  -14
[4,]   12
> #Inversa de la diagonal D multiplicado por B
> H = I %*% B
> H
      [,1]
[1,]  -2.0
[2,] -17.0
[3,]   7.0
[4,]   2.4
>
```

Y por último para hallar X, tomaremos una estimación inicial XO.



```
#XO es una estimación inicial
XO = matrix(rep(1,n), nrow = n, ncol = 1, byrow = T)
XO

X = H - (Tj %*% XO)
X
```

202:30 (Untitled) R Script

Console Terminal Jobs

```
> XO = matrix(rep(1,n), nrow = n, ncol = 1, byrow = T)
> XO
     [,1]
[1,]    1
[2,]    1
[3,]    1
[4,]    1
> X = H - (Tj %*% XO)
> X
            [,1]
[1,]  -0.6666667
[2,] -16.0000000
[3,]   8.0000000
[4,]   1.6000000
>
```