Tareas de matrices especiales

Objetivos. Estudiar una clase especial de matrices. Para matrices de esta clase realizar un algoritmo rápido de multiplicación por vectores. Aplicar este algoritmo para resolver sistemas lineales con métodos iterativos. Comparar la rapidez de varios métodos.

Requisitos. Experiencia básica de programación, con funciones y arreglos, en algún lenguaje de programación con tipización dinámica y orientación a matrices (Matlab o GNU Octave, Python + numpy o Sage, R, Wolfram Mathematica, etc.) o mucha experiencia de programación en lenguajes compilables sin orientación a matrices (C, C++, Fortran, Haskell, Java, C#, etc.). Conocimientos amplios de álgebra lineal y de métodos numéricos.

Índice

1. Datos iniciales que determinan la matriz. Trabajamos solamente con matrices reales cuadradas y denotamos su orden por n. Cada matriz de la clase elegida se determina por ciertos *datos iniciales*. Por ejemplo, una matriz tridiagonal

$$A = \begin{bmatrix} a_1 & b_1 & 0 & 0 & 0 \\ c_1 & a_2 & b_2 & 0 & 0 \\ 0 & c_2 & a_3 & b_3 & 0 \\ 0 & 0 & c_3 & a_4 & b_4 \\ 0 & 0 & 0 & c_4 & a_5 \end{bmatrix} \tag{1}$$

se determina por tres vectores a, b, c:

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}, \qquad c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}.$$

Otro ejemplo: la matriz circulante

$$\begin{bmatrix} a_1 & a_5 & a_4 & a_3 & a_2 \\ a_2 & a_1 & a_5 & a_4 & a_3 \\ a_3 & a_2 & a_1 & a_5 & a_4 \\ a_4 & a_3 & a_2 & a_1 & a_5 \\ a_5 & a_4 & a_3 & a_2 & a_1 \end{bmatrix} \tag{2}$$

se determina por su primera columna.

2. El tamaño total de los datos iniciales que determinan a la matriz. Denotamos por $D(n)$ al tamaño total de los datos iniciales que determinan la matriz de orden n. Para matrices tridiagonales el número $D(n)$ es la suma de las longitudes de los vectores a, b, c:

$$D(n) = n + (n-1) + (n-1) = 3n - 2.$$

Para matrices circulantes $D(n) = n$.

3. El tamaño total de los datos iniciales debe ser esencialmente menor que el número total de las entradas de la matriz. Formalmente, se debe cumplir la siguiente relación límite:

$$\lim_{n\to\infty} \frac{D(n)}{n^2} = 0. \tag{3}$$

Típicamente $D(n)$ es un polinomio de grado 1 de la variable n, o una constante.

4. La clase elegida de matrices especiales debe incluir algunas matrices simétricas positivas definidas. Se recomienda trabajar solamente con matrices reales cuadradas. Para cualquier orden n la clase elegida de matrices especiales debe incluir algunas matrices simétricas y positivas definidas. Por ejemplo, la matriz tridiagonal de la forma (1) es simétrica si $b = c$. La matriz circulante (2) es simétrica si $a_5 = a_1$ y $a_4 = a_2$. Una matriz real cuadrada A es simétrica y positiva definida si existe una matriz B de rango n tal que $A = B^\top B$. Para matrices de ciertos tipos esta propiedad se demuestra fácilmente. En otras situaciones esta propiedad se puede verificar con experimentos numéricos: la matriz debe ser simétrica, y sus valores propios deben ser estrictamente positivos.

5. Elegir una clase de matrices especiales. Elegir una de las clases de matrices propuestas por el profesor o encontrar otra clase. Las matrices deben cumplir con las condiciones 3 y 4.

6. Buscar aplicaciones (opcional). Si las matrices de la clase elegida tienen alguna aplicación interesante, es una ventaja adicional. Por ejemplo, las matrices tridiagonales surgen en el método de diferencias finitas que se usa para resolver numéricamente problemas de frontera para ecuaciones diferenciales ordinarias de segundo orden.

7. Organización del trabajo y presentación de resultados. Las tareas de matrices especiales se hacen por equipos fuera de clases. La interacción y comunicación dentro del equipo es la parte más importante y más difícil de la tarea. Las dudas posibles se resuelven con el profesor. Los productos de las tareas (textos en el formato LaTeX y códigos de programas) se envían al correo del profesor. Además los estudiantes hacen exposiciones breves (de 5 a 10 minutos), en las cuales explican sus avances a todos los compañeros.

Primera tarea de matrices especiales: construcción de la matriz en el formato completo

8. Describir la clase elegida de matrices y los datos iniciales. Escribir un archivo en el lenguaje TEX/LATEX, de una o dos páginas, en el cual introducir de manera clara la clase elegida de matrices especiales y explicar cómo se obtiene la matriz a partir de los datos iniciales. Se recomienda hacerlo con un ejemplo de tamaño pequeño ($5 \le n \le 16$).

9. Fórmula general para las entradas de la matriz (opcional). Algunas clases de matrices especiales admiten una fórmula general simple para la (j, k)-ésima entrada. Por ejemplo, si A es la matriz tridiagonal generada por las diagonales a, b, c, como en el ejemplo (1), entonces

$$A_{j,k} = \begin{cases} a_j, & \text{si } k = j, \\ b_j, & \text{si } k = j + 1, \\ c_{j-1}, & \text{si } k = j - 1. \end{cases}$$

10. Función que construye la matriz especial de la clase elegida en el formato completo a partir de los datos iniciales. Aquí está una función en el lenguaje Matlab que construye la matriz tridiagonal a partir de los vectores a, b, c, suponiendo que $n \ge 2$:

```
function [A] = tridiagonal_full(a, b, c),
   n = length(a);
   A = zeros(n);
   for j = 1 : n,
      A(j, j) = a(j);
   end
   for j = 1 : n - 1,
      A(j, j + 1) = b(j);
      A(j + 1, j) = c(j);
   end
end
```

11. Construir la matriz especial de la clase elegida usando comandos especiales del lenguaje elegido. En algunos lenguajes de programación y para algunas clases de matrices especiales, el problema anterior tiene soluciones más eficientes. La idea es evitar ciclos y usar comandos especiales:

```
function [A] = tridiagonal_full_fast(a, b, c),
   A = diag(a) + diag(b, 1) + diag(c, -1);
end
```

12. Construir la matriz especial de la clase elegida en el formato para matrices dispersas (opcional). En algunos lenguajes de programación hay un formato especial para guardar *matrices dispersas* (también se llaman matrices *esparcidas*, *ralas*, *huecas*). Algunas clases de matrices especiales se pueden guardar en este formato:

```
function [A] = tridiagonal_sparse(a, b, c),
   D = [b', 0; a; 0, c];
   A = spdiags(D, [-1, 0, 1]);
end
```

13. Escribir una función que haga una prueba con datos pequeños de la función que construye la matriz especial.

```
function [] = small_test_tridiagonal_full(),
   a = [1; 2; 3; 4; 5];
   b = [11; 12; 13; 14];
   c = [21; 22; 23; 24];
   A = tridiagonal_full(a, b, c);
   display(A);
   A1 = tridiagonal_full_fast(a, b, c);
   display(A1);
   A2 = tridiagonal_sparse(a, b, c);
   display(A2);
   display(full(A2));
end
```

14. Construcción de datos iniciales pseudoaleatorios (opcional). Para algunas clases de matrices especiales (por ejemplo, matrices tridiagonales) los datos iniciales se generan fácilmente. Otras clases de matrices especiales necesitan más trabajo en esta parte. Por ejemplo, si la matriz se determina por cierta gráfica, entonces hay que elegir el formato en la cual se guarda la gráfica (por ejemplo, con una lista de aristas o con listas de vecinos) y escribir una función que genere estos datos iniciales de manera pseudoaleatoria. El argumento de esta función debe ser n o algún otro parámetro que determina el tamaño de la matriz; también pueden haber otros parámetros adicionales como el grado máximo de la gráfica. Este trabajo se puede hacer después de entregar la primera tarea.

15. Sugerencia acerca del archivo LaTeX con la explicación de la clase especial de matrices. La idea es explicar el objeto de estudio de manera muy clara, simple y concisa. Este texto no se escribe para un experto en el área, sino para un estudiante de primeros semestres de licenciatura. Un ejemplo de tamaño adecuado casi siempre es mejor que fórmulas generales.

Segunda tarea de matrices especiales: multiplicación rápida por un vector

16. Objetivos de la segunda tarea de matrices especiales. Programar un algoritmo rápido para multiplicar las matrices especiales de la clase elegida por vectores. Este algoritmo rápido debe usar los datos iniciales que determinan a la matriz. Dentro del algoritmo no se debe construir la matriz en la forma completa. Comparar la rapidez de la función programada y de la función estándar que multiplica matrices (en la forma completa) por vectores.

17. Representar el producto de la matriz por el vector de tal manera que ayude a programar el algoritmo eficiente. Puede ser útil considerar la situación para un n pequeño. Veremos la situación con matrices tridiagonales. Si $x \in \mathbb{R}^5$ es un vector general, entonces para la matriz tridiagonal del ejemplo (1) el producto Ax se puede escribir de la siguiente manera:

$$Ax = \begin{bmatrix} a_1x_1 + b_1x_2 \\ c_1x_1 + a_2x_2 + b_2x_3 \\ c_1x_2 + a_3x_3 + b_3x_4 \\ c_1x_3 + a_4x_4 + b_4x_5 \\ c_1x_4 + a_5x_5 \end{bmatrix} = \begin{bmatrix} a_1x_1 \\ a_2x_2 \\ a_3x_3 \\ a_4x_4 \\ a_5x_5 \end{bmatrix} + \begin{bmatrix} b_1x_2 \\ b_2x_3 \\ b_3x_4 \\ b_4x_5 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_1x_1 \\ c_2x_2 \\ c_3x_3 \\ c_4x_4 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \odot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ 0 \end{bmatrix} \odot \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ x_5 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} \odot \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix},$$

donde $\odot$ es la multiplicación de vectores componente a componente.

18. Algunas clases de matrices necesitan más teoría. Por ejemplo, para trabajar de manera eficiente con matrices circulantes es importante saber el teorema de convolución:

$$F_n(a * b) = F_n(a) \odot F_n(b). \tag{4}$$

Aquí $a*b$ es la convolución discreta cíclica de dos vectores $a, b \in \mathbb{C}^n$, $\odot$ es la multiplicación de vectores por componentes y F_n es la transformada discreta cíclica de Fourier. Además se sabe que el producto de una matriz circulante por un vector se expresa a través de la convolución discreta cíclica de dos vectores:

$$C_n(a)x = a * x. \tag{5}$$

De las fórmulas (4) y (5) se sigue que

$$C_n(a)x = a * x = F_n^{-1}F_n(a * x) = F_n^{-1}(F_n(a) \odot F_n(x)).$$

La última expresión se puede calcular en el lenguaje Matlab como

```
ifft(fft(a) .* fft(x));
```

19. Escribir una función que multiplica la matriz especial por un vector, trabajando con los datos iniciales, sin construir la matriz completa.

```
function [y] = tridiagonal_mul_vector(a, b, c, x),
   bext = [b; 0];
   cext = [0; c];
   xmost = x(1 : end - 1);
   xrshift = [0; xmost];
   xrest = x(2 : end);
   xlshift = [xrest; 0];
   y = a .* x + bext .* xlshift + cext .* xrshift;
end
```

20. Pruebas con datos pequeños.

```
function [] = small_test_tridiagonal_mul_vector(),
   a = [-2; 1; 4; 2];
   b = [1; -2; -1];
   c = [3; 1; 5];
   x = [-4; -1; 3; -2];
   Afull = tridiagonal_full_fast(a, b, c);
   Asparse = tridiagonal_sparse(a, b, c);
   p1 = Afull * x;
   p2 = Asparse * x;
   p3 = tridiagonal_mul_vector(a, b, c, x);
   display(p1);
   display(p2);
   display(p3);
end
```

21. Pruebas con datos grandes.

```
function [] = large_test_tridiagonal_mul_vector(n),
   a = rand(n, 1); b = rand(n - 1, 1); c = rand(n - 1, 1);
   x = rand(n, 1);
   Afull = tridiagonal_full_fast(a, b, c);
   Asparse = tridiagonal_sparse(a, b, c);
   t1 = cputime(); p1 = Afull * x; t1 = cputime() - t1;
   t2 = cputime();
   p2 = tridiagonal_mul_vector(a, b, c, x);
   t2 = cputime() - t3;
   t3 = cputime(); p3 = Asparse * x; t3 = cputime() - t3;
   display([t1, t2, t3]);
   display([norm(p1 - p2), norm(p1 - p3)]);
end
```

22. Tabla con resultados de la prueba (el tiempo de ejecución). Los renglones de la tabla corresponden a las tres maneras de calcular el producto:

I. Con el comando `Afull * x`.

II. Con la función `tridiagonal_mul_vector`.

III. Con el comando `Asparse * x`.

	$n = 256$	$n = 512$	$n = 1024$	$n = 2048$	$n = 4096$
I.	???	???	???	???	???
II.	???	???	???	???	???
III.	???	???	???	???	???

23. Criterio de buena ejecución. Para n suficientemente grande, II debe ser más rápido que I. Para $n \leq 1000$, las normas de las diferencias entre los productos calculados por varios métodos deben ser menores a 10^{-8}.

Tercera tarea de matrices especiales: métodos iterativos para resolver sistemas lineales

24. Escribir funciones que aplican métodos iterativos de Jacobi, de gradiente y de gradiente conjugado a sistemas lineales con matrices especiales de la clase elegida, usando la función del Ejercicio 19. Por ejemplo, en el lenguaje Matlab el método de Jacobi para sistemas de ecuaciones lineales de la forma $Ax = y$ con matrices tridiagonales se puede realizar con la siguiente función:

```
function [x, s] = tridiagonal_Jacobi_solver(a, b, c, y, tol, smax),
   n = length(a); x = zeros(n, 1); r = b; s = 0;
   while (s < smax) && (norm(r) > tol),
      x = x + r ./ a;
      r = b - tridiagonal_mul_vector(a, b, c, x);
      s = s + 1;
   end
end
```

25. Solución de sistemas por medio de métodos directos, por ejemplo, con la factorización de Cholesky o con la factorización QR.

```
function [x] = tridiagonal_solve_via_QR(a, b, c, y),
   A = tridiagonal_full_fast(a, b, c);
   [Q, R] = qr(A);
   x = Q' * (R \ y);
end
```

En vez de aplicar las funciones estándares `qr`, `chol` y el comando `R \ y`, se recomienda escribir sus propias funciones que realicen la solución de sistemas con matrices triangulares, la factorización QR o la factorización de Cholesky.

26. Pruebas pequeñas.

```
function [] = small_test_tridiagonal_solve(),
   a = [5; 3; 4; 7];
   b = [-1; 1; 2];
   u = [-2; 1; 3; 2];
   y = tridiagonal_full_fast(a, b, b) * u;
   tol = 1.0E-8;
   x1 = tridiagonal_solve_via_QR(a, b, b, y);
   x2 = tridiagonal_Jacobi_solver(a, b, b, y, tol, 50);
   x3 = tridiagonal_gradient_solver(a, b, b, y, tol, 50);
   x4 = tridiagonal_conjugate_gradient_solver(a, b, b, y, tol, 50);
   display([u x1 x2 x3 x4]);
end
```

27. Pruebas grandes. Generar matrices de la clase elegida de tamaños grandes con algunos datos admisibles (para que la matriz sea simétrica y positiva definida), aplicar 4 métodos de solución de sistemas escritos abajo y comparar los resultados.

I. A partir de los datos iniciales construir la matriz en el formato completo y resolver el sistema con la descomposición QR (o con otro método directo).

II. Aplicar el método de Jacobi con la función que hace la multiplicación rápida.

III. Usar el método de gradiente con la función que hace la multiplicación rápida.

IV. Usar el método de gradiente conjugado con la función que hace la multiplicación rápida.

En la tabla de comparación escribir el tiempo de ejecución (en segundos) y la norma del vector $\|b - Ax\|$.

	$n = 256$	$n = 512$	$n = 1024$	$n = 2048$	$n = 4096$
I.	???	???	???	???	???
II.	???	???	???	???	???
III.	???	???	???	???	???
IV.	???	???	???	???	???

Tarea optativa: precondicionamiento

28. Para matrices A de la clase elegida encontrar alguna receta para construir una matriz de precondicionamiento M. La matriz de precondicionemiento también debe ser simétrica y positiva definida, y el producto MA debe ser cercano a la matriz identidad I_n, en cierto sentido. Por ejemplo, la matriz $MA - I_n$ debe ser pequeña (en el sentido de alguna norma matricial), o la mayor parte de los valores propios de la matriz MA debe concentrarse cerca del número 1. Luego realizar el método de gradiente conjugado con precondicionamiento y comparar la eficiencia con el método sin precondicionamiento.