PEC 3 - Métodos numéricos

Félix José Villalba Espinosa

2022-03-15

Ejercicios

1. Resolución de sistemas de ecuaciones lineales.

Tarea 1.1: Ax = b

$$\begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 0 & -3 & 1 & 2 & 2 \\ 2 & -2 & -4 & 2 & 7 \\ 3 & -3 & -1 & 5 & 9 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} -2 \\ 3 \\ -3 \\ 13 \\ 2 \end{pmatrix}$$

La descomposición de LU de A es:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 \\ 3 & 3 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} U = \begin{pmatrix} 1 & 2 & -2 & -1 & 1 \\ 0 & -3 & 1 & 2 & 2 \\ 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

Solución del sistema con métodos directos:

$$x_1 = 1x_2 = 2x_3 = 3x_4 = 2x_5 = 1$$

Descomposición y resolución a partir de L y U:

```
## Solucion de sistema de ecuaciones mediante metodos directos

# Resolucion del sistema mediante factorizacion LU
install.packages("pracma")
library(pracma)

require(pracma)
# Definicion del  sistema Ax = b
A = matrix (c(1, 2,-2,-1, 1,
              0,-3, 1, 2, 2,
              2,-2,-4, 2, 7,
              3,-3,-1, 5, 9,
              0, 0, 0, 0, 2) , nrow =5, byrow = TRUE )
```

```
b = c(-2,
       3,
      -3,
      13,
       2)

#Factorizacion LU

mlu = lu(A, scheme='ijk') # metodo de Doolittle

L = mlu$L
U = mlu$U

print(L)
print(U)

# Resolucion del sistema a partir de L y U

y = solve(L, b) #Ly = b                                    #Los dos sistemas dan el mismo resultado#
x = solve(U, y) #Ux = y

print(x)
```

Resolución directa con solve:

```
#Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve
x_ = solve(A, b) #Ax_ = b

print(x_)
```

Código completo

```
# Solucion de sistema de ecuaciones mediante metodos directos

# Resolucion del sistema mediante factorizacion LU
install.packages("pracma")
```

```
## package 'pracma' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\felix\AppData\Local\Temp\RtmpSGyDlD\downloaded_packages
```

```
library(pracma)
```

```
## Warning: package 'pracma' was built under R version 4.1.3
```

```
require(pracma)
# Definicion del  sistema Ax = b
A = matrix (c(1, 2,-2,-1, 1,
```

```
              0,-3, 1, 2, 2,
              2,-2,-4, 2, 7,
              3,-3,-1, 5, 9,
              0, 0, 0, 0, 2) , nrow =5, byrow = TRUE )

b = c(-2,
       3,
      -3,
      13,
       2)

#Factorizacion LU

mlu = lu(A, scheme='ijk') # metodo de Doolittle

L = mlu$L
U = mlu$U

print(L)
```

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

```
print(U)
```

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

```
# Resolucion del sistema a partir de L y U

y = solve(L, b) #Ly = b                                  #Los dos sistemas dan el mismo resultado#
x = solve(U, y) #Ux = y

print(x)
```

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

```
#Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve
x_ = solve(A, b) #Ax_ = b

print(x_)
```

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

Tarea 1.2: En esta tarea se pide resolver el sistema (S) por el método de Jacobi y por el método de Gauss-Seidel. Además, se pide comparar la convergencia de ambos métodos.

Primera interacción de Gauss-Seidel, tomando $x^0 = (0,0,0,0,0,0,0,0,0,0)^T$

```
## Solucion de sistema de ecuaciones mediante metodos iterativos
# Empleamos el comando de R itersolve(A, b, x0, tol, method), cuyos argumentos son:
# A: la matriz del sistema
# b: el lado derecho del sistema
# x0: aproximacio inicial
# tol: condici?n de parada en base al error cometido
# method: metodo a utilizar ("Gauss-Seidel" o "Jacobi")

require(pracma)


# Resolucion del sistema mediante Jacobi

n = 10 #20 #40, #dimension del sistema

A = diag(n)

print(A)

for (i in 1:n-1) { A[i,i+1]<-1/2}
for (i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2
```

$$Dx^{(k+1)} = (L+U)x^{(k)} + b$$

por lo que:

$$J = D^{-1}(L+U)yc = D^{-1}b$$

para este caso :

J =

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]

[1,] 0.0 -0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

[2,] -0.5 0.0 -0.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0

[3,] 0.0 -0.5 0.0 -0.5 0.0 0.0 0.0 0.0 0.0 0.0

[4,] 0.0 0.0 -0.5 0.0 -0.5 0.0 0.0 0.0 0.0 0.0

[5,] 0.0 0.0 0.0 -0.5 0.0 -0.5 0.0 0.0 0.0 0.0

[6,] 0.0 0.0 0.0 0.0 -0.5 0.0 -0.5 0.0 0.0 0.0

[7,] 0.0 0.0 0.0 0.0 0.0 -0.5 0.0 -0.5 0.0 0.0

[8,] 0.0 0.0 0.0 0.0 0.0 0.0 -0.5 0.0 -0.5 0.0

[9,] 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.5 0.0 -0.5

[10,] 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.5 0.0

C =

[,1]

[1,] 0.5

[2,] 0.0

[3,] 0.0

[4,] 0.0

[5,] 0.0

[6,] 0.0

[7,] 0.0

[8,] 0.0

[9,] 0.0

[10,] 0.0

```
## Solucion de sistema de ecuaciones mediante metodos iterativos
# Empleamos el comando de R itersolve(A, b, x0, tol, method), cuyos argumentos son:
# A: la matriz del sistema
# b: el lado derecho del sistema
# x0: aproximacio inicial
# tol: condici?n de parada en base al error cometido
# method: metodo a utilizar ("Gauss-Seidel" o "Jacobi")

require(pracma)


# Resolucion del sistema mediante Jacobi

n = 10 #20 #40, #dimension del sistema

A = diag(n)

print(A)

for (i in 1:n-1) { A[i,i+1]<-1/2}
for (i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2

# Primera iteracion

D = diag(diag(A))
L = -tril(A, -1)
```

```
U = -triu(A, 1)

J = inv(D) %*% (L+U)
c = inv(D) %*% b

print(J)
print(c)
```

Resultados

2. Método Jacobi con

$$10^{-6}$$

[1] 0.9090899 -0.8181799 0.7272699 -0.6363604 0.5454509 -0.4545420 0.3636330 -0.2727246 0.1818162 -0.0909081

$iter

[1] 292

Con número máximo interacciones 80:

[1] 0.90243829 -0.80593273 0.70942717 -0.61575467 0.52208217 -0.43212000 0.34215782 -0.25560518 0.16905254 -0.08452627

$iter

[1] 80

3. Método Gaus-Seidel $ 10^{-6}: $

$x

[1] 0.90908934 -0.81817893 0.72726885 -0.63635916 0.54544988 -0.45454097 0.36363241 -0.27272412 0.18181602 -0.09090801

$iter

[1] 141

Limitado a 80 iteraciones:

$x

[1] 0.90884751 -0.81773365 0.72667162 -0.63566945 0.54472976 -0.45385002 0.36302316 -0.27223844 0.18148265 -0.09074133

$iter

[1] 80

4. El método Gauss-Seidel es capaz de dar el mismo resultado con menos interacciones, 141.

5. Método Jacobi

n = 20 Da como resultado 987 iteraciones y

[1] 0.95237951 -0.90475902 0.85713866 -0.80951829 0.76189818 -0.71427806 0.66665828 -0.61903850 0.57141913 -0.52379976 0.47618082 -0.42856188

[13] 0.38094337 -0.33332485 0.28570671 -0.23808858 0.19047074 -0.14285289 0.09523524 -0.04761759

a 80 interacciones

[1] 0.91107020 -0.82430808 0.73754596 -0.65697567 0.57640539 -0.50518878 0.43397216 -0.37404782 0.31412347 -0.26609499 0.21806652 -0.18132462

[13] 0.14458272 -0.11758503 0.09058734 -0.07119181 0.05179627 -0.03755302 0.02330977 -0.01165488

n = 40

[1] 9.110721e-01 -8.243132e-01 7.375543e-01 -6.569925e-01 5.764306e-01 -5.052364e-01 4.340423e-01 -3.741744e-01 3.143066e-01 -2.664123e-01

[11] 2.185180e-01 -1.820767e-01 1.456355e-01 -1.192737e-01 9.291188e-02 -7.478815e-02 5.666443e-02 -4.482852e-02 3.299262e-02 -2.565436e-02

[21] 1.831610e-02 -1.399947e-02 9.682850e-03 -7.275502e-03 4.868154e-03 -3.596348e-03 2.324541e-03 -1.688638e-03 1.052734e-03 -7.521242e-04

[31] 4.515141e-04 -3.173083e-04 1.831024e-04 -1.265747e-04 7.004703e-05 -4.755576e-05 2.506449e-05 -1.641539e-05 7.766293e-06 -3.883147e-06

$iter

[1] 80

Da error

Método Gauss-Seidel

n = 20

$x

[1] 0.95237874 -0.90475758 0.85713656 -0.80951573 0.76189511 -0.71427475 0.66665465 -0.61903485 0.57141535 -0.52379615 0.47617725 -0.42855864

[13] 0.38094031 -0.33332223 0.28570437 -0.23808671 0.19046921 -0.14285182 0.09523452 -0.04761726

$iter

[1] 475

Con 80 interacciones:

$x

[1] 0.93652661 -0.87376615 0.81204193 -0.75164581 0.69283243 -0.63581441 0.58075881 -0.52778459 0.47696141 -0.42830943 0.38180029 -0.33735906

[13] 0.29486711 -0.25416591 0.21506142 -0.17732921 0.14072006 -0.10496586 0.06978585 -0.03489292

$iter

[1] 80

con n = 40

$x

[1] 0.97547302 -0.95094764 0.92642465 -0.90190480 0.87738885 -0.85287750 0.82837143 -0.80387130 0.77937771 -0.75489124 0.73041239 -0.70594165

[13] 0.68147945 -0.65702615 0.63258206 -0.60814746 0.58372255 -0.55930747 0.53490230 -0.51050709 0.48612180 -0.46174634 0.43738058 -0.41302430

[25] 0.38867725 -0.36433913 0.34000959 -0.31568820 0.29137453 -0.26706807 0.24276830 -0.21847465 0.19418651 -0.16990325 0.14562422 -0.12134874

[37] 0.09707612 -0.07280566 0.04853662 -0.02426831

$iter

[1] 1000

Con 80 interacciones:

$x

[1] 0.937020169 -0.874817867 0.813753412 -0.754160389 0.696340968 -0.640562233 0.587053561 -0.536005057 0.487566985 -0.441850150 0.398927115

[12] -0.358834159 0.321573841 -0.287118033 0.255411311 -0.226374555 0.199908643 -0.175898147 0.154214918 -0.134721485 0.117274219 -0.101726194

[23] 0.087929728 -0.075738576 0.065009770 -0.055605112 0.047392324 -0.040245879 0.034047537 -0.028686598 0.024059922 -0.020071719 0.016633158

[34] -0.013661810 0.011080950 -0.008818760 0.006807427 -0.004982190 0.003280333 -0.001640166

$iter

[1] 80

$method

[1] "Gauss-Seidel"

Código completo apartado 1.2:

```
#=====================================================APARTADO 2=========================================

## Solucion de sistema de ecuaciones mediante metodos iterativos
# Empleamos el comando de R itersolve(A, b, x0, tol, method), cuyos argumentos son:
# A: la matriz del sistema
# b: el lado derecho del sistema
# x0: aproximacio inicial
# tol: condici?n de parada en base al error cometido
# method: metodo a utilizar ("Gauss-Seidel" o "Jacobi")

require(pracma)


# Resolucion del sistema mediante Jacobi

n = 10 #20 #40, #dimension del sistema

A = diag(n)

print(A)

for (i in 1:n-1) { A[i,i+1]<-1/2}
for (i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2

# Primera iteracion
```

```
D = diag(diag(A))
L = -tril(A, -1)
U = -triu(A, 1)

J = inv(D) %*% (L+U)
c = inv(D) %*% b

print(J)
print(c)


#Calculamos los autovalores de J

lambda <-eigen(J,only.values=TRUE)

print(lambda)

#calculamos el radio espectral (el máximo de los valores absolutos de los autovalores) # https://www.ug
max_autovalor = max(abs(J))

print(max_autovalor)

# Primera iteracion

x0 = rep(0,n)
x1 = J%*%x0 + c

print(x1)


# Resolucion iterativa
sol = itersolve(A, b, x0, tol =, method = "Jacobi") #solucion con un error de aproximacion maximo
print(sol)
sol1 = itersolve(A, b, x0, nmax = , method = "Jacobi") #Solución con un numero maximo de iteraciones
print(sol1)

#calcular el error relativo
x_ = solve (A, b)
dif_J = sol$x -x_
error_J = norm(dif_J, "2")
error_rel_J = error_J/norm(x_, "2")
print(error_rel_J)

# Resolucion del sistema mediante Gauss-Seidel

n =  10 #20 #40 #dimension del sistema

A = diag(n)
for(i in 1:n-1) { A[i,i+1]<-1/2}
for(i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2
```

```
print(b)

# Primera iteracion
D = diag(diag(A))
L = -tril(A, -1)
M = D - L
U = -triu(A, 1)

G = inv (M)%*%U          #M^{-1}*U
d = inv (M)%*%b          #M^{-1}*b

print (G)
print (d)

x0 = rep(0,n)
x1 = G %*% x0 + d

print(x0)
print(x1)

# Resolucion iterativa
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Gauss-Seidel")#solucion con un error de aproximacion m
print(sol)
sol1 = itersolve(A, b, x0, nmax = , method = "Gauss-Seidel")#Solución con un numero maximo de iteracion
print(sol1)

#calculo del error relativo     Se realiza en el último momento para incluir x_


#Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve
x_ = solve(A, b)

dif_GS = sol$x -x_

print(dif_GS)

# Error relativo
error_GS = norm(dif_GS, "2")

error_rel_GS = error_GS/norm (x_, "2")

print(error_rel_GS)
```

Código completo de la PEC 3:

```
## Solucion de sistema de ecuaciones mediante metodos directos

# Resolucion del sistema mediante factorizacion LU
install.packages("pracma")
library(pracma)
```

```
require(pracma)
# Definicion del  sistema Ax = b
A = matrix (c(1, 2,-2,-1, 1,
              0,-3, 1, 2, 2,
              2,-2,-4, 2, 7,
              3,-3,-1, 5, 9,
              0, 0, 0, 0, 2) , nrow =5, byrow = TRUE )

b = c(-2,
       3,
      -3,
      13,
       2)

#Factorizacion LU

mlu = lu(A, scheme='ijk') # metodo de Doolittle

L = mlu$L
U = mlu$U

print(L)
print(U)

# Resolucion del sistema a partir de L y U

y = solve(L, b) #Ly = b                                  #Los dos sistemas dan el mismo resultado#
x = solve(U, y) #Ux = y

print(x)

#Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve
x_ = solve(A, b) #Ax_ = b

print(x_)

#=================================================APARTADO 2=========================================

## Solucion de sistema de ecuaciones mediante metodos iterativos
# Empleamos el comando de R itersolve(A, b, x0, tol, method), cuyos argumentos son:
# A: la matriz del sistema
# b: el lado derecho del sistema
# x0: aproximacio inicial
# tol: condici?n de parada en base al error cometido
# method: metodo a utilizar ("Gauss-Seidel" o "Jacobi")

require(pracma)


# Resolucion del sistema mediante Jacobi

n = 10 #20 #40, #dimension del sistema
```

```
A = diag(n)

print(A)

for (i in 1:n-1) { A[i,i+1]<-1/2}
for (i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2

# Primera iteracion

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

J = inv(D) %*% (L+U)
c = inv(D) %*% b

print(J)
print(c)


#Calculamos los autovalores de J

lambda <-eigen(J,only.values=TRUE)

print(lambda)

#calculamos el radio espectral (el máximo de los valores absolutos de los autovalores) # https://www.ug
max_autovalor = max(abs(J))

print(max_autovalor)

# Primera iteracion

x0 = rep(0,n)
x1 = J%*%x0 + c

print(x1)


# Resolucion iterativa
sol = itersolve(A, b, x0, tol =, method = "Jacobi") #solucion con un error de aproximacion maximo
print(sol)
sol1 = itersolve(A, b, x0, nmax = , method = "Jacobi") #Solución con un numero maximo de iteraciones
print(sol1)

#calcular el error relativo
x_ = solve (A, b)
dif_J = sol$x -x_
error_J = norm(dif_J, "2")
error_rel_J = error_J/norm(x_, "2")
```

```
print(error_rel_J)

# Resolucion del sistema mediante Gauss-Seidel

n =  10 #20 #40 #dimension del sistema

A = diag(n)
for(i in 1:n-1) { A[i,i+1]<-1/2}
for(i in 2:n) { A[i,i-1]<-1/2}

b = rep(0,n)
b[1]<-1/2

print(b)

# Primera iteracion
D = diag(diag(A))
L = -tril(A, -1)
M = D - L
U = -triu(A, 1)

G = inv (M)%*%U          #M^{-1}*U
d = inv (M)%*%b          #M^{-1}*b

print (G)
print (d)

x0 = rep(0,n)
x1 = G %*% x0 + d

print(x0)
print(x1)

# Resolucion iterativa
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Gauss-Seidel")#solucion con un error de aproximacion m
print(sol)
sol1 = itersolve(A, b, x0, nmax = , method = "Gauss-Seidel")#Solución con un numero maximo de iteracion
print(sol1)

#calculo del error relativo     Se realiza en el último momento para incluir x_


#Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve
x_ = solve(A, b)

dif_GS = sol$x -x_

print(dif_GS)

# Error relativo
error_GS = norm(dif_GS, "2")

error_rel_GS = error_GS/norm (x_, "2")
```

```
print(error_rel_GS)
```