Métodos de Computación Científica

Trabajo práctico de programación #3

03-10-2012

Victoria Martínez de la Cruz - LU. 87620

1) Ingresamos la matriz A y el vector de soluciones b. Hallamos la descomposición LU de A usando la función predefinida `lu(A)` y procedemos a resolver el sistema.

```
>> A = [-4 1 1 0; 1 -4 0 1; 1 0 -4 1; 0 1 1 -4]
A =

  -4   1   1   0
   1  -4   0   1
   1   0  -4   1
   0   1   1  -4

>> b = [-200 -400 0 200]'
b =

  -200
  -400
     0
   200

>> [L,U] = lu(A)
L =

   1.00   0.00   0.00  0.00
  -0.25   1.00   0.00  0.00
  -0.25  -0.07   1.00  0.00
  -0.00  -0.27  -0.29  1.00

U =

 -4.00   1.00   1.00   0.00
  0.00  -3.75   0.25   1.00
  0.00   0.00  -3.73   1.07
  0.00   0.00   0.00  -3.43
```

Aprovechamos que las matrices obtenidas son triangulares para resolver el sistema de forma más eficiente.

Para esto, usamos el comando `linsolve(A,B,opts).` Este comando nos permite especificar la forma de resolución más apropiada según el formato de la matriz A modificando el valor del parámetro opts.

Primero resolvemos el sistema Ly = B, donde y = Ux

En este caso, L es triangular inferior. Por eso, establecemos el valor de LT.opts en true y procedemos a resolver.

```
>> opts.LT = true;
>> r1 = linsolve(L,b,opts)
```

A continuación resolvemos Ux = y, cuyo resultado será el resultado definitivo. Establecemos el valor de opts adecuadamente y resolvemos el sistema.

```
>> opts.UT = true;
>> opts.LT = false;
>> r1 = linsolve(U,r1,opts)

r1 =
       100.0000
       150.0000
       50.0000
       100.0000
```

2) Para resolver el sistema usando el método SOR se implementó la siguiente función

```
function [x, error, iter, flag]  = sor(A, x, b, w, max_it, tol)

% sor.m resuelve sistemas lineales de la forma Ax=b usando
% el método SOR (si omega = 1 equivale al método Gauss-Seidel).
%
% input   A        Matriz A de números reales
%         x        Vector x inicial
%         b        Vector b
%         w        Omega
%         max_it   Máximo número de iteraciones
%         tol      Tolerancia de error
%
% output  x        Vector solución
%         error    Error
%         iter     Número de iteraciones realizadas
%         flag     0 = solución encontrada
%                  1 = no converge
```

```
  flag = 0;                                   % inicializacion
  iter = 0;

  bnrm2 = norm( b );
  if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end

  r = b - A*x;
  error = norm( r ) / bnrm2;
  if ( error < tol ) return, end

  [ M, N, b ] = split( A, b, w, 2 );          % split

  for iter = 1:max_it                         % comienzo de iteraciones
     x_1 = x;
     x   = M \ ( N*x + b );                   % aproximación actualizada

     error = norm( x - x_1 ) / norm( x );     % error
     if ( error <= tol ), break, end          % controlar convergencia

  end
  b = b / w;

  if ( error > tol ) flag = 1; end;           % no converge

% FIN sor.m
```

El valor de omega puede ser calculado con la siguiente función

```
function w = w_optimo(A, b)

n = size(A,1);      % hallamos L, U and D de A
D = diag(diag(A));
L = tril(-A,-1);
U = triu(-A,1);

Tj = inv(D)*(L+U);                 % hallamos la matriz de iteraciones de Jacobi
rho_Tj = max(abs(eig(Tj)));        % hallamos el radio espectral de Tj
w = 2./(1+sqrt(1-rho_Tj^2));       % hallamos el omega óptimo
```

La salida de esa función nos garantiza un w óptimo.

Finalmente el resultado de ejecutar la función SOR para nuestro sistema, considerando un máximo de 20 iteraciones y un error menor a 10^-6

```
>> A = [4 -2 1; 1 5 -3; 2 2 5]
A =

   4  -2   1
   1   5  -3
   2   2   5

>> b = [11 -6 7]'
b =

   11
   -6
    7

>> x = [0 0 0]'
x =

   0
   0
   0


>> sor(A,x,b,w_optimo(A,b),20,1.0000e-06)

ans =

   2
  -1
   1
```

3) Se implementó una función que recibe como parámetros de entrada los puntos dato y la cantidad de datos a ingresar, n.

Se construye el sistema de ecuaciones indicado en el enunciado y luego se resuelve para obtener las incógnitas a, b y c.

```
function v = ejercicio3(x, y, n)
M = zeros(3);
y1 = 0;
```

```
y2 = 0;
y3 = 0;
M(1,1) = n;

for( i=1:n )
        M(1,2) = M(1,2) + x(i);
        M(1,3) = M(1,3) + x(i)^2;
        M(2,3) = M(2,3) + x(i)^3;
        M(3,3) = M(3,3) + x(i)^4;
        y1 = y1 + y(i);
        y2 = y2 + y(i)*x(i);
        y3 = y3 + y(i)*(x(i)^2);
end

M(2,1) = M(1,2);
M(2,2) = M(1,3);
M(3,1) = M(2,2);
M(3,2) = M(2,3);

v = linsolve(M,[y1; y2; y3]);
```

Podrían utilizarse los mismos resultados usando la función `polyfit` de Matlab.

4)

Usando la función definida en el ejercicio 3 se obtiene

```
>> ejercicio3( [0,1,2,3,4,5],[2.5,9.2,13.3,26.7,31.8,50.4] , 6)
ans =
3.1893
3.4932
1.1339
```

Usando `polyfit` se obtiene

```
>> polyfit( [0,1,2,3,4,5],[2.5,9.2,13.3,26.7,31.8,50.4] , 2)
ans =
1.1339 3.4932 3.1893
```

El último parámetro de `polyfit` indica que se quiere obtener un polinomio cuadrático (de grado 2)

Como los resultados fueron equivalentes, podríamos decir que ha sido implementada correctamente.