PEC 3 Resolución de sistemas lineales

Alan Coila Bustinza

2022-03-11

Resolución numérica de sistemas de ecuaciones lineales

1. Métodos numéricos directos

Comenzamos creando las variablee A y b, que contendran las matrices que nos plantea el ejercicio

```
# Asignamos a la variable A la matriz que plantea el problema
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), ncol = 5,
            byrow=TRUE)
# Asignamos a la varible b la matriz correspondiente
b<-matrix(c(-2,3,-3,13,2),byrow=FALSE)
```

Primero debemos corroborar que la matriz A es invertible o no singular, esto lo descartamos calculado su determinante el cual debe ser no nulo.

```
# calculamos su determinante con la función det()
det(A)
```

```
## [1] 24
```

La martriz A tiene un determinante diferente de 0 por lo que podemos continuar la tarea de resolver de sistemas de ecuaciones lineales. El método de factorización LU, plantea que mediante la obtención de dos matrices triangulares, una inferior L(lower) con "1" en su diagonal y una superior U (upper).

$$A = LU$$

De manera que podemos resolver el sistema reemplazando

$$Ax = b$$
$$LUx = b$$

De donde obtenemos dos identidades:

$$Ux = y \qquad Ly = b$$

A partir de de estas podemos resolver el sistema de ecuaciones

Para ello instalamos e importamos el paquete pracma

```
install.packages("pracma")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.1'
## (as 'lib' is unspecified)
```

```
library(pracma)
```

De este paquete, usaremos la función lu() que recibe como argumentos, una matriz y el argumento scheme qu

```
# realizamos la factorizacion mediante la función lu y asignamos
# las variables correspondientes
mlu<-lu(A, scheme='ijk') # metodo de Doolittle

#lu() retorna dos matrices Lower y Upper, que almacenamos en  y U
L<-mlu$L
U<-mlu$U
```

Reemplazamos los valores que hemos obtenidoen las ecuaciones previas. Asi obtener dos nuevos sistemas de ecuaciones con el cual resolver el sistema original.

$$Ux = y \qquad\qquad Ly = b$$

$$\begin{bmatrix} 1 & 2 & -2 & -1 & 1 \\ 0 & -3 & 1 & 2 & 2 \\ 0 & 0 & -2 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 \\ 3 & 3 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \\ -3 \\ 13 \\ 2 \end{bmatrix}$$

$$\begin{aligned} x_1 + 2x_2 - 2x_3 - x_4 + x_5 &= y_1 \\ -3x_2 + x_3 + 2x_4 + 2x_5 &= y_2 \\ -2x_3 + x_5 &= y_3 \\ 2x_4 + x_5 &= y_4 \\ 2x_5 &= y_5 \end{aligned} \qquad\qquad \begin{aligned} y_1 &= -2 \\ y_2 &= 3 \\ 2y_1 + 2y_2 + y_3 &= -3 \\ 3y_1 + 3y_2 - y_3 + y_4 &= 13 \\ y_5 &= 2 \end{aligned}$$

Para resolver ambos sistemas usaremos la funcion solve()

```
y = solve(L,b) #Ly = b
x = solve(U,y) #Ux = y
x
```

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

Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve

```
x_ = solve(A,b) #Ax_ = b
x_
```

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

1.2. Metodos numéricos iterativos

Construimos la matriz del problema, para la resolucion de ambos metodos, asi como tambien las matrices auxiliares triangulares y diagonales

```
n = 10 #dimension del sistema

A = diag(n) #construimos la matriz A, primero añadiendo los elementos de su diagonal

for(i in 1:n-1) { A[i,i+1]<--1/2 } #ahora añadimos los elementos superiores e inferiores
for(i in 1:n) { A[i,i-1]<-1/2 }
```

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

```
D = diag(diag(A)) # creamos la matriz diagonal
L = -tril(A, -1) # ahora creamos la matriz trigangular superior e inferior con el signo cambiado
U = -triu(A, 1)
```

Pregunta 1

Construcción de la matriz de Jacobi

El metodo Jacobi que esta dada por:

$$x^{k+1} = Jx^k + D^{-1}b$$

Donde la matriz de Jacobi:

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

```
J = inv(D)%*%(L+U) # mediante la inversa de la matriz D y el producto punto de la suma de las matrices
c = inv(D)%*%b
```

Convergencia

Vamos a estudiar la convergencia para este sistema de ecuaciones mediante radio espectral, por lo cual calcularemos los autovalores de J

```
lambda<-eigen(J)$values # mediante la funcion eigen obtenemos los autovalores
lambda
```

```
##  [1] 0+0.9594930i 0-0.9594930i 0+0.8412535i 0-0.8412535i 0+0.6548607i
##  [6] 0-0.6548607i 0+0.4154150i 0-0.4154150i 0+0.1423148i 0-0.1423148i
```

Ahora obtendremos el maximo valor absoluto de los autovalores de J

```
max_autovalor = max(abs(J))
```

Vemos que es menor que 1 por lo que concluimos que el metodo de Jacobi nos calcula una sucesion de iterantes que convergen en la solucion.

Pregunta 2

Primera iteracion

Considerando el iterante inicial $x^0 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^t$ la primera iteracion seria:

```
x0=rep(0,n) #creamos el vector x0
x1=J%*%x0+c # realizamos la primera iteracion
x1
```

```
##       [,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
```

Iteración de punto fijo $x^{k+1} = Jx^k + c$

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: condición de parada en base al error cometido method: metodo a utilizar ("Gauss-Seidel" o "Jacobi")

Con esta funcion podremos calcular lo solicitado :

las iteraciones para un error 10^{-6}

error $= 10^{-6}$

```
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Jacobi")#solucion con un error de aproximacion maximo
print(sol)
```

```
## $x
##  [1]  0.4142135088 -0.1715728178  0.0710678732 -0.0294375130  0.0121928471
##  [6] -0.0050512403  0.0020903666 -0.0008710387  0.0003482891 -0.0001741446
##
## $iter
## [1] 292
##
## $method
## [1] "Jacobi"
```

El numero de iteraciones es de 292

El error cometido al realizar 80 iteraciones:

```
sol1 = itersolve(A, b, x0, nmax = 8 , method = "Jacobi")#Soluci?n con un numero maximo de iteraciones
print(sol1)
```

```
## $x
##  [1]  0.39843750 -0.14843750  0.10156250 -0.05468750 -0.00781250  0.00781250
##  [7]  0.00781250 -0.00390625  0.00000000  0.00000000
##
## $iter
## [1] 8
##
## $method
## [1] "Jacobi"
```

Calculo del error relativo

Sabemos que el error se puede calcular a partir de $r = b - A\bar{x}$

```
errorJ <- b-A%*%sol1$x # multiplicaremos la matriz A por el vector
                       #de la aproximacion obtenido con 80 iteraciones
```

Por lo que el error es 0.0390625

Comprobación

Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve

```
x_ = solve(A, b)
x_
```

```
##                 [,1]
##  [1,]  0.4142135516
##  [2,] -0.1715728967
##  [3,]  0.0710677582
##  [4,] -0.0294373802
##  [5,]  0.0121929977
##  [6,] -0.0050513848
##  [7,]  0.0020902282
##  [8,] -0.0008709284
##  [9,]  0.0003483714
## [10,] -0.0001741857
```

Pregunta 3

Construcción de la matriz de Gauss-Seidel

El metodo Jacobi que esta dada por:

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

Donde la matriz de Gauss-Seidel:

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

Como previamente hemos creado las matrices comunes al metodo de Jacobi, unicamente crearemos las matrices necesarias

```
M = D - L  #en este caso calculamos la matriz M como la diferencia de la matriz diagonal y la triangula
G = inv(M)%*%U # obtenemos la matriz de Gauss
d = inv(M)%*%b
```

Primera iteracion

Considerando el iterante inicial $x^0 = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^t$ la primera iteracion seria:

```
x0=rep(0,n) #creamos el vector x0
x1=G%*%x0+d # realizamos la primera iteracion
x1
```

```
##                 [,1]
##  [1,]  0.5000000000
##  [2,] -0.2500000000
##  [3,]  0.1250000000
##  [4,] -0.0625000000
##  [5,]  0.0312500000
##  [6,] -0.0156250000
```

```
##  [7,]  0.0078125000
##  [8,] -0.0039062500
##  [9,]  0.0019531250
## [10,] -0.0009765625
```

Iteración de punto fijo $x^{k+1} = Gx^k + d$

Utilizamos el mismo comando descrito previamente

```
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Gauss-Seidel")#solucion con un error de aproximacion m
print(sol)
```

```
## $x
##  [1]  0.4142136165 -0.1715730160  0.0710679183 -0.0294375651  0.0121931907
##  [6] -0.0050515700  0.0020903915 -0.0008710586  0.0003484607 -0.0001742304
##
## $iter
## [1] 141
##
## $method
## [1] "Gauss-Seidel"
```

```
sol1 = itersolve(A, b, x0, nmax = 80 , method = "Gauss-Seidel")#Soluci?n con un numero maximo de iterac
print(sol1)
```

```
## $x
##  [1]  0.4142034926 -0.1715543755  0.0710429166 -0.0294086916  0.0121630445
##  [6] -0.0050226449  0.0020648865 -0.0008507266  0.0003345050 -0.0001672525
##
## $iter
## [1] 80
##
## $method
## [1] "Gauss-Seidel"
```

Calculo del error relativo Sabemos que el error se puede calcular a partir de $r = b - A\bar{x}$

```
errorG <- b-A%*%sol1$x # multiplicaremos la matriz A por el vector
                       #de la aproximacion obtenido con 80 iteraciones
```

Por lo que el error es 2.9978811×10^{-5}

Comprobación

Comprobamos que obtenemos el mismo resultado, resolviendo de forma directa con solve

```
x_ = solve(A, b)
x_
```

```
##                [,1]
##  [1,]  0.4142135516
##  [2,] -0.1715728967
##  [3,]  0.0710677582
##  [4,] -0.0294373802
##  [5,]  0.0121929977
##  [6,] -0.0050513848
##  [7,]  0.0020902282
```

```
##  [8,] -0.0008709284
##  [9,]  0.0003483714
## [10,] -0.0001741857
```

Pregunta 4

En general el metodo de Gauss-Seidel converge mas rápido que el de Jacobi al usar datos actualizados con cada iteracion, podemos comprobarloal realizar la iteraciones que para obtenre un error menor a 10^{-6}, el metodo de Jacobi requiere 292 iteraciones mientras que el metodo de Gauss-Sediel requiere 141, la diferencia tambien puede observarse, por consecuencia, en el error.

Pregunta 5

Para n=20

```
n = 20
A = diag(n)
for(i in 1:n-1) { A[i,i+1]<--1/2 }
for(i in 1:n) { A[i,i-1]<-1/2 }
b = matrix(rep(0,n))
b[1,1]<-1/2
D = diag(diag(A))
L = -tril(A, -1)
U = -triu(A, 1)
x0 = rep(0,n)
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Jacobi")
sol1 = itersolve(A, b, x0, nmax = 80 , method = "Jacobi")
```

Error para 80 iteraciones:0.0059479, Numero de iteraciones para error menos o igual a 10^{-6} : 987

Para n=40

```
n = 40
A = diag(n)
for(i in 1:n-1) { A[i,i+1]<--1/2 }
for(i in 1:n) { A[i,i-1]<-1/2 }
b = matrix(rep(0,n))
b[1,1]<-1/2
D = diag(diag(A))
L = -tril(A, -1)
U = -triu(A, 1)
x0 = rep(0,n)
sol = itersolve(A, b, x0,  tol = 1e-6, method = "Jacobi")
sol1 = itersolve(A, b, x0, nmax = 8 , method = "Jacobi")
```

Error para 40 iteraciones:0.0390625 Numero de iteraciones para error menos o igual a 10^{-6} : 1000