AnÃ¡lisis Numericos Eddy Herrera Daza

Sistemas de Ecuaciones MÃ©todos Iterativos

Ejercicios

Instalar las librerias:

```
library(pracma)
```

```
## Warning: package 'pracma' was built under R version 3.5.3
```

```
library(Matrix)
```

```
## Warning: package 'Matrix' was built under R version 3.5.3
```

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

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

Pracma Proporciona una gran cantidad de funciones de análisis numérico y álgebra lineal, optimización numérica, ecuaciones diferenciales, series de tiempo **matrix** Diversos tipos de matrices, operaciones y algoritmos relacionados

Las siguientes funciones sirven para crear(generar) matrices básicas. Los argumentos son n,m. Donde, son escalares numéricos que especifican el tamaño de la matriz. Por defecto si sólo se introduceel valor de n, se asume cuadrada la matriz

```
n=2#dimensión de la matriz
D1<-eye(n, m = n)#matriz aii=1 y aij=0
D2<-ones(n, m = n)#matriz aij=1
D3<-zeros(n, m = n)#matriz aij=0
print("D1")
```

```
## [1] "D1"
```

```
print(D1)
```

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

```r
print("D2")
```

```
## [1] "D2"
```

```r
print(D2)
```

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

```r
print("D3")
```

```
## [1] "D3"
```

```r
print(D3)
```

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

```r
4*D1#matriz donde aii=4 y aij=0
```

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

b. Evalue la matriz de transición para el métodos:
   ## Jacobi
   Primero crear la matriz A, para el sistema de la forma $AX = B$

```r
A = matrix(c(8, 9, 2, 2, 7, 2,
2, 8, 6), nrow=3, byrow=TRUE)
A
```

```
##      [,1] [,2] [,3]
## [1,]    8    9    2
## [2,]    2    7    2
## [3,]    2    8    6
```

```r
b = c(69, 47,68)
b
```

```
## [1] 69 47 68
```

```r
det(A)#verificar que el sistema tenga solución
```

```
## [1] 140
```

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

```
## [1] 2 5 4
```

Como podemos observar el sistema tiene solución, dado que su determinante es diferente de cero.Y su solución es:$x_1 = 2; x_2 = 5; x_3 = 4$
Entonces, para determinar si el método de Jacobi converge es necesario hallar la mátriz de transición(T_J). Primero podemos crear una función que dada una matriz M retorne en una matriz diagonal

```
diagonal1 <- function(M) {

  M[col(M)!=row(M)] <- 0

  return(M)
}
diagonal1(A)
```

```
##      [,1] [,2] [,3]
## [1,]    8    0    0
## [2,]    0    7    0
## [3,]    0    0    6
```

La otra opción es utilizar la funcionalidad de las librerias que retornan una matriz diagonal. Posteriormente, la matriz asociada al sistema A se descompone en: $A = L + U + D$:

```
D = diagonal1(A)
D
```

```
##      [,1] [,2] [,3]
## [1,]    8    0    0
## [2,]    0    7    0
## [3,]    0    0    6
```

```
L = tril(A,k=-1,diag = FALSE)
L
```

```
## 3 x 3 Matrix of class "dtrMatrix"
##      [,1] [,2] [,3]
## [1,]    0    .    .
## [2,]    2    0    .
## [3,]    2    8    0
```

```
U = triu(A,k=1,diag = FALSE)
U
```

```
## 3 x 3 Matrix of class "dtrMatrix"
##      [,1] [,2] [,3]
## [1,]    0    9    2
## [2,]    .    0    2
## [3,]    .    .    0
```

```r
D+L+U#verificación retorna A
```

```
## 3 x 3 Matrix of class "dgeMatrix"
##      [,1] [,2] [,3]
## [1,]    8    9    2
## [2,]    2    7    2
## [3,]    2    8    6
```

Luego la matriz de transición para el caso del métdodo de Jacobi esta dada por:$T_J = D^{-1} * (L + U)$

```r
TJ = (-solve(D))%*%(L+U)
TJ
```

```
## 3 x 3 Matrix of class "dgeMatrix"
##            [,1]      [,2]       [,3]
## [1,]  0.0000000 -1.125000 -0.2500000
## [2,] -0.2857143  0.000000 -0.2857143
## [3,] -0.3333333 -1.333333  0.0000000
```

La pregunta ahora a resolver es que este sistema al aplicarle un metodo iterativo converge la solución. Para ésto aplicamos el teorema de convergencia (5.1):

```r
print("Norma")
```

```
## [1] "Norma"
```

```r
print(norm(TJ,"I"))
```

```
## [1] 1.666667
```

Como podemos ver la norma es mayor que uno ||T||>1 entonces, la convergencia del método de Jacobi **no se puede asegurar**

```r
propios <- function(matriz) {
  a <- eigen(matriz)#utilizar la funcion eigen
  names(a$values) <- 1:length(a$values)#genera valores
  names(a) <- c("valores","vectores")
  colnames(a$vectores) <- 1:nrow(a$vectores)
  a
}
```

Ahora hay que tener en cuenta que la salida es una lista. La vamos aplicar sobre la matriz de transisición y verificar si el método de Jacobi realmente no converge. Para esto utilizamos el radio espectral

$$p(\alpha)$$

si el maximo valor de sus valores absolutos son menores que uno entonces, el método de jacobi converge

```
y <- TJ
propios(matriz=y) # O simplemente:
```

```
## eigen() decomposition
## $valores
##          1          2          3
## -0.9945840  0.7066154  0.2879685
##
## $vectores
##              1          2          3
## [1,] 0.6017986  0.5486879  0.72241339
## [2,] 0.3753839 -0.4944419 -0.03141823
## [3,] 0.7049293  0.6741430 -0.69074727
```

```
propios(y)
```

```
## eigen() decomposition
## $valores
##          1          2          3
## -0.9945840  0.7066154  0.2879685
##
## $vectores
##              1          2          3
## [1,] 0.6017986  0.5486879  0.72241339
## [2,] 0.3753839 -0.4944419 -0.03141823
## [3,] 0.7049293  0.6741430 -0.69074727
```

Como se puede observar los valores absolutos de los valores propios que es:radio espectral=0.9945840<1. Por lo tanto, el método de Jcobi converje. No obstante, como radio espectral es muy cercano a 1, se tiene que se necesitarán varias iteraciones para resolverlo por este método. Supongase que tiene tolerancia de epsilon de la maquina

```
Solucion_J<-itersolve(A,b,nmax=3000,
                      tol =.Machine$double.eps^(0.3125)
                      ,method = "Jacobi" )
Solucion_J
```

```
## $x
## [1] 1.999940 4.999963 3.999930
##
## $iter
## [1] 2084
##
## $method
## [1] "Jacobi"
```

Como se puede observar la solución es aproximada a la solución exacta, pero es necesaria 2084 iteraciones para obtener una aprociamación con la tolerancia deseada, lo que demuestra lo ineficiente del método de Jacobi cuando el radio espectral es muy cercano a 1. ## Método de Gauss_Seidel
Para el caso del método de Gauss_Seidel esta dada por:
$T_{gs} = (I + D^{-1}L)^{-1}(-D^{-1}U)$

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

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

```
z = c(69, 47,68,11)
det(C)#verificar que el sistema tenga solución
```

```
## [1] 125
```

```
solve(C,z)
```

```
## [1] 52.8 48.4 52.6 41.2
```

Se va aplicar para el sistema asociado $CX = z$, para esto Para el caso del método de Gauss_Seidel esta dada por:
$T_{gs} = (I + D^{-1}L)^{-1}(-D^{-1}U)$. Luego primero los valores propios y luego el radio espectral

```
n=4#dimensión de la matriz
Ig<-eye(n, m = n)
Dg = diagonal1(C)
Lg = tril(C,k=-1,diag = FALSE)
Ug = triu(C,k=1,diag = FALSE)
W=(Ig+(solve(Dg)%*%Lg))
H=solve(W)
N=(-solve(Dg))%*%(Ug)
TG<- H%*%N
TG#matriz de transisción
```

```
## 4 x 4 Matrix of class "dgeMatrix"
##      [,1]       [,2]      [,3]      [,4]
## [1,]    0 0.25000000 0.2500000 0.2500000
## [2,]    0 0.06250000 0.3125000 0.3125000
## [3,]    0 0.07812500 0.1406250 0.3906250
## [4,]    0 0.09765625 0.1757812 0.2382812
```

Luego la matriz de tolerancia esta dada porTG y sus valores propiosestan a continuación y el radio espectral esta dado por 0.56994495, lo que indica que el método converje

```
y <- TG
propios(y)
```

```
## eigen() decomposition
## $valores
##                       1                       2                       3
```

```
##  0.56994495+0.00000000i -0.06426935+0.05218412i -0.06426935-0.05218412i
##                       4
##  0.00000000+0.00000000i
##
## $vectores
##                 1                     2                     3    4
## [1,] 0.6024566+0i  0.8468693+0.0000000i  0.8468693+0.0000000i 1+0i
## [2,] 0.5234606+0i -0.2620539+0.3710560i -0.2620539-0.3710560i 0+0i
## [3,] 0.4548228+0i -0.0814887-0.2296380i -0.0814887+0.2296380i 0+0i
## [4,] 0.3951850+0i  0.1258317+0.0353545i  0.1258317-0.0353545i 0+0i
```

Al aplicar el métdo utilizando las iteraciones de Gauss_seidel

```
Solucion_G<-itersolve(C,z,nmax=3000,
                      tol =.Machine$double.eps^(0.3125)
                      ,method = "Gauss-Seidel" )
Solucion_G
```

```
## $x
## [1] 52.79952 48.39959 52.59964 41.19969
##
## $iter
## [1] 21
##
## $method
## [1] "Gauss-Seidel"
```

Como se puede ver fueron necesarias 21 iteraciones utilizando el métod de Gauss-Seidel.
Método SOR
En este método la

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

```
w=1.2
S=(solve((Dg+w*Lg)))
Tsor<- S%*%((1-w)*Dg-w*Ug)
propios(Tsor)
```

```
## eigen() decomposition
## $valores
##                     1                     2                     3
##  0.3312377+0.0000000i -0.2688189+0.0708682i -0.2688189-0.0708682i
##                     4
##  0.0625000+0.0000000i
##
## $vectores
##                 1                     2                     3            4
## [1,] 0.6904918+0i  0.7157766+0.0000000i  0.7157766+0.0000000i 0.8677218+0i
## [2,] 0.5238337+0i -0.3430855+0.3904187i -0.3430855-0.3904187i 0.4338609+0i
## [3,] 0.3974005+0i -0.0485055-0.3742705i -0.0485055+0.3742705i 0.2169305+0i
## [4,] 0.3014834+0i  0.2273946+0.1529378i  0.2273946-0.1529378i 0.1084652+0i
```

como se ve el radio espectral para este caso es 0.3312377<1. Luego se va determinar la aproximación a la solución utilizando los tres métodos para comparar el número de iteraciones

```
library(Rlinsolve)
```

```
## Warning: package 'Rlinsolve' was built under R version 3.5.3
```

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

```
## Warning: package 'bigmemory' was built under R version 3.5.3
```

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

Con la librería Rlinsolve para determinar la solución por SOR

```
solucion_sor<- lsolve.sor(C,z,reltol = .Machine$double.eps^(0.3125),w=1.2)
```

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

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

```
solucion_sor
```

```
## $x
##          [,1]
## [1,] 52.79981
## [2,] 48.39985
## [3,] 52.59989
## [4,] 41.19991
##
## $iter
## [1] 11
##
## $errors
##                [,1]
##  [1,] 7.383016e-01
##  [2,] 2.878766e-01
##  [3,] 9.191226e-02
##  [4,] 3.235710e-02
##  [5,] 1.014395e-02
##  [6,] 3.524247e-03
##  [7,] 1.121867e-03
##  [8,] 3.829031e-04
##  [9,] 1.241788e-04
## [10,] 4.166934e-05
## [11,] 1.371648e-05
## [12,] 4.547767e-06
```

```
solve(C,z)
```

```
## [1] 52.8 48.4 52.6 41.2
```

Como se puede observar se necesitan 11 interaciones para obtener la aproximación a la solución del sistema.

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 = FALSE)] <- 0
} else {
M[col(M) >= row(M) + k + 1] <- 0
}
return(M)
}
tril1(C)#aplicada sobre la matriz C
```

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

Dado el siguiente sistema:

$$2x - z = 1$$

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

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

El valor de α y β para asegura la convergencia por el método de Jacobi: Si la matriz del sistema original es diagonalmente dominante, seguro converge por metodo de Jacobi.La matriz diagonalmente dominante se denomina como aquella matriz en la cual cada valor de su diagonal debe ser mayor que la suma de los otros coeficientes de la fila en la que se encuentra el valor. En este caso beta debe ser:

$$|\beta| > |-1| + |2|$$
$$\alpha > |-1| + |1|$$

```
beta = 4
alpha = 3

A = matrix(c(2, 0, -1,
             beta,2 , -1,
             -1, 1, alpha), 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,]    4    2   -1    2
## [3,]   -1    1    3    1
```

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

```
## [1]  71.5 -82.5  74.0
```

Genere una tabla que tenga 10 iteraciones del método de Jacobi con vector inicial $x_0 = [1, 2, 3]^t$

```
x = 0
y = 0
z = 0

diag1 <- function(M) {

  M[col(M)!=row(M)] <- 0

  return(M)
}

jacobiPr2 <- function(A,B, x0, tol){
  x_k = matrix(x0)

  D = diag1(A)
  L = tril(A,k=-1,diag = FALSE)
  U = triu(A,k=1,diag = FALSE)

  it = 1
  repeat
  {
    inn = matrix(B-((L+U)%*%x_k))
    D1 = (solve(D))
    xk1 = D1%*%inn
    cat("Error ",it," ",norm(xk1-x_k,"F")/norm(x_k),"\n")
    x_k = xk1

    x[[it]] = x_k[1]
    y[[it]] = x_k[2]
    z[[it]] = x_k[3]
    cat("Solucion iteracion ",it,": ",x[[it]]," ",y[[it]]," ",z[[it]],"\n")
    it = it + 1

    if(it == tol)
      break
  }

  cat("Solucion a ", tol ," iteraciones: ",x_k,"\n")
}

x1 = c(1,2,3)
jacobiPr2(A, B, x1, 10)
```

```
## Error  1   0.5833333
## Solucion iteracion  1 :  2   0.5   0
## Error  2   1.559202
## Solucion iteracion  2 :  0.5   -3   0.8333333
## Error  3   0.8090648
## Solucion iteracion  3 :  0.9166667   0.4166667   1.5
## Error  4   0.4117647
## Solucion iteracion  4 :  1.25   -0.08333333   0.5
## Error  5   0.7087282
## Solucion iteracion  5 :  0.75   -1.25   0.7777778
## Error  6   0.4207137
## Solucion iteracion  6 :  0.8888889   -0.1111111   1
## Error  7   0.1944444
## Solucion iteracion  7 :  1   -0.2777778   0.6666667
## Error  8   0.2227432
## Solucion iteracion  8 :  0.8333333   -0.6666667   0.7592593
## Error  9   0.1724236
## Solucion iteracion  9 :  0.8796296   -0.287037   0.8333333
## Solucion a  10  iteraciones:  0.8796296 -0.287037 0.8333333
```

```
Solucion_J2<-itersolve(A,b=B,x0=x1,nmax=1000,
                       tol = 1e-8,method = "Jacobi" )
Solucion_J2
```

```
## $x
## [1]  0.875 -0.375  0.750
##
## $iter
## [1] 51
##
## $method
## [1] "Jacobi"
```

Solución de un sistema No lineal

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 pauqte BB y la función BBsolve() del paquete,grafique la solución

```
library(BB)
```

```
## Warning: package 'BB' was built under R version 3.5.3
```

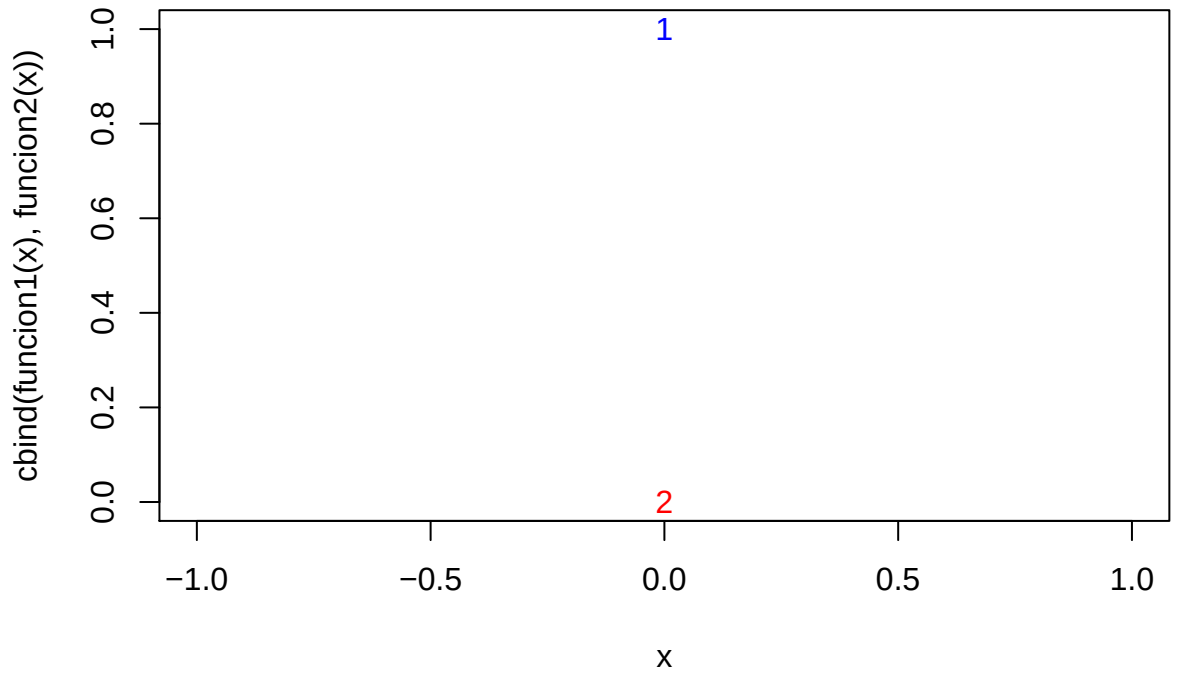
```
ecuaciones = function(x) {
  n = length(x)
  F = rep(NA, n)
  F[1] = x[1] - x[2]
  F[2] = x[1]^2 + x[2]^2 -1
  F
}
p0 = c(1,1)
sol = BBsolve(par = p0, fn=ecuaciones)
```

```
##   Successful convergence.
```

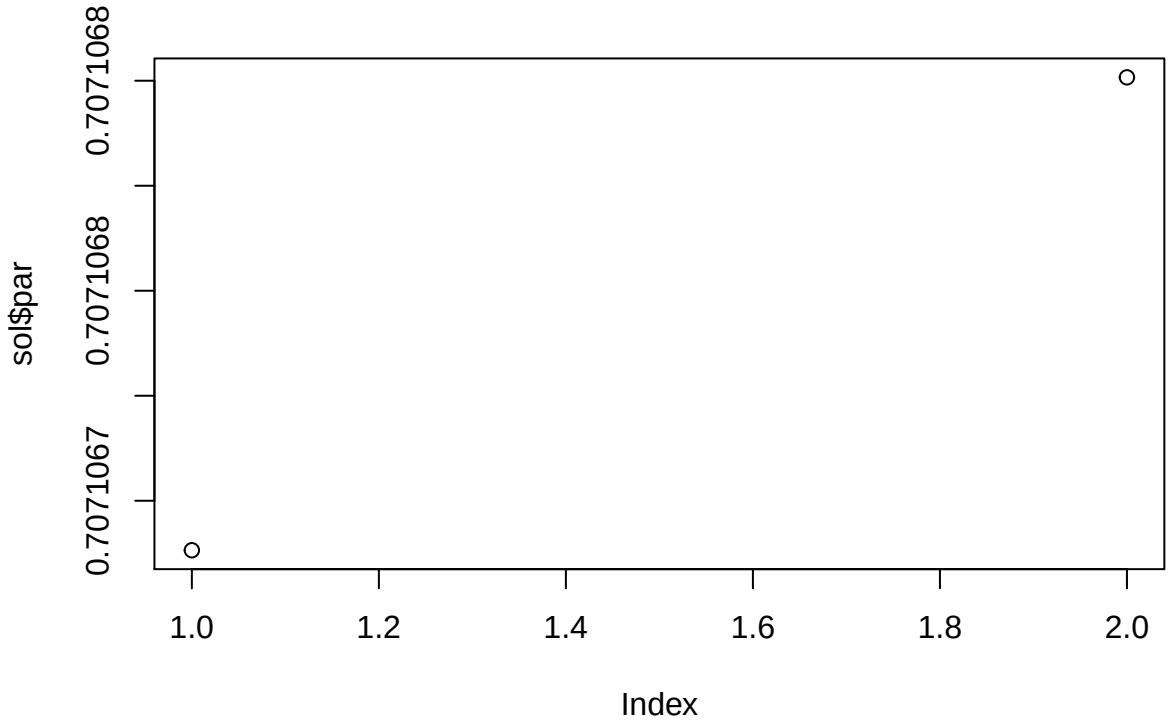
```
sol$par
```

```
## [1] 0.7071067 0.7071068
```

```
funcion1 <- function(x) sqrt(1 - x^2)
funcion2 <- function(x) x
matplot(x,cbind(funcion1(x),funcion2(x)),col=c("blue","red"))
```




```
plot(sol$par)
```




```
sol$par
```

```
## [1] 0.7071067 0.7071068
```