# Programación: comparación de la eficiencia de varios métodos iterativos para resolver sistemas de ecuaciones lineales

**Objetivos.** Comparar la eficiencia (rapidez) de varios métodos iterativos aplicándolos al mismo sistema de ecuaciones lineales.

**Requisitos.** Haber programado algún método directo para resolver sistemas de ecuaciones lineales y varios métodos iterativos, incluso el método de gradiente conjugado.

**1. Restricciones para la matriz del sistema.** La matriz $A$ debe ser cuadrada, real, simétrica y positiva definida. Matrices de este tipo se pueden generar como

$$A = B^\top B + hI_n,$$

donde $B \in \mathcal{M}_n(\mathbb{R})$ es una matriz arbitraria y $h > 0$. Para crear una matriz simétrica estrictamente diagonal dominante (y con esto garantizar la convergencia del método de Jacobi) y al mismo tiempo positiva definida, se pueden usar los siguientes comandos:

$$A = B + B^\top, \qquad h = \|A\|_{\mathrm{matr},\infty}, \quad A = A + hI_n.$$

Para acelerar la convergencia de los métodos iterativos se recomienda elegir $h$ bastante grande.

**2. Pruebas con matrices pequeñas.** Para asegurarse que todas las funciones programadas son correctas, se recomienda hacer una prueba con matrices pequeñas.

```
function [] = smalltest_iterative_comparison(),
   n = 3;
   B = rand(n);
   A = B' + B;
   h = norm(A, Inf);
   A = A + (h + 5) * eye(n);
   disp(norm(A - A')); # check if A is symmetric
   disp(eig(A)); # check if A is positive definite
   u = rand(n, 1); # exact solution
   b = A * u;
   # direct methods
   x1 = solve_with_LU(A, b);
   x2 = solve_with_QR(A, b);
   x3 = solve_with_Choleski(A, b);
   disp([norm(x1 - u), norm(x2 - u), norm(x3 - u)]);
   # iterative methods
```

```
   tol = 1.0E-6; smax = 2 * n + 100;
   [x4, s4] = solve_with_Jacobi(A, b, tol, smax);
   [x5, s5] = solve_with_Gauss_Seidel(A, b, tol, smax);
   [x6, s6] = solve_with_gradient_method(A, b, tol, smax);
   [x7, s7] = solve_with_conjugate_gradient(A, B, tol, smax);
   disp([norm(x4-u), norm(x5-u), norm(x6-u), norm(x7-u)]);
   disp([s4, s5, s6, s7]);
end
```

**3. Pruebas con matrices grandes.** Modificando la función anterior escribir una función

```
function [] = largetest_iterative_comparison(n),
   ...
end
```

que haga pruebas con matrices del tamaño indicado $n$ y que muestre el tiempo de ejecución de cada método (en segundos).

**4. Tabla de resultados.** Escribir los resultados en una tabla. Se recomienda poner $n$ igual a potencias de 2. Aumentar $n$ hasta valores tan grandes que el tiempo de ejecución sea de 10 a 100 segundos. Un ejemplo con datos aleatorios, solamente para mostrar la idea:

| | $n = 128$ | $n = 256$ | $n = 512$ | $n = 1024$ |
|---|---|---|---|---|
| LU | $t = 2$ | $t = 15$ | $t > 100$ | $t > 100$ |
| QR | $t = 3$ | $t = 22$ | $t > 100$ | $t > 100$ |
| Choleski | $t = 1$ | $t = 7$ | $t = 50$ | $t > 100$ |
| Jacobi | $t = 0.2$<br>$s = 80$ | $t = 0.8$<br>$s = 290$ | no conv. | no conv. |
| G-S | $t = 0.1$<br>$s = 50$ | $t = 0.3$<br>$s = 105$ | no conv. | no conv. |
| grad. | $t = 0.2$<br>$s = 60$ | $t = 0.7$<br>$s = 110$ | $t = 3$<br>$s = 150$ | no conv. |
| grad. conj. | $t = 0.07$<br>$s = 7$ | $t = 0.35$<br>$s = 11$ | $t = 2$<br>$s = 30$ | $t = 30$<br>$s = 60$ |

En las celdas de la tabla indicar el tiempo de ejecución en segundos y (en el caso de métodos iterativos) el número de las iteraciones hechas. Si el número de los pasos hechos es mayor o igual que el número máximo de los pasos permitidos, entonces escribir "no converge". Si el tiempo de ejecución es más grande que 100 segundos, entonces se puede escribir "$t > 100$".