Implementación en Fortran95 de los métodos iterativos básicos para la solución de sistemas de ecuaciones lineales

Pablo Santamaría

v0.1 (Mayo 2014)

Sea

$$A\,\mathbf{x} = \mathbf{b},$$

un sistema de n ecuaciones lineales con n incógnitas, donde A es una matriz cuadrada de orden n de elementos reales y no singular (por lo cual el sistema admite una solución y ésta es única). Un método iterativo para resolver dicho sistema comienza con una aproximación inicial $\mathbf{x}^{(0)}$ a la solución $\mathbf{x}$, y genera una sucesión de vectores $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ que se espera converja a $\mathbf{x}$. Los dos métodos iterativos básicos son el *método de Jacobi* y el *método de Gauss–Seidel* cuyas implementaciones algorítmicas se describen a continuación.

Método de Jacobi

Dada A de $n \times n$ con elementos diagonales no nulos.
Escoger $\mathbf{x}^{(0)}$ de orden n, usualmente $\mathbf{x}^{(0)} = \mathbf{0}$.
Para $k = 1, 2, \ldots$

Tomar $x_i^{(k)} = \dfrac{b_i - \sum_{\substack{j=1 \\ (j \neq i)}}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}$, para $i = 1, \ldots, n$.

Método de Gauss–Seidel

Dada A de $n \times n$ con elementos diagonales no nulos.
Escoger $\mathbf{x}^{(0)}$ de orden n, usualmente $\mathbf{x}^{(0)} = \mathbf{0}$.
Para $k = 1, 2, \ldots$

Tomar $x_i^{(k)} = \dfrac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}$, para $i = 1, \ldots, n$.

Como en todo método iterativo, debe establecerse un criterio de paro para las iteraciones, el cual, en este contexto, puede ser

$$\frac{\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|}{\|\mathbf{x}^{(k)}\|} < \epsilon,$$

respecto de una norma y cierta tolerancia ϵ prefijada.

Nótese que en el método de Jacobi cada componente de una nueva iteración es generada a partir de las componentes de la iteración anterior en forma independiente y simultáneamente, de aquí que el método se conozca también como *método de desplazamientos simultáneos*. En el método de Gauss-Seidel, por otra parte, las componentes de una nueva iteración son sucesivamente actualizadas tan pronto una es conocida, y por ello al método se lo conoce también como *método de desplazamientos sucesivos*.

Desde el punto de vista de la implementación computacional, lo anterior implica que el método de Jacobi requiere del almacenamiento de dos vectores, el de la iteración actual y el de la anterior, en tanto que el método de Gauss-Seidel permite considerar un solo vector cuyas componentes se actualizan al momento de la iteración. Por otra parte, la naturaleza simultánea de las correcciones en el método de Jacobi, hacen del

mismo un método inherentemente *paralelizable*, mientras que, en principio, el método de Gauss-Seidel es *secuencial*.

La naturaleza paralelizable del método de Jacobi queda manifiesta en Fortran 95 donde podemos manipular los arreglos como un todo. En efecto, reescribiendo las ecuaciones de la iteración k-ésima del método como sigue

$$\begin{aligned} x_i^{(k)} &= \frac{b_i - \sum_{\substack{j=1 \\ (j\neq i)}}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}} \\ &= \frac{b_i + a_{ii} x_i^{(k-1)} - \sum_{j=1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}} \\ &= x_i^{(k-1)} + \frac{b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}}, \end{aligned}$$

notemos que $\sum_{j=1}^{n} a_{ij} x_j^{(k-1)}$ es la componente i-ésima del producto matricial $A\,x^{(k-1)}$, y por lo tanto la iteración procede como

$$\begin{aligned} x_i^{(k)} &= x_i^{(k-1)} + \frac{b_i - (A\,\mathbf{x}^{(k-1)})_i}{a_{ii}} \\ &= x_i^{(k-1)} + \frac{(\mathbf{b} - A\,\mathbf{x}^{(k-1)})_i}{a_{ii}}, \end{aligned}$$

para $i = 1, \ldots, n$. Si introducimos el vector $\mathbf{d} = (a_{11}, a_{22}, \ldots, a_{nn})^t$, podemos escribir, finalmente, la iteración k-ésima del método de Jacobi como

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \frac{\mathbf{b} - A\,\mathbf{x}^{(k-1)}}{\mathbf{d}}.$$

donde la "división" de arreglos se interpreta componente a componente.

Por otra parte, en las ecuaciones de la iteración k-ésima del método de Gauss-Seidel

$$x_i^{(k)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)}}{a_{ii}},$$

la primer sumatoria puede pensarse como el *producto escalar* del vector de $i-1$ elementos $(a_{i1}, a_{i2}, \ldots, a_{ii-1})^t$ conformado por las primeras $i-1$ columnas de la fila i-ésima, por el vector de $i-1$ elementos $(x_1^{(k)}, x_2^{(k)}, \ldots, x_{i-1}^{(k)})^t$ conformado por los valores obtenidos hasta el momento en la iteración. Análogamente, la segunda sumatoria puede escribirse como el producto escalar del vector $(a_{ii+1}, a_{ii+2}, \ldots, a_{in})^t$ por el vector conformado por valores obtenidos en la iteración anterior $(x_{i+1}^{(k-1)}, x_{i+2}^{(k-1)}, \ldots, x_n^{(k-1)})^t$.

El siguiente módulo Fortran implementa sendos métodos según lo expuesto, considerando la norma máxima para el criterio de paro de las iteraciones.

Código 1. Implementación de los métodos iterativos de Jacobi y Gauss-Seidel

```
MODULE iterative_methods

  USE, INTRINSIC :: iso_fortran_env, ONLY: WP => REAL64
  IMPLICIT NONE

CONTAINS

  SUBROUTINE iterative_jacobi(a,b,x,tol,iter,clave)
    ! ----------------------------------------------------------------
    ! Argumentos de la subrutina
    ! ----------------------------------------------------------------
    REAL(WP), INTENT(IN)    :: a(:,:) ! Matriz del sistema
    REAL(WP), INTENT(IN)    :: b(:)   ! Término independiente
    REAL(WP), INTENT(INOUT) :: x(:)   ! Aproximación inicial/solución
    REAL(WP), INTENT(IN)    :: tol    ! Tolerancia para el error
    INTEGER,  INTENT(INOUT) :: iter   ! Max iteraciones/iter realizadas
    INTEGER,  INTENT(OUT)   :: clave  ! Clave de éxito/error
                                      ! clave = 0, OK.
                                      ! clave < 0, error en asig. memoria
                                      ! clave > 0, max iter. alcanzado
    ! ----------------------------------------------------------------
    ! Variables locales
    ! ----------------------------------------------------------------
    REAL(WP), ALLOCATABLE :: diag(:) ! Arreglo para guardar la
                                     ! diagonal de la matriz del sistema
    REAL(WP), ALLOCATABLE :: x0(:)   ! Arreglo para guardar la
                                     ! iteración anterior
    INTEGER  :: k                    ! Contador de iteraciones
    INTEGER  :: i                    ! Indice
    ! ----------------------------------------------------------------
    ! Procedimiento
    ! ----------------------------------------------------------------
    ALLOCATE(diag(SIZE(a,1)),x0(SIZE(a,1)), STAT=clave)
    IF (clave /=0 ) THEN
        clave = -1
        RETURN
    ENDIF

    clave = 1
    diag  = [ (a(i,i), i=1,SIZE(a,1)) ]
    DO k=1,iter
       x0 = x
       x  = x0 + (b - MATMUL(a,x0))/diag
       IF ( MAXVAL(ABS(x-x0)) <= MAXVAL(ABS(x))*tol ) THEN
          clave = 0
          iter  = k
          EXIT
       ENDIF
    ENDDO

    DEALLOCATE(diag,x0)

  END SUBROUTINE iterative_jacobi

  SUBROUTINE iterative_gauss_seidel(a,b,x,tol,iter,clave)
    ! ----------------------------------------------------------------
    ! Argumentos de la subrutina
    ! ----------------------------------------------------------------
    REAL(WP), INTENT(IN)    :: a(:,:) ! Matriz del sistema
    REAL(WP), INTENT(IN)    :: b(:)   ! Término independiente
    REAL(WP), INTENT(INOUT) :: x(:)   ! Aproximación inicial/solución
    REAL(WP), INTENT(IN)    :: tol    ! Tolerancia para el error
    INTEGER,  INTENT(INOUT) :: iter   ! Max iteraciones/iter realizadas
```

```
      INTEGER,  INTENT(OUT)   :: clave  ! Clave de éxito/error
                                         ! clave  = 0, OK.
                                         ! clave /= 0, max iter. alcanzado
      ! ----------------------------------------------------------------
      ! Variables locales
      ! ----------------------------------------------------------------
      INTEGER  :: n               ! Dimensión del problema
      INTEGER  :: k               ! Contador de iteraciones
      INTEGER  :: i               ! Indice
      REAL(WP) :: xi              ! Componente del vector iterado
      REAL(WP) :: xnorma          ! Norma del vector iterado
      REAL(WP) :: difnorma        ! Norma de la diferencia entre dos
                                  ! iteraciones del vector
      ! ----------------------------------------------------------------
      ! Procedimiento
      ! ----------------------------------------------------------------
      clave = 1
      n     = SIZE(a,1)
      DO k=1,iter
         xnorma   = 0.0_WP
         difnorma = 0.0_WP
         DO i=1,n
            xi       = (b(i) - DOT_PRODUCT(a(i,1:i-1),x(1:i-1)) &
          &              - DOT_PRODUCT(a(i,i+1:n),x(i+1:n)))/a(i,i)
            xnorma   = MAX(xnorma,ABS(xi))
            difnorma = MAX(difnorma,ABS(xi-x(i)))
            x(i)     = xi
         END DO
         IF (difnorma <= xnorma*tol) THEN
            iter  = k
            clave = 0
            EXIT
         ENDIF
      ENDDO

    END SUBROUTINE iterative_gauss_seidel

  END MODULE iterative_methods
```