Programación: el método del gradiente conjugado

Objetivos. Programar el método iterativo del gradiente conjugado.

Requisitos. Solución de sistemas simétricas positivas definidas usando métodos iterativos con búsqueda exacta sobre cada recta, ortogonalización de Gram–Schmidt.

1. El mínimo de una forma cuadrática sobre una recta (repaso). Sea A una matriz real simétrica estrictamente positiva definida de orden n, y sea $b \in \mathbb{R}^n$. Dado un vector $x \in \mathbb{R}^n$ y una *dirección* $p \in \mathbb{R}^n \setminus \{0\}$, consideremos la función

$$g(\alpha) = \frac{1}{2}(x + \alpha p)^\top A(x + \alpha p) - (x + \alpha p)^\top b.$$

Escriba g como un polinomio de grado 2 en la variable α, calcule g' y encuentre el punto $\alpha_{\min}$ donde f alcanza su valor mínimo. Se recomienda usar la notación $r = b - Ax$.

2. El cambio del residuo al cambiar el vector (repaso). Sean A, b, x, p, r los mismos que en el ejercicio anterior. Definimos el vector nuevo y como

$$y = x + \alpha p.$$

Exprese el residuo nuevo $b - Ay$ a través del residuo previo:

$$b - Ay = \underbrace{b - Ax}_{r} + ???.$$

3. *A*-ortogonalización del residuo con respecto a la dirección anterior. Supongamos que $r, p \in \mathbb{R}^n$, $p \neq \mathbf{0}_n$. Encuentre un escalar $\beta \in \mathbb{R}^n$ tal que

$$p^\top A(r - \beta p) = 0.$$

4. Algoritmo (método del gradiente conjugado).

```
function [x, s] = conjgrad(A, b, tol, smax),
   n = length(b); x = zeros(n, 1);
   r = b; p = r; s = 0;
   while ??? && ???,
      q = A * p;
      al = ???;
      x = ???; r = ???;
      be = ???;
      p = r - be * p;
      s = s + 1;
   end
end
```

5. Prueba con datos pequeños.

```
function [] = smalltest_conjgrad(),
   A = [3 -2; -2 2]; u = [1; -6]; # exact solution
   b = A * u; n = length(b);
   x = conjgrad(A, b, 1.0E-8, n);
   disp(x);
   disp(norm(A * x - b));
end
```

6. Prueba en el plano con un dibujo. El siguiente código consiste de dos funciones y se debe guardar en un archivo `plotgradientdescent.m`

```
function [] = plotcg(),
   A = [3 -2; -2 2]; u = [1; -6]; b = A * u; n = length(b);
   xsequence = zeros(n, 3);
   for s = 1 : 2,
      xsequence(:, s + 1) = conjgrad(A, b, 1.0E-8, s);
   end
   disp(xsequence);
   r = 1.2 * max(abs(u));
   plotdomain = [u(1) - r, u(1) + r, u(2) - r, u(2) + r];
   plot(xsequence(1, :), xsequence(2, :), '-w', 'linewidth', 3);
   hold on;
   ezcontourf(@(var1, var2) f(var1, var2, A, b, u), plotdomain, 100);
end

function result = f(var1, var2, A, b, u),
   [size1, size2] = size(var1);
   result = zeros(size1, size2);
   for j = 1 : size1,
      for k = 1 : size2,
         x = [var1(j, k); var2(j, k)];
         result(j, k) = sqrt(0.5 * (x - u)' * A * (x - u));
      end
   end
end
```

7. Prueba con datos grandes.

```
function [] = largetest_conjgrad(),
   n = 100; R = rand(n, n); A = R' * R + eye(n); b = rand(n, 1);
   t1 = cputime(); [x, s] = conjgrad(A, b, 1.0E-5, n); t2 = cputime();
   disp(norm(A * x - b)); disp(s); disp(t2 - t1);
end
```