Método de Jacobi

Santiago Orjuela Convers

Abril 2020

El método de Jacobi consiste en encontrar la solución de un sistema de ecuaciones a partir de multiplicar varias veces una matriz T_j llamada la matriz de transición con un vector que cambia con cada iteración, ese vector se denota por $x^{(i)}$, donde i es la iteración número i. Las matrices y vectores que se utilizarán en este proceso son obtenidos de la siguiente forma:

$$\begin{aligned} Ax &= B \\ (D+L+U)x &= B \\ Dx &= B - Lx - Ux \\ x &= D^{-1}B - D^{-1}(L+U)x \\ x &= C - T_j x \end{aligned}$$

Donde $C = D^{-1}B$ y $T_j = D^{-1}(L+U)$. El proceso es iterativo y se hace de la siguiente manera:

$$x^{(i+1)} = C - T_j x^{(i)}$$

El vector $x^{(0)}$, donde $i = 0$, se toma como $x^{(0)} = (0, 0, \cdots, 0)$. Al hacer la multiplicación, se observa que el vector $x^{(1)} = C$, por lo que el vector de la siguiente iteración será $x^{(2)} = C - T_j x^{(1)} = C - T_j C$ y así sucesivamente.

Como se observa del método de Jacobi, dada una matriz M, es necesario conocer su matriz diagonal, su matriz triangular superior (U), su matriz triangular inferior (L) y la inversa de la matriz diagonal. Para ese motivo, creé tres funciones separadas que calculan cada una de las matrices recientemente mencionadas. Los códigos son los siguientes:

```
diagonal <- function(M){
  m <- length(M[1,])
  n <- length(M[,1])
  D <- matrix(0,n,m)
  for (i in 1:m){
    for(j in 1:n){
      if (i == j){
        D[i,j] <- M[i,j]
```

```
      }
    }
  }
  print(D)
}

superior <- function(M){
  m <- length(M[1,])
  n <- length(M[,1])
  U <- matrix(0,n,m)
  for (i in 1:m){
    for (j in 1:n){
      if (i < j){
        U[i,j] <- M[i,j]
      }
    }
  }
  print(U)
}

inferior <- function(M){
  m <- length(M[1,])
  n <- length(M[,1])
  L <- matrix(0,n,m)
  for (i in 1:m){
    for (j in 1:n){
      if (i > j){
        L[i,j] <- M[i,j]
      }
    }
  }
  print(L)
}

invdia <- function(M){
  m <- length(M[1,])
  n <- length(M[,1])
  D1 <- matrix(0,n,m)
  for (i in 1:m){
    for (j in 1:n){
      if (i == j & M[i,j] != 0){
        D1[i,j] <- 1/M[i,j]
      }
    }
  }
```

```
  print(D1)
}
```

Como ejemplo, aplicaré cada una de esas funciones a la siguiente matriz M:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
a <- c(1,2,3)
b <- c(4,5,6)
c <- c(7,8,9)
M <- rbind(a,b,c)
print(M)
```

```
##   [,1] [,2] [,3]
## a    1    2    3
## b    4    5    6
## c    7    8    9
```

```
diagonal(M)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    5    0
## [3,]    0    0    9
```

```
superior(M)
```

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

```
inferior(M)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    4    0    0
## [3,]    7    8    0
```

```
invdia(M)
```

```
##      [,1] [,2]      [,3]
## [1,]    1  0.0 0.0000000
## [2,]    0  0.2 0.0000000
## [3,]    0  0.0 0.1111111
```

Una vez definidas estas funciones, se crea otra función llamada norma, que es para medir la longitud de vectores. Esta función se crea para medir la diferencia entre los vectores de cada iteración consecutiva y saber que tan cerca está el uno del otro, pues si dos vectores están muy cerca, quiere decir que se está acercando a la solución.

```
norma <- function(x){
  i <- 1:length(x); N <- sqrt(sum(x[i]^2))
  return(N)
}
```

Por ejemplo, se creará un vector y se sacará su norma.

```
d <- matrix(a,ncol=1)
print(d)

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

norma(d)

## [1] 3.741657
```

Una vez creadas las funciones necesarias, se crea la función que utilizará el método de Jacobi para solucionar sistemas de ecuaciones, la función se llamará Jacobi y tendrá como entrada una matriz cuadrada A, un vector B que será el valor deseado, y un número entero positivo que será el número de iteraciones. La función está definida de la siguiente manera:

```
Jacobi <- function(A,B,k){
  m <- length(A[1,])
  L <- inferior(A)
  U <- superior(A)
  D <- diagonal(A)
  D1 <- invdia(D)
  Tj <- D1 %*% (L+U)
  lI <- diag(m)
  C <- D1 %*% B
  x <- integer(m)
  it <- matrix(0,k,m)
  a <- eigen(A)
  b <- eigen(Tj)
  tol <- 0
      for(i in 1:k){
```

```
        it[i,] <- C- Tj %*% x
        x <- C - Tj %*% x
        if (i == 1){
          tol <- 1
        } else{
          tol <- norma(it[i,]-it[i-1,])/norma(it[i,])
        }
        print(tol)
      }
  print(it)
  print(a)
  print(b)
}
```

Se mostrarán dos ejemplos de la función Jacobi. Para el primer ejemplo, considere el siguiente sistema:

$$\begin{bmatrix} 2 & 1 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

```
a1 <- c(2,1)
b1 <- c(1,5)
c1 <- c(3,6)
A <- rbind(a1,b1)
B <- matrix(c1,ncol=1)
print(A)
```

```
##    [,1] [,2]
## a1    2    1
## b1    1    5
```

```
print(B)
```

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

```
Jacobi(A,B,15)
```

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

```
## [2,]    0    0
##      [,1] [,2]
## [1,]    2    0
## [2,]    0    5
##      [,1] [,2]
## [1,]  0.5  0.0
## [2,]  0.0  0.2
## [1] 1
## [1] 0.5270463
## [1] 0.1312237
## [1] 0.0479133
## [1] 0.01353569
## [1] 0.004748165
## [1] 0.001357833
## [1] 0.0004743891
## [1] 0.000135826
## [1] 4.743464e-05
## [1] 1.358303e-05
## [1] 4.743421e-06
## [1] 1.358307e-06
## [1] 4.743417e-07
## [1] 1.358308e-07
##            [,1]      [,2]
##  [1,] 1.5000000 1.2000000
##  [2,] 0.9000000 0.9000000
##  [3,] 1.0500000 1.0200000
##  [4,] 0.9900000 0.9900000
##  [5,] 1.0050000 1.0020000
##  [6,] 0.9990000 0.9990000
##  [7,] 1.0005000 1.0002000
##  [8,] 0.9999000 0.9999000
##  [9,] 1.0000500 1.0000200
## [10,] 0.9999900 0.9999900
## [11,] 1.0000050 1.0000020
## [12,] 0.9999990 0.9999990
## [13,] 1.0000005 1.0000002
## [14,] 0.9999999 0.9999999
## [15,] 1.0000000 1.0000000
## eigen() decomposition
## $values
## [1] 5.302776 1.697224
##
## $vectors
##           [,1]       [,2]
## [1,] 0.2897841 -0.9570920
```

```
## [2,] 0.9570920  0.2897841
##
## eigen() decomposition
## $values
## [1]  0.3162278 -0.3162278
##
## $vectors
##           [,1]       [,2]
## [1,] 0.8451543 -0.8451543
## [2,] 0.5345225  0.5345225
```

Las primeras cuatro matrices que muestra el resultado son la inferior, la superior, la diagonal y la diagonal inversa. Luego, los 15 valores que se muestran son las distancias entre el vector resultante de una iteración y el de la iteración anterior, vemos que entre los últimos valores se tienen números muy cercanos al cero, por lo que el sistema converge a la solución del mismo. La matriz que se imprime que tiene 15 filas y dos columnas, contiene en cada fila cada uno de los vectores obtenidos en cada iteración, por ejemplo, el vector $x^{(1)} = (1{,}5, 1{,}2)$, $x^{(2)} = (0{,}9, 0{,}9)$, $x^{(3)} = (1{,}05, 1{,}02)$ y así sucesivamente. Por último, se tienen los autovalores y los autovectores de la matriz A y de la matriz T_j. El vector 15 es la solución del sistema, por lo tanto se tiene que:

$$\begin{bmatrix} 2 & 1 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

Para el segundo ejemplo considere el siguiente sistema de ecuaciones:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$$

```
a2 <- c(1,2,3)
b2 <- c(4,5,6)
c2 <- c(7,8,9)
d2 <- c(14,32,50)
V <- rbind(a2,b2,c2)
W <- matrix(d2,ncol=1)
print(V)
```

```
##    [,1] [,2] [,3]
## a2    1    2    3
## b2    4    5    6
## c2    7    8    9
```

```
print(W)
```

```
##      [,1]
## [1,]   14
## [2,]   32
## [3,]   50
```

```
Jacobi(V,W,30)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    4    0    0
## [3,]    7    8    0
##      [,1] [,2] [,3]
## [1,]    0    2    3
## [2,]    0    0    6
## [3,]    0    0    0
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    5    0
## [3,]    0    0    9
##      [,1] [,2]      [,3]
## [1,]    1  0.0 0.0000000
## [2,]    0  0.2 0.0000000
## [3,]    0  0.0 0.1111111
## [1] 1
## [1] 1.723673
## [1] 1.264106
## [1] 1.463332
## [1] 1.351498
## [1] 1.412356
## [1] 1.377591
## [1] 1.397865
## [1] 1.38591
## [1] 1.393066
## [1] 1.388767
## [1] 1.391368
## [1] 1.389792
## [1] 1.39075
## [1] 1.390168
## [1] 1.390522
## [1] 1.390306
## [1] 1.390438
## [1] 1.390358
## [1] 1.390407
## [1] 1.390377
## [1] 1.390395
```

```
## [1] 1.390384
## [1] 1.390391
## [1] 1.390387
## [1] 1.390389
## [1] 1.390388
## [1] 1.390389
## [1] 1.390388
## [1] 1.390388
##                [,1]          [,2]          [,3]
##  [1,]  1.400000e+01  6.400000e+00  5.555556e+00
##  [2,] -1.546667e+01 -1.146667e+01 -1.102222e+01
##  [3,]  7.000000e+01  3.200000e+01  2.777778e+01
##  [4,] -1.333333e+02 -8.293333e+01 -7.733333e+01
##  [5,]  4.118667e+02  2.058667e+02  1.829778e+02
##  [6,] -9.466667e+02 -5.426667e+02 -4.977778e+02
##  [7,]  2.592667e+03  1.361067e+03  1.224222e+03
##  [8,] -6.380800e+03 -3.536800e+03 -3.220800e+03
##  [9,]  1.675000e+04  8.976000e+03  8.112222e+03
## [10,] -4.227467e+04 -2.312827e+04 -2.100089e+04
## [11,]  1.092732e+05  5.902720e+04  5.344431e+04
## [12,] -2.783733e+05 -1.515453e+05 -1.374533e+05
## [13,]  7.154647e+05  3.876491e+05  3.512251e+05
## [14,] -1.828959e+06 -9.938355e+05 -9.010439e+05
## [15,]  4.690817e+06  2.544427e+06  2.305939e+06
## [16,] -1.200666e+07 -6.519774e+06 -5.910120e+06
## [17,]  3.076992e+07  1.669748e+07  1.513387e+07
## [18,] -7.879655e+07 -4.277657e+07 -3.877436e+07
## [19,]  2.018762e+08  1.095665e+08  9.930983e+07
## [20,] -5.170624e+08 -2.806728e+08 -2.544073e+08
## [21,]  1.324567e+09  7.189387e+08  6.516466e+08
## [22,] -3.392817e+09 -1.841630e+09 -1.669276e+09
## [23,]  8.691086e+09  4.717384e+09  4.275862e+09
## [24,] -2.226235e+10 -1.208390e+10 -1.095296e+10
## [25,]  5.702670e+10  3.095344e+10  2.805641e+10
## [26,] -1.460761e+11 -7.928905e+10 -7.186827e+10
## [27,]  3.741829e+11  2.031028e+11  1.840939e+11
## [28,] -9.584874e+11 -5.202590e+11 -4.715670e+11
## [29,]  2.455219e+12  1.332670e+12  1.207943e+12
## [30,] -6.289169e+12 -3.413706e+12 -3.094211e+12
## eigen() decomposition
## $values
## [1]  1.611684e+01 -1.116844e+00 -1.303678e-15
##
## $vectors
##            [,1]        [,2]       [,3]
```

```
## [1,] -0.2319707 -0.78583024  0.4082483
## [2,] -0.5253221 -0.08675134 -0.8164966
## [3,] -0.8186735  0.61232756  0.4082483
##
## eigen() decomposition
## $values
## [1]  2.561553 -1.561553 -1.000000
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.8066939 -0.9175796  0.4082483
## [2,] -0.4378663  0.2113105 -0.8164966
## [3,] -0.3968855  0.3367427  0.4082483
```

En este ejemplo, se puede observar que en cada iteración, el vector resultante cambia de signo y se nota en la distancia entre los vectores que a partir de la iteración número 24 la distancia es siempre igual. La razón por la que sucede esto es porque el método no converge en este ejemplo. Se tiene un resultado y es que el sistema de ecuaciones va a tener solución si y sólo si el radio espectral de la matriz $T_j < 1$. En la salida de la función Jacobi, se tiene que los autovalores de la matriz T_j son: 2,561553, -1,561553, -1; como el radio espectral de una matriz está definido como.

$$\rho(A) = Sup\{|\lambda_1|, |\lambda_2|, \cdots, |\lambda_n|\}$$

Entonces $\rho(T_j) = 2{,}561553$, por lo que el radio espectral es mayor que 1 y entonces este sistema de ecuaciones no tendrá solución con el método de Jacobi. Sin embargo, no tener solución con el método de Jacobi no implica que el sistema no tenga solución, en este caso la solución al sistema existe.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$$