Programación: forma matricial del método de Jacobi para resolver sistemas de ecuaciones lineales

Objetivos. Programar una función que resuelva sistemas de ecuaciones lineales con el método iterativo de Jacobi 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 Jacobi.

1. Multiplicación y división de vectores componente a componente. Dados dos vectores $a, b \in \mathbb{R}^n$, denotemos por $a \odot b$ al vector de longitud n cuya j-ésima componente es

$$(a \odot b)_j = a_j b_j.$$

Si además $b_j \neq 0$ para cada $j \in \{1, \ldots, n\}$, entonces denotemos por $a \oslash b$ al vector de longitud n cuya j-ésima componente es

$$(a \oslash b)_j = \frac{a_j}{b_j}.$$

En el lenguaje de MATLAB (y en GNU Octave, Scilab, FreeMat) la multiplicación y división por componentes se denotan por `.*` y `./`:

```
a = [6; -5; 1]
b = [-2; 2; 3]
a .* b
a ./ b
```

2. Multiplicación de una matriz diagonal por un vector. Se $d = [d_j]_{j=1}^n$ un vector. Se denota por $\operatorname{diag}(d)$ la matriz diagonal con entradas diagonales $d_1, \ldots, d_n$. Por ejemplo, si $n = 3$, entonces

$$\operatorname{diag}(d) = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix}.$$

Notamos que

$$\operatorname{diag}(d)v = \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} d_1 v_1 \\ d_2 v_2 \\ d_3 v_3 \end{bmatrix} = d \odot v.$$

En general, si $d, v \in \mathbb{R}^n$, entonces

$$\operatorname{diag}(d)v = d \odot v.$$

3. Multiplicación de una matriz por un vector, prueba numérica.

```
d = [5; -1; 2; 3]
v = [-3; 2; 3; 5]
diag(d)
diag(d) * v
d .* v
```

4. Solución de sistemas de ecuaciones lineales con matrices diagonales. Si $d, b \in \mathbb{R}^n$ y $d_j \neq 0$ para cada $j \in \{1, \ldots, n\}$, entonces el sistema de ecuaciones lineales

$$\operatorname{diag}(d)x = b$$

tiene una única solución la cual se puede calcular como

$$x = b \oslash d.$$

Prueba numérica:

```
d = [-2; 3; 2];
b = [5; -1; 0];
x = b ./ d;
norm(diag(d) * x - b)
```

5. Sacar el vector de las entradas diagonales de una matriz. Dada una matriz A, denotemos por d al vector de sus entradas diagonales y consideremos la matriz diagonal D con entradas diagonales $A_{1,1}, \ldots, A_{n,n}$:

$$d = \left[A_{j,j}\right]_{j=1}^{n}, \qquad D = \operatorname{diag}(d).$$

Por ejemplo, para $n = 3$,

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}, \qquad d = \begin{bmatrix} A_{1,1} \\ A_{2,2} \\ A_{3,3} \end{bmatrix}, \qquad D = \begin{bmatrix} A_{1,1} & 0 & 0 \\ 0 & A_{2,2} & 0 \\ 0 & 0 & A_{3,3} \end{bmatrix}.$$

En el lenguaje de MATLAB el vector d se puede obtener con el comando `diag`:

```
A = rand(3)
d = diag(A)
D = diag(d)
```

Idea del método de Jacobi en forma matricial

6. Sea $A \in \mathcal{M}_n(\mathbb{R})$ tal que $A_{j,j} \neq 0$ para cada $j \in \{1, \ldots, n\}$. Denotemos por d al vector de las entradas diagonales de A y por D a la matriz diagonal generada por el vector d:

$$d = \big[A_{j,j}\big]_{j=1}^{n}, \qquad D = \operatorname{diag}(d).$$

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

$$\begin{aligned} & Ax = b \\ \iff \quad & b - Ax = 0 \\ \iff \quad & Dx = Dx + (b - Ax) \\ \iff \quad & x = x + D^{-1}(b - Ax) \\ \iff \quad & x = x + (b - Ax) \oslash d. \end{aligned}$$

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

$$x^{(s+1)} = x^{(s)} + (b - Ax^{(s)}) \oslash d. \tag{1}$$

7. Residuo y su modificación. Es cómodo guardar el residuo $b - Ax$ en una variable r. Notamos que si el vector x se modifica mediante la fórmula

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

entonces el vector residuo correspondiente se cambia de la siguiente manera:

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

En el algoritmo podemos usar los siguientes comandos:

```
p = r ./ d;
q = A * p;
x = x + p;
r = r - q;
```

8. Condición de terminación. El ciclo `while` debe terminarse cuando el número s de las iteraciones realizadas es mayor o igual al número máximo de iteraciones permitidas $s_{\max}$ o cuando la norma del residuo es menor que la "tolerancia" dada. En otras palabras, la condición de terminación es

$$(s \geq s_{\max}) \qquad \vee \qquad (\|r\| < \mathrm{tol}).$$

Escriba la condición de continuación.

9. Esquema del algoritmo.

```
function [x, s] = JacobiSolveMatrixForm(A, b, tol, smax),
   d = ???;
   n = length(b); x = zeros(n, 1); r = b;
   s = ???;
   while (s ??? smax) && (norm(r) ??? tol),
      p = ???;
      q = ???;
      x = ???;
      r = ???;
      s = s + 1;
   end
end
```

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

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

11. 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 `cputime`.

```
function largeTestJacobiSolve(),
   ...
end
```