

# Tarea4

## Ejercicios

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

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

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

```
library(psych)
```

```
##  
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:pracma':  
##  
##      logit, polar
```

```
library(rootSolve)
```

```
##  
## Attaching package: 'rootSolve'
```

```
## The following objects are masked from 'package:pracma':  
##  
##    gradient, hessian
```

```
library(matlib)
```

```
##  
## Attaching package: 'matlib'
```

```
## The following object is masked from 'package:psych':  
##  
##    tr
```

```
## The following objects are masked from 'package:pracma':  
##  
##    angle, inv
```

```
library(xlsx)  
library(BB)
```

```
setwd("E:/Documentos/Uni/Analisis Numerico/Tarea 4/Solucion")
```

1. Para el siguiente ejercicio, instale el paquete “pracma”
- a. Revise las siguientes funciones con la matriz del ejercicio 2

Sol. Las funciones realizan lo siguiente: `eye(n, m = n)` Retorna una matriz diagonal de tamaño  $n \times n$  `ones(n, m = n)` Retorna una matriz llena de 1 de tamaño  $n \times n$  `zeros(n, m = n)` Retorna una matriz llena de 0 de tamaño  $n \times n$

- b. Evalúe la matriz de transición para el método **SOR** Sol, se añade la función realizada para la matriz de transición, basado en la fórmula para  $T$  en el método **SOR** tomada de la bibliografía de este documento.
2. Dada la siguiente matriz, utilice las funciones del paquete para descomponer la matriz  $A = L + D + U$  (Jacobi)

```
A2 = 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)

A2
```

```
##      [,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
```

```
descomponer<-function(A,n){
  D<- (A*eye(n, m = n)) #Diagonal de A
  L<-A
  L[lower.tri(L, diag = TRUE) ]<-0 #triangular inferior
  U<-A
  U[upper.tri(U, diag = TRUE) ]<-0#triangular superior
  des<-D+L+U
  des
}
descomponer(A2,4)
```

```
##      [,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
```

- 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  $1e^{-9}$

```
##Segundo B
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)
b <- matrix(c(1.45,3,5.12,-4),nrow=4,ncol=1)

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

```

```

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

```

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

```

##Segundo C
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)
b <- matrix(c(1.45,3,5.12,-4),nrow=4,ncol=1)

itersolve(A, b, x0 = 1:4, nmax = 5, tol = 1e-9, method = "Jacobi")

```

```

## $x
## [1] -12.167413  1.392056  -7.628979 -200.531199
##
## $iter
## [1] 5
##
## $method
## [1] "Jacobi"

```

3. Sea el sistema  $AX = b$

a. Implemente una función en R para que evalúe las raíces del polinomio característico asociado a la matriz  $A$

```
## [1] -8.864279 -5.833187 1.286718 4.810733
```

b. Use el teorema de convergencia para determinar cuál método iterativo es más favorable.

Sol La eficiencia para estos métodos iterativos está dada por  $T(n) = kO(n^2)$ , con  $k$  como el número de iteraciones, para eso se calcula la cantidad de iteraciones que toma cada método para resolver la matriz  $A$  y se desbloquean las iteraciones máximas y se obtiene lo siguiente:

```
A = matrix(c(4, -1, -1, -1, -1, 4,
            -1, -1, -1, -1, 4, -1,
            -1, -1, -1, 4), nrow=4, byrow=TRUE)
b = c(1, 5, 1.5, -2.33)

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

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

```
itersolve(A, b, x0 = 1:4, tol = 1e-9, method = "Jacobi")
```

```
## $x
## [1] 1.234 2.034 1.334 0.568
##
## $iter
## [1] 66
##
```

```
## $method
## [1] "Jacobi"
```

Con esto se puede ver que se obtiene un menor numero de iteraciones en la Gauss-Seidel, siendo de 36 vs 66 de JAcobi por lo que el metodo mas favorable seria el Gauss-Seidel

c. Evalúe la matriz de transición para cada caso y en el caso del método de relajación determine el valor óptimo de  $\omega$

```
A = matrix(c(4, -1, -1, -1, -1, 4,
            -1, -1, -1, -1, 4, -1,
            -1, -1, -1, 4), nrow=4, byrow=TRUE)
TransicionJac<-function(A,n){
  D<- (A*eye(n, m = n)) #Diagonal de A
  L<-A
  L[lower.tri(L, diag = FALSE) ]<-0 #triangular inferior

  auxI<-inv(D)
  U<-A
  U[upper.tri(U, diag = FALSE) ]<-0#triangular superior

  Tra<-auxI*(L+U)
  Tra
}
funcTransicionSOR<-function(A,n,w){
  D<- (A*eye(n, m = n)) #Diagonal de A
  L<-A
  L[lower.tri(L, diag = FALSE) ]<-0 #triangular inferior
  aux<-(D-(w*L))
  auxI<-inv(aux)
  U<-A
  U[upper.tri(U, diag = FALSE) ]<-0#triangular superior
  aux2<-((1-w)*D)+(U*w)
  Tra<-auxI*aux2
  Tra
}
TransicionJac(A,4)
```

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

```
funcTransicionSOR(A,4,1.9)
```

```
##      [,1]      [,2]      [,3]      [,4]
## [1,] -1.111111  0.000000  0.000000  0.000000
## [2,]  0.000000 -1.111111  0.000000  0.000000
## [3,]  0.000000  0.000000 -1.111111  0.000000
## [4,]  0.000000  0.000000  0.000000 -1.111111
```

d. 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)
```

```
itersolve(A, b, tol = 1e-5, method = "Gauss-Seidel")
```

```
## $x
## [1] 1.2339811 2.0339836 1.3339858 0.5679876
##
## $iter
## [1] 20
##
## $method
## [1] "Gauss-Seidel"
```

```
itersolve(A, b, x0 = 1:4, tol = 1e-5, method = "Jacobi")
```

```
## $x
## [1] 1.2340682 2.0340682 1.3340682 0.5680682
##
## $iter
## [1] 34
##
## $method
## [1] "Jacobi"
```

d Comparar con la solución por defecto La diferencia con la solución por defecto es que la solución por los métodos iterativos tiene más decimales, por lo que se acerca más a la solución  $x$  teniendo un error menor

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

```
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)
solve(A,b)
```

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

3.

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

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

##Modificado:

trill <- function(M, k = 0) {
  if (k == 0) {
    M[upper.tri(M, diag = FALSE)] <- 0
  } else {
    M[col(M) >= row(M) + k + 1] <- 0
  }
  D1<-eye(row(M), col(M))
  D2<-ones(row(M), col(M))
  D4<-(-1*(D1-D2))
  M<-M*D4
  return(M)
}
```

- b. Implemente una función en R para que dada una matriz  $A$  se obtenga una matriz diagonal  $D$  donde en la diagonal están los mismos elementos de  $A$
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 pruébelo para  $n = 5$

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  20   16  -18  -28  -2  -15
## [2,]  14   16  -28  -25  17   26
## [3,] -21    3  -13   12  -1  -24
## [4,]  18  -10   13   -6   6   18
## [5,] -29    2  -26   -6  12   23
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    0    0    0    0 -1.908872
## [2,]    0    1    0    0    0  2.786432
## [3,]    0    0    1    0    0  3.604952
## [4,]    0    0    0    1    0 -1.820118
## [5,]    0    0    0    0    1  3.739823
```

```
## Multiplicaciones 20
```

5. Dado el siguiente sistema:

$$2x - z = 1$$

$$\beta x + 2y - z = 2$$

$$-x + y + z = 1$$

- a. Encuentre el valor de  $\alpha$  y  $\beta$  para asegurar la convergencia por el método de Jacobi
- 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

```
jacobi <- function(a, ciclos) {
  n <- nrow(a)
  id <- diag(x = 1, nrow=n, ncol=n)
```

```

Q<-id
for (k in 1:ciclos) {
  for ( i in 1:(n-1)) {
    for ( j in (i+1):n) {
      control <- 10^(-k)
      if( abs(a[i,j]) > control) {
        print(c(a[i,j],control))
        angulo <- 0.5*atan(2*a[i,j]/(a[i,i] - a[j,j]))
        c<-cos(angulo)
        s<-sin(angulo)
        p<-id
        p[i,i]<-c
        p[j,j]<-c
        p[i,j]<-s
        p[j,i]<- s
        Q <- Q%*%p
        a<-t(p)%*%a%*%p
        a[i,j]<-0
        a[j,i]<-0
      }
    }
  }
  cat("estado: ", a, "\n")
}
return(list(raices=diag(a),vectores=Q,estado=a))
}

```

*# Aplicacion*

```

A<-rbind(c(2, 0, -1),c(1, 2,-1),c(-1,1,2))
jacobi(A,10)

```

```

## [1] -1.0  0.1
## estado:  3 1.414214 0 -0.7071068 2 0.7071068 0 -1.110223e-16 1
## [1] -0.7071068  0.0100000
## estado:  2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado:  2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1

```

```
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
## estado: 2.5 0 -0.3250576 0 2.5 0.627963 5.103705e-17 -9.8596e-17 1
```

```
## $raices
## [1] 2.5 2.5 1.0
##
## $vectores
##           [,1]      [,2]      [,3]
## [1,] 0.6279630 0.3250576 0.7071068
## [2,] -0.4597008 0.8880738 0.0000000
## [3,] -0.6279630 -0.3250576 0.7071068
##
## $estado
##           [,1]      [,2]      [,3]
## [1,] 2.5000000 0.0000000 5.103705e-17
## [2,] 0.0000000 2.5000000 -9.859600e-17
## [3,] -0.3250576 0.627963 1.000000e+00
```

```
Jacobi<- function(ciclos){
  A <- matrix(c(2, 0, -1,1, 2,-1,-1,1,3),nrow=3,ncol=3)
  B<- matrix(c(1,2,1), nrow =3, ncol=1)
  Xk<-matrix(c(1,2,3),nrow =3, ncol =1)
  L<- lower.tri(A, diag = FALSE)
  U<- upper.tri(A, diag = FALSE)
  D<- diag(x=1,nrow =3 ,ncol=3, names = TRUE)
  R<-L+U

  cat("X0: ", Xk, "\n")
  for (k in 1:ciclos)
  {
```

```

P<- (R%*%Xk)
Xk<- (1/det(A)*(D))%*%(B-P)
cat("X",k," : ", Xk, "\n")
k<-k+1
}
}
Jacobi(10)

```

```

## X0:  1 2 3
## X 1 :  -0.3636364 -0.1818182 -0.1818182
## X 2 :  0.1239669 0.231405 0.1404959
## X 3 :  0.05709992 0.1577761 0.05860255
## X 4 :  0.0712383 0.1712998 0.07137491
## X 5 :  0.06884776 0.1688533 0.06886017
## X 6 :  0.06929877 0.1692993 0.0692999
## X 7 :  0.06921826 0.1692183 0.06921836
## X 8 :  0.06923303 0.169233 0.06923304
## X 9 :  0.06923036 0.1692304 0.06923036
## X 10 :  0.06923084 0.1692308 0.06923084

```

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

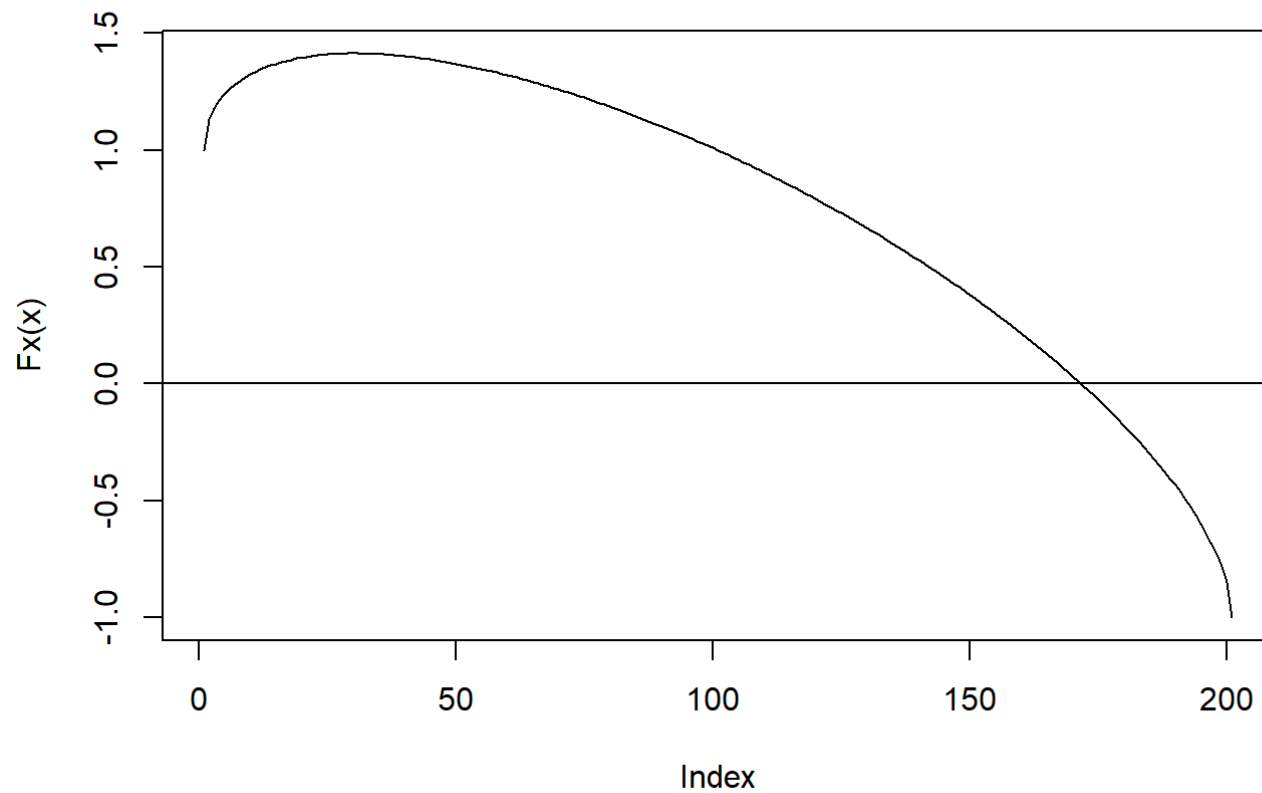
```

## A = QR:
## 9.433333 4.78 -4.789667 1.933333 7.066667 1.113333 -1.748667 1.6 6.733333 2.78 -1.082 1.066667 7.066667 2.3366
67 -4.748667 1.1

```

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 b Analizar y comentar el siguiente código



```
## Solucion= 0.5      Error= 0.2320508      Iteracion= 1
## Solucion= 0.7886751  Error= 0.07616322      Iteracion= 2
## Solucion= 0.6957109  Error= 0.01148628      Iteracion= 3
## Solucion= 0.7064095  Error= 0.0006976451      Iteracion= 4
## Solucion= 0.7071124  Error= 5.571976e-06      Iteracion= 5
## Solucion= 0.7071068  Error= 2.746025e-09      Iteracion= 6
```

```
## Successful convergence.
```

[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]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

```
## [9991] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [9997] 1.0000000 1.0000000 1.0000001 0.9999989
```

8. Demuestre y realice varias pruebas que la matriz de transición por el método de Gauss-Seidel esta dada por

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

Referencias:

- <https://numericalmethods2014.wordpress.com/s-o-r/>
- [https://profs.info.uaic.ro/~fliacob/An2/2012-2013/Resurse/Relative%20la%20Matlab/Faddeev\\_Leverrier%20Method%20for%20matrices.pdf](https://profs.info.uaic.ro/~fliacob/An2/2012-2013/Resurse/Relative%20la%20Matlab/Faddeev_Leverrier%20Method%20for%20matrices.pdf)