Taller en R

Lesly Piñeros, Ernesto Cardona

April 13, 2020

1. Aproximación por suma parcial $s = \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6} \approx 1.64493$

   Solución

   Para implementar la suma en R, se usaran vectores de la siguiente forma:

$$\sum_{i=1}^{k} \frac{1}{i^2} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \cdots + \frac{1}{k^2}$$
$$= sum(1, \frac{1}{2^2}, \frac{1}{3^2}, \cdots, \frac{1}{k^2})$$

```
#aproximaci n por suma parcial

x = 1:10000000
sum(1/x^2)
[1] 1.644934
```

2. Encontrar valor aproximado de la medida utilice una partición regular.

$$\int_0^1 e^{-x^2} dx \approx 0.746821.$$

   Solución.

   Usando la formula de sumas de Rieman.

$$\int_a^b f(x)dx = \lim_{\|\Delta x_i\| \to 0} \sum_{i=1}^{n} f(x_i^*)\Delta x_i$$

   Tomando un n muy grande obtendriamos la siguiente expresión para nuestra función.

$$\int_0^1 e^{-x^2} dx = \sum_{i=1}^{n} e^{-(i/n)^2}(1/n)$$

```
#segundo punto
i= 1:100000
n= length(i) #numero de particiones que se usaran
a= 0 #limite inferior de la integral
b= 1 #limite superior de la integral
delta_x= (b-a)/n
sum(1/n*exp(-(a+(i*(delta_x)))^2)) #integral
[1] 0.746821
```

3. Implementar una función **cribaEratostenes(n)** que devuelve una lista con los primos n.
Solución.
La Criba de Eratóstenes es un procedimiento para determinar todos los números primos hasta cierto número natural dado. Esto se hace recorriendo una tabla de números usando el siguiente procedimiento:

   1) Listar los valores de 2 a n.
   2) Tomar el primer número no tachado o marcado como número primo y se marca como primo.
   3) Tachar los múltiplos del número que se acaban de marcar como primo; para este paso tomaremos el residuo de los numeros del la lista respecto al primo no tachado, si es igual a cero, entonces tenemos un multplico de dicho primo, y debemos sacarlo de la lista de numeros.
   4) 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.

```
primos = function(x){
  vec = 2:x #se crea un vector que inicia desde 2 hasta el numero dado
  d = 1
  while(vec[d]^2 <= x){
    for(n in vec){
      if(vec[d] != n){
        if(n%%vec[d]==0){  #se usa modulo cero para identificar los multiplos del
            numero
          vec = vec[-which(vec==n)] # quita los multiplos al numero que se acaba de
              marcar
        }
      }
    }
    d=d+1
  }
  vec
}
primos(300)
```

4. Correr el programa, realice varias pruebas y modifique la forma para cuando el discriminante es "peligrosamente" igual a cero utilizando una precisión doble.
Solución.
Los únicos números que se pueden representar exactamente en el tipo numérico de R son números enteros y fracciones cuyo denominador es una potencia de 2. Todos los demás números se redondean internamente a 53 dígitos binarios de precisión, pero todavía nos queda la opción de trabajar con más decimales, para esto usaremos el paquete de r **Rmpfr**.

```
# Mi primer RScripts
#Raices de a*x^2+b*x +c
# --- Limpiar el ambiente de trabajo
rm(list=ls())
require(Rmpfr)
cat("Raices de a*x^2+b*x +c.\n")

# Lectura de coeficientes desde el teclado (ENTER)
a = as.numeric(readline("a = "))
b = as.numeric(readline("b = "))
c = as.numeric(readline("c = "))

a = mpfr(a, precBits = 256)
```

```
b = mpfr(b, precBits = 256)
c = mpfr(c, precBits = 256)

# Calculos y criterio de parada
if(a==0) stop("No es cuadratica")
# calculo del Discriminante
d = b^2-4*a*c

# ---
if(d==0){
  x1 = -b/(2*a)
  cat("Una raiz real x1 = ", x1)
}
if(d>0){
  x1 = (-b+sqrt(d))/(2*a)
  x2 = (-b-sqrt(d))/(2*a)
  cat("Dos raices reales")
  print(x1)
  print(x2)
  }
if(d<0) cat("Las raices son complejas")
```

5. Problemas con R.
$A = L + D + U$ Donde L y U son matrices triangular inferior y triangular superior de A, D es la diagonal de A y D no singular:

$$\begin{aligned} AX &= B \\ (L + D + U)X &= B \\ DX &= B - LX - UX \\ X &= D^{-1}(B - LX - UX) \\ X &= D^{-1}B - D^{-1}(L + U)X \\ X^{k+1} &= D^{-1}B - D^{-1}(L + U)X^{k}. \end{aligned}$$

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

- Desagregar la matriz A en la suma de una matriz (D) diagonal (no singular) más una triangular superior U y una triangular inferior L.
- Determine la matriz de transición T_j.
- Encuentre o genere una lista de los valores propios y vectores.
- Resuelva el sistema.

$$T_J = D^{-1} * (L + D)$$

Solución
Usando el metodo de Jacobi, e implementando los pasos dados en el ejercicio, obetenemos el siguiente codigo.

```
options(digits=16);

n=4
#vector b
b=c(6,25,-11,15)
#matriz A
```

```
a=c(10,-1,2,0,-1,11,-1,3,2,-1,10,-1,0,3,-1,8);
A = matrix(a,n, byrow=TRUE);

#Creacion de U, L Y D

U=-A
D= matrix(0,n,n)
I=matrix(0,n,n)
L=-A
for( i in 1:n){
  D[i,i]=A[i,i]
  U[i,1:i]=0
  L[1:i,i]=0
  I[i,i]=1
}

#calcular la inversa de D
if(det(D)){
  Di=(solve(D,I))
  #Hallar Cj y Tj
  Cj=Di%*%b
  Tj=Di%*%(L+U)
  #x inicial
  x=c(0,0,0,0)
  #iteracion
  N=100
  for(i in 1:N){
    x=(Tj%*%x)+Cj
  }
  #solucion x
  print(x)
}else{
  print("no es posible solucionarlo con este metodo")
}
                        [,1]
[1,]   1.0000000000000002
[2,]   2.0000000000000000
[3,]  -1.0000000000000000
[4,]   0.9999999999999998
```

Algunos ejemplos con otras matrices.

```
 matriz A
      [,1] [,2] [,3] [,4]
[1,]     2   -1    3    1
[2,]    -7    9   -1    2
[3,]     1   -2   -5   -1
[4,]     2    3   -1    8
 vector b
[1]  1  2 -1  1
soluci n
                           [,1]
[1,]   2.3316772878679916658
[2,]   2.3316945891934146573
[3,]   0.0001078165721604685
[4,]  -1.3322135238402650348
matriz A
      [,1] [,2] [,3]
```

```
[1,]    1    0    0
[2,]    0   -5    0
[3,]   -1    0    1
vector b
[1]  1 -1  1
soluci n
     [,1]
[1,]  1.0
[2,]  0.2
[3,]  2.0

   matriz A
     [,1] [,2] [,3]
[1,]    0    1   -3
[2,]    1   -5    1
[3,]   -1    1    1
vector b
[1] 3 1 1
Respuesta del programa
[1] "no es posible solucionarlo con este metodo"
```