

R Análisis Numéricos Eddy Herrera Daza

```
##Ejercicios
```

```
library(pracma)
library(Matrix)
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:pracma':
##
##      expm, lu, tril, triu
```

```
library(Rlinsolve)
```

```
## Loading required package: bigmemory
```

```
##
## * Rlinsolve for Solving (Sparse) System of Linear Equations is authored by Kisung You (University of
##
## * Underdetermined system is currently not supported. Solvers with regularization constraints to be
```

```
library(reticulate)
library(BB)
library(matrixcalc)
library(base)
library("rgl")
#require("rgl")
```

```
#setwd("C:/Users/auditorio/Documents/analisis_numerico/parciales/parcial1")
```

1. Para el siguiente ejercicio, instale el paquete “pracma”
 - a. Revise las siguientes funciones con la matriz del ejercicio 2
 - b. Evalúe la matriz de transición para el método **SOR**

```
A = matrix(c(-8.1, -7, 6.123, -2, -1, 4,
-3, -1, 0, -1, -5, 0.6,
-1, 0.33, 6, 1/2), nrow=4, byrow=TRUE)
```

```
D1<-eye(3, m = 3)
D2<-ones(3, m = 3)
D3<-zeros(3, m = 3)
```

```
D1
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

D2

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
## [3,]    1    1    1
```

D3

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

```
D<-diag(diag(A))
L = A
L[lower.tri(L,diag = TRUE)] <- 0
U = A
U[upper.tri(U,diag = TRUE)] <- 0
S = A
S[upper.tri(U,diag = FALSE)] <- 0
w = 0.5
MT = solve((eye(4) + w*solve(D)%*%L))%*%(eye(4)- w*solve(D)%*%S)
MT
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,] 0.2814741 -0.1833525  1.095044 -0.08225679
## [2,] 0.2725000  0.4138250 -0.697500  0.07375000
## [3,] 0.0600000 -0.1198000  0.140000  0.03000000
## [4,] 1.0000000 -0.3300000 -6.000000  0.50000000
```

2. Dada la siguiente matriz, utilice las funciones del paquete para descomponer la matriz $A = L + D + U$ (Jacobi)

```
##      [,1] [,2] [,3] [,4]
## [1,] -8.1 -7.00 6.123 -2.0
## [2,] -1.0 4.00 -3.000 -1.0
## [3,] 0.0 -1.00 -5.000 0.6
## [4,] -1.0 0.33 6.000 0.5
```

```
##      [,1] [,2] [,3] [,4]
## [1,] -8.1    0    0  0.0
## [2,] 0.0     4    0  0.0
## [3,] 0.0     0   -5  0.0
## [4,] 0.0     0    0  0.5
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0   -7  6.123 -2.0
## [2,]    0    0 -3.000 -1.0
## [3,]    0    0 0.000  0.6
## [4,]    0    0 0.000  0.0
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0  0.00    0    0
## [2,]   -1  0.00    0    0
## [3,]    0 -1.00    0    0
## [4,]   -1  0.33    6    0
```

- b. Utilice la función `itersolve(A, b, tol, method = "Gauss-Seidel")` y solucionar el sistema asociado a la matriz A con $b = [1.45, 3, 5.12, -4]^t$ con una tolerancia de $1e^{-9}$

```
MT = solve((eye(4)+solve(D)%*%L))%*%(-solve(D)%*%U)
cat("MATRIZ DE TRANSICION")
```

```
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##      [,1]      [,2]      [,3] [,4]
## [1,] -1.116109  0.275464  5.400356    0
## [2,]  0.930000 -0.374400 -4.080000    0
## [3,]  0.240000 -0.279200 -1.440000    0
## [4,]  2.000000 -0.660000 -12.000000    0
```

```
cat("\nRADIO ESPECTRAL:")
```

```
##
## RADIO ESPECTRAL:
```

```
R=max(abs(eig(MT)))
```

```
print(R)
```

```
## [1] 2.989631
```

```
cat("\n\n")
```

```
b=c(1.45,3,5.12,-4)
res = itersolve(A,b,tol = 1e-9,method = "Gauss-Seidel")
res
```

```
## $x
## [1] -3.696138e+196  2.853479e+196  1.611118e+196 -2.860899e+197
##
## $iter
## [1] 1000
##
## $method
## [1] "Gauss-Seidel"
```

Como el radio espectral es mayor a 1 se tiene que para el método de Gauss-Seidel el método diverge (como se puede ver en los resultados).

- c. Genere 5 iteraciones del método de Jacobi, calcular error relativo para cada iteración

```
MT = -solve(D)%*%(L+U)
cat("MATRIZ DE TRANSICION\n")
```

```
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,] 0.00 -0.8641975  0.7559259 -0.2469136
## [2,] 0.25  0.0000000  0.7500000  0.2500000
## [3,] 0.00 -0.2000000  0.0000000  0.1200000
## [4,] 2.00 -0.6600000 -12.0000000  0.0000000
```

```
cat("\nRADIO ESPECTRAL:")
```

```
##
## RADIO ESPECTRAL:
```

```
R=max(abs(eig(MT)))
```

```
cat(R,"\n")
```

```
## 1.458065
```

```
iteraciones <- c(1:5)
respuestas <- c()
b<-c(1.45,3,5.12,-4)

for(i in iteraciones)
{
  res = itersolve(A,b,tol = 1e-9,nmax=i,method = "Jacobi")
  respuestas <- c(respuestas, Norm(res$x))
}
tabla <- data.frame(
  "Iteraciones" = 1:length(respuestas),
  "Norma" = respuestas
)

print(tabla)
```

```
##   Iteraciones   Norma
## 1           1  8.102044
## 2           2  4.554986
## 3           3 19.737490
## 4           4 10.610326
## 5           5 38.134540
```

```
erroresR = c()
for(n in iteraciones-1){
  erroresR = c(erroresR,abs(respuestas[n]-respuestas[n+1])/abs(respuestas[n+1]))
}

tabla <- data.frame(
  "Iteraciones" = 1:length(erroresR),
  "Error R." = erroresR
)

print(tabla)
```

```
##   Iteraciones  Error.R.
## 1           1 0.7787199
## 2           2 0.7692216
## 3           3 0.8602153
## 4           4 0.7217660
```

Como se puede ver en las primeras 5 iteraciones el m?todo se comporta de manera inestable, y por el rango espectral, se sabe que el m?todo no va a converger(no se estabiliza).

3. Sea el sistema $AX = b$

- Implemente una función en R para que evalúe las raíces del polinomio característico asociado a la matriz A
- Use el teorema de convergencia para determinar cuál método iterativo es más favorable.
- Evalúe la matriz de transición para cada caso y en el caso del método de relajación determine el valor óptimo de ω
- Teniendo en cuenta lo anterior resolver el sistema

```
A = matrix(c(4, -1, -1, -1, -1, 4,
-1, -1, -1, -1, 4, -1,
-1, -1, -1, 4), nrow=4, byrow=TRUE)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4  -1  -1  -1
## [2,]  -1    4  -1  -1
## [3,]  -1  -1    4  -1
## [4,]  -1  -1  -1    4
```

```
b = c(1, 5, 1.5, -2.33)
```

```
##A
R=eig(A)

cat("Valores propios del polinomio caracteristica:",R,"\n")
```

```
## Valores propios del polinomio caracteristica: 5 5 5 1
```

```
##metodo de Gauss Seidel
MT = solve((eye(4)+solve(D)%*%L))%*%(-solve(D)%*%U)
cat("\nMATRIZ DE TRANSICION\n")
```

```
##
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##          [,1]      [,2]      [,3] [,4]
## [1,] -1.116109  0.275464  5.400356  0
## [2,]  0.930000 -0.374400 -4.080000  0
## [3,]  0.240000 -0.279200 -1.440000  0
## [4,]  2.000000 -0.660000 -12.000000  0
```

```
cat("\nRADIO ESPECTRAL:")
```

```
##
## RADIO ESPECTRAL:
```

```
R=max(abs(eig(MT)))
print(R)
```

```
## [1] 2.989631
```

```
res = itersolve(A,b,tol = 1e-9,method = "Gauss-Seidel")
res
```

```
## $x
## [1] 1.234 2.034 1.334 0.568
##
## $iter
## [1] 36
##
## $method
## [1] "Gauss-Seidel"
```

```
##metodo de Jacobi
MT = -solve(D)%*%(L+U)
cat("\nMATRIZ DE TRANSICION\n")
```

```
##
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##          [,1]      [,2]      [,3]      [,4]
## [1,] 0.00 -0.8641975  0.7559259 -0.2469136
## [2,] 0.25  0.0000000  0.7500000  0.2500000
## [3,] 0.00 -0.2000000  0.0000000  0.1200000
## [4,] 2.00 -0.6600000 -12.0000000  0.0000000
```

```
cat("\nRADIO ESPECTRAL:")
```

```
##  
## RADIO ESPECTRAL:
```

```
R=max(abs(eig(MT)))  
print(R)
```

```
## [1] 1.458065
```

```
res = itersolve(A,b,tol = 1e-9,method = "Jacobi")  
res
```

```
## $x  
## [1] 1.234 2.034 1.334 0.568  
##  
## $iter  
## [1] 70  
##  
## $method  
## [1] "Jacobi"
```

```
##metodo de relajacion  
w = 0.5  
MT = solve((eye(4) + w*solve(D)%*%L))%*%(eye(4)- w*solve(D)%*%S)  
cat("\nMATRIZ DE TRANSICION\n")
```

```
##  
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##           [,1]      [,2]      [,3]      [,4]  
## [1,] 0.2814741 -0.1833525  1.095044 -0.08225679  
## [2,] 0.2725000  0.4138250 -0.697500  0.07375000  
## [3,] 0.0600000 -0.1198000  0.140000  0.03000000  
## [4,] 1.0000000 -0.3300000 -6.000000  0.50000000
```

```
cat("\nRADIO ESPECTRAL:")
```

```
##  
## RADIO ESPECTRAL:
```

```
R=max(abs(eig(MT)))  
print(R)
```

```
## [1] 0.561444
```

d Comparar con la solución por defecto

```
solucion<- solve(A,b)
solucion
```

```
## [1] 1.234 2.034 1.334 0.568
```

```
lsolve.sor(A,b,w=1.2)
```

```
## * lsolve.sor : Initialiszed.
```

```
## * lsolve.sor : computations finished.
```

```
## $x
```

```
##          [,1]
```

```
## [1,] 1.2339787
```

```
## [2,] 2.0339833
```

```
## [3,] 1.3339866
```

```
## [4,] 0.5679908
```

```
##
```

```
## $iter
```

```
## [1] 10
```

```
##
```

```
## $errors
```

```
##          [,1]
```

```
## [1,] 3.778031e-01
```

```
## [2,] 1.271753e-01
```

```
## [3,] 4.264339e-02
```

```
## [4,] 1.533817e-02
```

```
## [5,] 4.304342e-03
```

```
## [6,] 1.683052e-03
```

```
## [7,] 4.533538e-04
```

```
## [8,] 1.787649e-04
```

```
## [9,] 4.990544e-05
```

```
## [10,] 1.885037e-05
```

```
## [11,] 5.620038e-06
```

El valor optimo para lograr el menor numero de itaraciones es $w=1.2$.

El metodo mas optimo es el de relajación con $w = 1.2$ (10 iteraciones).

3.

a. Pruebe el siguiente algoritmo con una matriz A_3 , modifiquelo para que $a_{ii} = 0$ para todo i

```
tril1 <- function(M, k = 0) {
  if (k == 0) {
    M[upper.tri(M, diag = TRUE)] <- 0
  } else {
    M[col(M) >= row(M) + k + 1] <- 0
  }
  return(M)
}
```



```
matriz <- matrix(c(1:9),nrow = 3)

trill(matriz)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    2    0    0
## [3,]    3    6    0
```

- b. Implemente una función en R para que dada una matriz A se obtenga una matriz diagonal D donde en la diagonal estan los mismo elementos de A

```
diagonal = function(A){
  diag<-eye(nrow(A))*diag(A)
  return(diag)
}

dia = diagonal(A)
dia
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    4    0    0    0
## [2,]    0    4    0    0
## [3,]    0    0    4    0
## [4,]    0    0    0    4
```

4. Cree una función que cuente el número de multiplicaciones en el método directo de Gauss Jordan, para resolver un sistema de n ecuaciones y pruebelo para $n = 5$

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

A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2   -1    4    1   -1
## [2,]   -1    3   -2   -1    2
## [3,]    5    1    3   -4    1
## [4,]    3   -2   -2   -2    3
## [5,]   -4   -1   -5    3   -4
```

```
b = c(7,1,33,24,-49)
```

```
gaussJordan= function(a,b){
  p <- nrow(A)
  (U.pls <- cbind(A,b))

  U.pls[1,] <- U.pls[1,]/U.pls[1,1]
```

```

can_de_mult = 0;
i <- 2
while (i < p+1) {
  j <- i
  while (j < p+1) {
    can_de_mult = can_de_mult +1;
    U.pls[j, ] <- U.pls[j, ] - U.pls[i-1, ] * U.pls[j, i-1]
    j <- j+1
  }
  while (U.pls[i,i] == 0) {
    U.pls <- rbind(U.pls[-i,],U.pls[i,])
  }
  U.pls[i,] <- U.pls[i,]/U.pls[i,i]
  i <- i+1
}
for (i in p:2){
  for (j in i:2-1) {
    can_de_mult = can_de_mult +1;
    U.pls[j, ] <- U.pls[j, ] - U.pls[i, ] * U.pls[j, i]
  }
}
lista <- list("resultado" = U.pls,"multiplicaciones" = can_de_mult)
return (lista)

}

res = gaussJordan(A,b)

print(res$resultado)

```

```

##          b
## [1,] 1 0 0 0 0 1
## [2,] 0 1 0 0 0 -2
## [3,] 0 0 1 0 0 3
## [4,] 0 0 0 1 0 -4
## [5,] 0 0 0 0 1 5

```

```

cat("Numero total de multiplicaciones: ", res$multiplicaciones);

```

```

## Numero total de multiplicaciones: 20

```

5. Dado el siguiente sistema:

$$\begin{aligned}
 2x - z &= 1 \\
 \beta x + 2y - z &= 2 \\
 -x + y + \alpha z &= 1
 \end{aligned}$$

a. Encuentre el valor de α y β para asegura la convergencia por el método de Jacobi.

```

A = matrix(c(2, 0, 1,
             2, 2, -1,
             -1, 1, 2), nrow=3, byrow=TRUE)

```

```
B = matrix (c(1,2,1),nrow=3, byrow=TRUE)
```

```
Ab = cbind(A,B)
print(Ab)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    0    1    1
## [2,]    2    2   -1    2
## [3,]   -1    1    2    1
```

```
D <- diag(diag(A))
L = A
L[lower.tri(L,diag = TRUE)] <- 0
U = A
U[upper.tri(U,diag = TRUE)] <- 0

MT = -solve(D)%*(L+U)

R=max(abs(eig(MT)))

cat("Radio Espectral",R,"\n")
```

```
## Radio Espectral 0.8053436
```

```
cat("alpha = 2 \n")
```

```
## alpha = 2
```

```
cat("beta = 2 \n")
```

```
## beta = 2
```

Alpha y beta con valor igual a 2 aseguran la convergencia por el metodo de Jacobi, debido a que su radio espectral es menor a 1.

- b. Genere una tabla que tenga 10 iteraciones del método de Jacobi con vector inicial $x_0 = [1, 2, 3]^t$
- c. Grafique cada ecuación y la solución La grafica se realiza plot3d por tal razón solo se puede apreciar el resultado cuando se realiza la ejecución del codigo en R. $2x - z = 1$ Se encuentra representado por el plano de color Azul
 $2x + 2y - z = 2$ Se encuentra representado por el plano de color Rojo
 $-x + y + 2z = 1$ Se encuentra representado por el plano de color Azul Las soluciones se encuentran representadas por los planos de color Negros

```
x0 = c(1,2,3)
nor = c()
m = c()
```

```
for(i in 1:10){
```

```

sol <- itersolve(A,B,x0,nmax=i,method="Jacobi")
cat("Iteración ", i ,"\n")

P <- matrix (c (sol$x), nrow=3,byrow=TRUE)

print(P)

b <- norm(P)

nor <- c(nor,b)
m <- c(m,i)

# p1<-planes3d(2, 0, 1, 1, alpha = 10,col="blue")
# p3<-planes3d(2, 2, -1, 2, alpha = 10,col="red")
# p2<-planes3d(-1, 1, 2, 1, alpha = 10, col="yellow")
# p4<-planes3d(sol$x[1],sol$x[2],sol$x[3], alpha = 1 , col="black")
# plot3d(x, y, z, type = "s", col = "white", size = 0, add=p2+p1+p3+p4)
# decorate3d(axes=TRUE,box=TRUE)
# aspect3d(1, 1, 1)
}

```

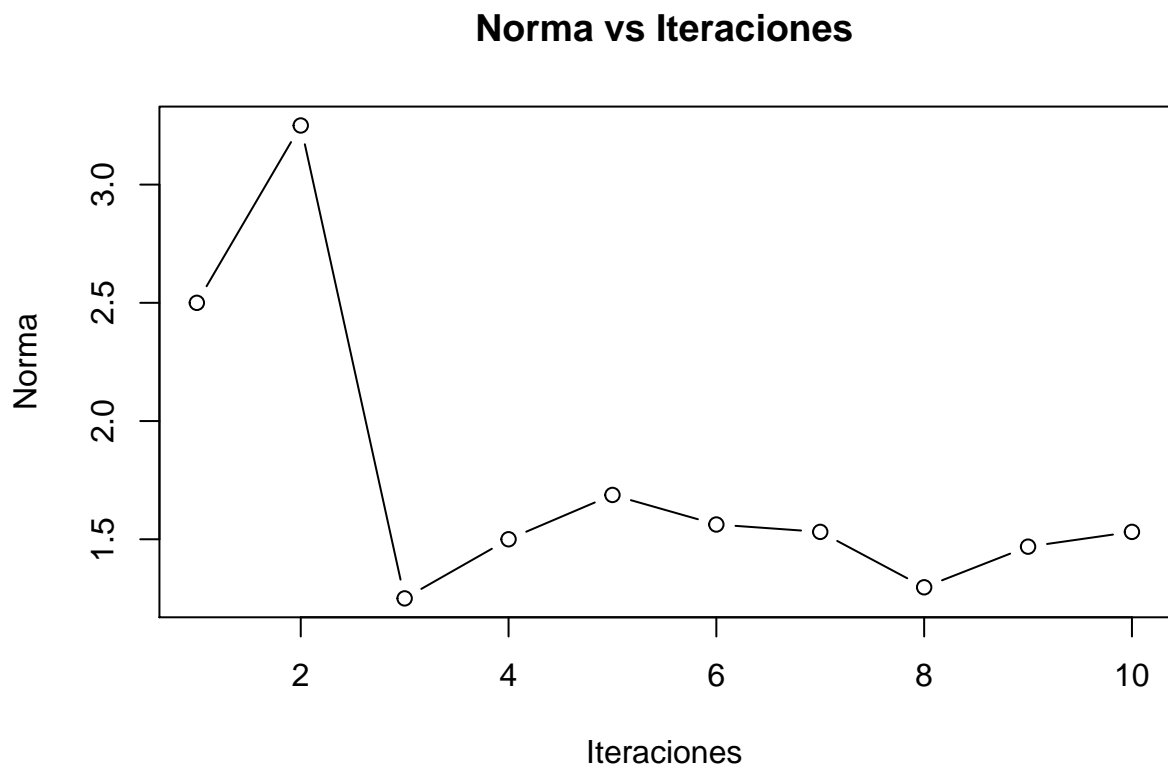
```

## Iteración 1
##      [,1]
## [1,] -1.0
## [2,]  1.5
## [3,]  0.0
## Iteración 2
##      [,1]
## [1,]  0.50
## [2,]  2.00
## [3,] -0.75
## Iteración 3
##      [,1]
## [1,]  0.875
## [2,]  0.125
## [3,] -0.250
## Iteración 4
##      [,1]
## [1,] 0.625
## [2,] 0.000
## [3,] 0.875
## Iteración 5
##      [,1]
## [1,] 0.0625
## [2,] 0.8125
## [3,] 0.8125
## Iteración 6
##      [,1]
## [1,] 0.09375
## [2,] 1.34375
## [3,] 0.12500

```

```
## Iteración 7
##      [,1]
## [1,] 0.43750
## [2,] 0.96875
## [3,] -0.12500
## Iteración 8
##      [,1]
## [1,] 0.562500
## [2,] 0.500000
## [3,] 0.234375
## Iteración 9
##      [,1]
## [1,] 0.3828125
## [2,] 0.5546875
## [3,] 0.5312500
## Iteración 10
##      [,1]
## [1,] 0.2343750
## [2,] 0.8828125
## [3,] 0.4140625
```

```
plot(x = m, y = nor,
     xlab = "Iteraciones", ylab = "Norma", type="b",
     main = "Norma vs Iteraciones")
```



6. Instalar el paquete Matrix y descomponga la matriz A (del punto dos) de la forma LU y la factorizarla

como $A = QR$

```
A = matrix(c(-8.1, -7, 6.123, -2, -1, 4,  
            -3, -1, 0, -1, -5, 0.6,  
            -1, 0.33, 6, 1/2), nrow=4, byrow=TRUE)  
cat("Matriz A Original: ", "\n")
```

```
## Matriz A Original:
```

```
print(A)
```

```
##      [,1] [,2] [,3] [,4]  
## [1,] -8.1 -7.00 6.123 -2.0  
## [2,] -1.0 4.00 -3.000 -1.0  
## [3,] 0.0 -1.00 -5.000 0.6  
## [4,] -1.0 0.33 6.000 0.5
```

```
cat("Matriz L y U : ", "\n")
```

```
## Matriz L y U :
```

```
B=lu.decomposition(A)  
print(B)
```

```
## $L  
##      [,1] [,2] [,3] [,4]  
## [1,] 1.0000000 0.0000000 0.0000000 0  
## [2,] 0.1234568 1.0000000 0.0000000 0  
## [3,] 0.0000000 -0.2055838 1.0000000 0  
## [4,] 0.1234568 0.2455076 -1.068263 1  
##  
## $U  
##      [,1] [,2] [,3] [,4]  
## [1,] -8.1 -7.000000 6.123000 -2.0000000  
## [2,] 0.0 4.864198 -3.755926 -0.7530864  
## [3,] 0.0 0.000000 -5.772157 0.4451777  
## [4,] 0.0 0.000000 0.000000 1.4073689
```

```
cat("Multiplicación de matriz L y U: ", "\n")
```

```
## Multiplicación de matriz L y U:
```

```
print(B$L*%*%B$U)
```

```
##      [,1] [,2] [,3] [,4]  
## [1,] -8.1 -7.00 6.123 -2.0  
## [2,] -1.0 4.00 -3.000 -1.0  
## [3,] 0.0 -1.00 -5.000 0.6  
## [4,] -1.0 0.33 6.000 0.5
```

```
#-----
#Factorización de matriz A como QR.
cat("\n")
```

```
C=qr(A)
print(C)
```

```
## $qr
##           [,1]      [,2]      [,3]      [,4]
## [1,] 8.2225300  6.3690859 -6.3966078  2.0310050
## [2,] 0.1216171 -5.0540721  1.4128123  0.6669168
## [3,] 0.0000000 -0.1978603  8.0360748  0.3488413
## [4,] 0.1216171  0.2273528 -0.7262035  0.9584103
##
## $rank
## [1] 4
##
## $qraux
## [1] 1.9850983 1.9534999 1.6874798 0.9584103
##
## $pivot
## [1] 1 2 3 4
##
## attr("class")
## [1] "qr"
```

```
C1=qr.Q(C)
C2=qr.R(C)
cat("Mutiplicación de matriz C1 y C2","\n")
```

```
## Mutiplicación de matriz C1 y C2
```

```
C1%*%C2
```

```
##           [,1] [,2] [,3] [,4]
## [1,] -8.1 -7.00  6.123 -2.0
## [2,] -1.0  4.00 -3.000 -1.0
## [3,]  0.0 -1.00 -5.000  0.6
## [4,] -1.0  0.33  6.000  0.5
```

7.

- a. Determinar numéricamente la intersección entre la circunferencia $x^2 + y^2 = 1$ y la recta $y = x$. Usamos una aproximación inicial (1,1). Utilice el paquete BB y la función BBSolve() del paquete, grafique la solución

```
trigexp = function(x) {
n = length(x)
##Declara F como una secuencia de n valores 'vacios'
F = rep(NA, n)
```

```

##primera ecuacion no lineal con  $x[1] = x$  y  $x[2] = y$ 
F[1] = x[1]^2 + x[2]^2 - 1
##ultima ecuacion no lineal con  $x[1] = x$  y  $x[2] = y$ 
F[2] = x[1] - x[2]
##retorna F con el sistema de ecuaciones no lineales
F
}
n = 2
##n = numero de ecuaciones
p0 = c(1,1)
sol = BBSolve(par=p0, fn=trigexp)

```

```
## Successful convergence.
```

```

##BBSolve resuelve el sistema de ecuaciones no lineales determinado en trigexp
sol$par

```

```
## [1] 0.7071068 0.7071068
```

```
##solucion al sistema de ecuaciones no lineales
```

b Analizar y comentar el siguiente código

```

trigexp = function(x) {
  ##calcula el tamaño de x
  n = length(x)
  ##Declara F como una secuencia de n valores 'vacios'
  F = rep(NA, n)
  ##primera ecuacion no lineal con  $x[1] = x$  y  $x[2] = y$ 
  F[1] = 3*x[1]^2 + 2*x[2] - 5 + sin(x[1] - x[2]) * sin(x[1] + x[2])
  ##tn1 va a ser el índice de para las ecuaciones 2 hasta n-1
  tn1 = 2:(n-1)
  #declaracion de n-2 ecuaciones lineales con sus parametros variando segun el numero de la ecuacion
  F[tn1] = -x[tn1-1] * exp(x[tn1-1] - x[tn1]) + x[tn1] *
    ( 4 + 3*x[tn1]^2) + 2 * x[tn1 + 1] + sin(x[tn1] -
    x[tn1 + 1]) * sin(x[tn1] + x[tn1 + 1]) - 8
  ##ultima ecuacion no lineal con  $x[1] = x$  y  $x[2] = y$ 
  F[n] = -x[n-1] * exp(x[n-1] - x[n]) + 4*x[n] - 3
  ##retorna F con el sistema de ecuaciones no lineales
  F
}
n = 10000
##n = numero de ecuaciones
p0 = runif(n) # n initial random starting guesses
sol = BBSolve(par=p0, fn=trigexp)

```

```
## Successful convergence.
```

```

##BBSolve resuelve el sistema de ecuaciones no lineales determinado en trigexp
sol$par

```


[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

## [9721] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9727] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9733] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9739] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9745] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9751] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9757] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9763] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9769] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9775] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9781] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9787] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9793] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9799] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9805] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9811] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9817] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9823] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9829] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9835] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9841] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9847] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9853] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9859] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9865] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9871] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9877] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9883] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9889] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9895] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9901] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9907] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9913] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9919] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9925] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9931] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9937] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9943] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9949] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9955] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9961] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9967] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9973] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9979] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9985] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9991] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9997] 1.0000000 1.0000000 1.0000001 0.9999995

```

##solucion al sistema de ecuaciones no lineales

8. Demuestre y realice varias pruebas que la matriz de transición por el método de Gauss-Seidel esta dada por $T = (-D^{-1}U)(I + LD^{-1})^{-1}$

$$Ax = B$$

se puede descomponer A en su su la diagonal, la matriz triangular superior y la matriz triangular inferior

$$A = L + D + U$$

Remplazando en la primera ecuación queda:

$$Lx + Dx + Ux = B$$

Pasando a restar Ux y Lx queda:

$$Dx = B - Lx - Ux$$

multiplicando por la inversa de D(a la izquierda) en toda la ecuación:

$$x = D^{-1}B - D^{-1}Lx - D^{-1}Ux$$

Y la forma iterativa para el metodo de Gauss queda de la siguiente manera

$$x^{k+1} = D^{-1}B - D^{-1}Lx^{k+1} - D^{-1}Ux^k$$

Teniendo en cuenta la definicion de convergencia:

$$E^{k+1} = T * E^k$$

Ahora se puede restar la solucion x la solucion en la interacion k+1

$$x - x^{k+1} = -D^{-1}L(x - x^{k+1}) - D^{-1}U(x - x^k)$$

Teniendo en cuenta la definicion de error para metodos iterativos se puede simplificar la formula:

$$E^{k+1} = -D^{-1}LE^{k+1} - D^{-1}UE^k$$

Pasando a sumar y factorizando el error en la iteracion k+1 la ecuacion queda:

$$E^{k+1}(I + D^{-1}L) = -D^{-1}UE^k$$

Sacando la inversa de $I + D^{-1}L$ a ambos lados de la ecuacion:

$$E^{k+1} = (I + D^{-1}L)^{-1}(-D^{-1}U)E^k$$

Finalmente teniendo en cuenta la ecuacion que relaciona el error y la matriz de transicion se puede deduir que el valor de la matriz de transicion es:

$$T = (I + D^{-1}L)^{-1}(-D^{-1}U)$$

Pruebas de la matriz de transicion para Gauss-Seidel


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

A

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2  -1    4    1  -1
## [2,]   -1    3   -2   -1    2
## [3,]    5    1    3   -4    1
## [4,]    3   -2   -2   -2    3
## [5,]   -4   -1   -5    3   -4
```

```
b = c(7,1,33,24,-49)
```

```
D<-diag(diag(A))
```

```
L = A
```

```
L[lower.tri(L,diag = TRUE)] <- 0
```

```
U = A
```

```
U[upper.tri(U,diag = TRUE)] <- 0
```

```
MT = solve((eye(5)+solve(D)%*%L))%*%(-solve(D)%*%U)
cat("\nMATRIZ DE TRANSICION\n")
```

```
##
```

```
## MATRIZ DE TRANSICION
```

```
print(MT)
```

```
##      [,1]      [,2]      [,3]      [,4] [,5]
## [1,] 2.2222222 3.888889 6.444444 -2.333333 0
## [2,] 0.1111111 -1.680556 -2.402778 0.7083333 0
## [3,] -1.3333333 -2.083333 -3.416667 1.2500000 0
## [4,] 0.0000000 -1.375000 -2.875000 1.1250000 0
## [5,] -1.0000000 -0.250000 -1.250000 0.7500000 0
```

```
A = matrix(c(2,3,
             5,7), nrow=2, byrow=TRUE)
```

A

```
##      [,1] [,2]
## [1,]    2    3
## [2,]    5    7
```

```
D<-diag(diag(A))
```

```
L = A
```

```
L[lower.tri(L,diag = TRUE)] <- 0
```

```
U = A
```

```
U[upper.tri(U,diag = TRUE)] <- 0
```

```
MT = solve((eye(2)+solve(D)%*%L))%*%(-solve(D)%*%U)

cat("\nMATRIZ DE TRANSICIÓN\n")
```

```
##
## MATRIZ DE TRANSICIÓN
```

```
print(MT)
```

```
##           [,1] [,2]
## [1,]  1.0714286    0
## [2,] -0.7142857    0
```