Programación: el método de Gauss–Seidel en forma matricial

Objetivos. Programar una función que resuelva sistemas de ecuaciones lineales con el método iterativo de Gauss–Seidel en la forma matricial.

Requisitos. Operaciones con vectores (operaciones lineales y el producto punto), matrices triangulares, solución de sistemas con matrices triangulares, ciclo `while`, la idea del método de Gauss–Seidel.

1. Solución de sistemas triangulares inferiores. Escribir una función que resuelva sistemas de ecuaciones lineales de la forma $Tx = b$, donde T es una matriz cuadrada triangular inferior con entradas diagonales no nulas.

> Entrada: una matriz T, un vector b.
>
> Propiedades de la entrada: suponer (y no verificar) que T es una matriz cuadrada triangular inferior con entradas diagonales no nulas y que la longitud de b coincide con el orden de T.
>
> Salida: un vector x tal que $Tx = b$.

Idea: aplicar la sustitución hacia adelante. Se recomienda usar operaciones de nivel 1 (el producto punto de algunos fragmentos de columnas o las operaciones lineales con algunos fragmentos de columnas).

```
function x = solvelt(T, b),
   n = length(b); x = zeros(n, 1);
   ...
end
```

2. Descomposición de una matriz en la suma de una matriz triangular inferior y una matriz estrictamente triangular superior. Cada matriz cuadrada A se puede escribir de manera única como $A = T + U$, donde T es triangular inferior y U es estrictamente triangular superior. Por ejemplo, para $n = 3$,

$$\underbrace{\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} A_{1,1} & 0 & 0 \\ A_{2,1} & A_{2,2} & 0 \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}}_{T} + \underbrace{\begin{bmatrix} 0 & A_{1,2} & A_{1,3} \\ 0 & 0 & A_{2,3} \\ 0 & 0 & 0 \end{bmatrix}}_{U}.$$

En el lenguaje de MATLAB las matrices T y U se pueden obtener con los siguientes comandos:

```
T = tril(A);
U = triu(A, 1);
```

Método de Gauss–Seidel en forma matricial 1, con dos matrices triangulares

3. Usando la notación anterior T y U, escribimos el sistema original $Ax = b$ en la forma

$$Tx = b - Ux$$

y luego en la forma

$$x = T^{-1}(b - Ux).$$

Empezando con algún vector $x^{(0)}$, construimos la sucesión de vectores $(x^{(s)})_{s=0}^{\infty}$ mediante la fórmula recursiva

$$x^{(s+1)} = T^{-1}(b - Ux^{(s)}). \tag{1}$$

En el programa se recomienda trabajar solamente con el *vector actual* y el *vector previo*, esto es, en cada paso del método de Gauss–Seidel guardar una copia del vector actual `x` en la variable `xprev`, luego calcular el vector actual por la fórmula (1):

```
xprev = x;
x = solvelt(T, b - U * xprev);
```

4. Esquema del algoritmo.

```
function [x, s] = GaussSeidelMatrixForm1(A, b, tol, smax),
   T = ???; U = ???;
   n = length(b); x = zeros(n, 1);
   er = tol + 1; s = ???;
   while (er ??? tol) && (s ??? smax),
      xprev = x;
      ...
      d = norm(x - xprev); s = s + 1;
   end
end
```

Método de Gauss–Seidel en forma matricial 2, con una matriz triangular y con el vector residuo

5. Método de Gauss escrito en términos del vector residuo. Como antes, denotamos por T a la parte triangular inferior de la matriz A, incluso la diagonal. En cada paso s denotamos por $r^{(s)}$ al vector residuo:

$$r^{(s)} := b - Ax^{(s)}.$$

Notamos que el sistema original de ecuaciones lineales $Ax = b$ se puede escribir en las siguientes formas equivalentes:

$$\mathbf{0}_n = b - Ax \qquad \Longleftrightarrow \qquad \mathbf{0}_n = T^{-1}(b - Ax) \qquad \Longleftrightarrow \qquad x = x + T^{-1}(b - Ax).$$

La fórmula recursiva sería

$$x^{(s+1)} = x^{(s)} + T^{-1}r^{(s)}.$$

Por supuesto, en vez de calcular la matriz inversa T^{-1}, hay que calcular $T^{-1}r^{(s)}$ como el vector solución de un sistema de ecuaciones lineales con la matriz triangular inferior T.

6. Cambio del residuo al cambiar la aproximación de la solución. Si

$$x^{(s+1)} = x^{(s)} + p^{(s)},$$

entonces

$$r^{(s+1)} = b - Ax^{(s+1)} = b - A(x^{(s)} + p^{(s)}) = b - Ax^{(s)} - Ap^{(s)} = r^{(s)} - Ap^{(s)}.$$

Denotaremos el vector $Ap^{(s)}$ por $q^{(s)}$.

7. Esquema del algoritmo. Ahora es cómodo usar el vector $r^{(s)}$ en la condición del ciclo `while`.

```
function [x, s] = GaussSeidelMatrixForm2(A, b, tol, smax),
   T = ???;
   n = length(b); x = zeros(n, 1); r = b;
   s = ???;
   while (s ??? smax) && (norm(r) ??? tol),
      p = solvelt(???, r);
      q = A * p;
      x = ???;
      r = r - q;
      s = ???;
   end
end
```

Pruebas

8. Prueba con matrices pequeñas. Para garantizar la convergencia generamos una matriz estrictamente diagonal dominante. En vez de generar una matriz pseudoaleatoria puede construir el ejemplo a mano.

```
function [] = testGaussSeidel1(),
   n = 4;
   A = 2 * rand(n) - ones(n) + n * eye(n);
   b = 2 * rand(n, 1) - ones(n, 1);
   [x, s] = GaussSeidelMatrixForm1(A, b, 1.0E-5, 100);
   display(s); display(norm(A * x - b));
end
```

También hacer pruebas del otro algoritmo, `GaussSeidelMatrixForm2`.

9. Pruebas con matrices grandes. Elegir n de tal manera que el tiempo de ejecución sea de 1 a 100 segundos, medir el tiempo de ejecución con el comando comando `cputime`.

```
function [] = testGaussSeidel2(),
   ...
end
```