Libro: Quarteroni, “Scientific Computing with MATLAB and Octave”
Capítulo 5: SISTEMAS LINEALES.
Sección 5.9: MÉTODOS ITERATIVOS

Programa itermeth.m: (para Jacobi, Gauss-Seidel y método del Gradiente)

```
function [x,iter ]= itermeth (A,b,x0 ,nmax ,tol ,P)
% ITERMETH General iterative method
% X = ITERMETH (A,B,X0 ,NMAX ,TOL ,P) attempts to solve the
% system of linear equations A*X=B for X. The N-by -N
% coefficient matrix A must be non -singular and the
% right hand side column vector B must have length
% N. If P='J' the Jacobi method is used , if P='G' the
% Gauss -Seidel method is selected . Otherwise , P is a
% N-by -N matrix that plays the role of a preconditioner
% for the gradient method , which is a dynamic
% Richardson method. Iterations
% stop when the ratio between the norm of the kth
% residual and the norm of the initial residual is less
% than TOL , then ITER is the number of performed
% iterations . NMAX specifies the maximum
% number of iterations . If P is not defined , the
% unpreconditioned gradient method is performed .
[n,n]= size (A);
if nargin == 6
if ischar(P)==1
if P=='J'
L=diag (diag(A)); U=eye(n); beta =1; alpha =1;
elseif P == 'G'
L=tril (A); U=eye(n); beta =1; alpha =1;
end
else
[L,U]= lu(P); beta = 0;
end
else
L = eye(n); U = L; beta = 0;
end
iter =0; x=x0; r=b-A*x0; r0=norm (r); err=r0;
while err > tol & iter < nmax
z = L\r; z = U\z; iter = iter + 1;
if beta == 0
alpha = z'*r/(z'*A*z);
end
x = x + alpha*z;
r = b - A * x;
err = norm (r) / r0;
end
```

<u>EJEMPLO 5.13</u>: Resolver A x = b, donde A y b son definidos de la siguiente manera

```
>> n=10;
>> A=3*eye(n)
A =
   3   0   0   0   0   0   0   0   0   0
   0   3   0   0   0   0   0   0   0   0
   0   0   3   0   0   0   0   0   0   0
   0   0   0   3   0   0   0   0   0   0
   0   0   0   0   3   0   0   0   0   0
   0   0   0   0   0   3   0   0   0   0
   0   0   0   0   0   0   3   0   0   0
```

```
     0     0     0     0     0     0     0     3     0     0
     0     0     0     0     0     0     0     0     3     0
     0     0     0     0     0     0     0     0     0     3
>> A=A+(-2)*diag( ones(n-1,1) , 1 )
A =
     3    -2     0     0     0     0     0     0     0     0
     0     3    -2     0     0     0     0     0     0     0
     0     0     3    -2     0     0     0     0     0     0
     0     0     0     3    -2     0     0     0     0     0
     0     0     0     0     3    -2     0     0     0     0
     0     0     0     0     0     3    -2     0     0     0
     0     0     0     0     0     0     3    -2     0     0
     0     0     0     0     0     0     0     3    -2     0
     0     0     0     0     0     0     0     0     3    -2
     0     0     0     0     0     0     0     0     0     3
>> A=A+(-1)*diag( ones(n-1,1) , -1 )
A =
     3    -2     0     0     0     0     0     0     0     0
    -1     3    -2     0     0     0     0     0     0     0
     0    -1     3    -2     0     0     0     0     0     0
     0     0    -1     3    -2     0     0     0     0     0
     0     0     0    -1     3    -2     0     0     0     0
     0     0     0     0    -1     3    -2     0     0     0
     0     0     0     0     0    -1     3    -2     0     0
     0     0     0     0     0     0    -1     3    -2     0
     0     0     0     0     0     0     0    -1     3    -2
     0     0     0     0     0     0     0     0    -1     3
>> b=A*ones(n,1)
b =
     1
     0
     0
     0
     0
     0
     0
     0
     0
     2
```

Usaremos el método de Jacobi y Gauss-Seidel para encontrar x. El vector de partida es el vector nulo y las iteraciones serán detenidas haciendo tol=10^{-12} (lea los comentarios del programa itermeth.m). El máximo número de iteraciones es 400.

```
>> x0=zeros(n,1);
>> [x,iterJ]=itermeth(A,b,x0,400,1.e-12,'J')                    % SOLUCIÓN CON MÉTODO DE JACOBI
x =
     9.999999999978660e-01
     9.999999999973692e-01
     9.999999999971378e-01
     9.999999999977868e-01
     9.999999999981258e-01
     9.999999999987960e-01
     9.999999999991388e-01
```

```
    9.999999999995403e-01
    9.999999999997440e-01
    9.999999999999143e-01
iterJ =
   277
>> [x,iterG]=itermeth(A,b,x0,400,1.e-12,'G')      % SOLUCIÓN CON MÉTODO DE GAUSS-SEIDEL
x =
    9.999999999978892e-01
    9.999999999974091e-01
    9.999999999976833e-01
    9.999999999982164e-01
    9.999999999987585e-01
    9.999999999992057e-01
    9.999999999995330e-01
    9.999999999997518e-01
    9.999999999998864e-01
    9.999999999999621e-01
iterG =
   143
```

El método de Gauss-Seidel es más rápido que el de Jacobi.

<u>EJEMPLO 5.15</u>: Resolver A x = b, donde A y b son

```
>> A                                         % A ES UNA MATRIZ 100 x 100
A =
    4   -1   -1    0    0    0    0    0    0    0
   -1    4   -1   -1    0    0    0    0    0    0
   -1   -1    4   -1   -1    0    0    0    0    0
    0   -1   -1    4   -1   -1    0    0    0    0
    0    0   -1   -1    4   -1   -1    0    0    0
    0    0    0   -1   -1    4   -1   -1    0    0
    0    0    0    0   -1   -1    4   -1   -1    0
    0    0    0    0    0   -1   -1    4   -1   -1
    0    0    0    0    0    0   -1   -1    4   -1
    0    0    0    0    0    0    0   -1   -1    4
>> b                                         % b ES UN VECTOR 100 x 1
b =
    2
    1
    0
    0
    0
    0
    0
    0
    1
    2
```

La matriz A es simétrica y todos sus autovalores son positivos (usar el comado eig(A) del Matlab). Es decir, A es simétrica definida positiva. Usaremos el método de Jacobi, Gauss-Seidel y de Gradiente para encontrar x. El vector de partida es el vector nulo y las iteraciones serán detenidas haciendo tol=10^{-5}. El máximo número de iteraciones es 5000.

```
>> x0=zeros(n,1);
>> [x,iterJ]=itermeth(A,b,x0,5000,1.e-5,'J')           % SOLUCIÓN CON MÉTODO DE JACOBI
```

```
x =
   9.998628370054627e-01
   9.997782193005722e-01
   9.996737848726449e-01
   …..
   9.967946616056959e-01
   9.967915913830186e-01
   9.967915913830186e-01
   9.967946616056960e-01
   …..
   9.996737848726449e-01
   9.997782193005723e-01
   9.998628370054627e-01
iterJ =
     5000
```

El método de Jacobi se detiene con el numéro máximo de iteraciones!!

```
>> [x,iterG]=itermeth(A,b,x0,5000,1.e-5,'G')      % SOLUCIÓN CON MÉTODO DE GAUSS-SEIDEL
x =
   9.999589884464646e-01
   9.999337392078773e-01
   9.999026066966885e-01
         ….
   9.990741362617592e-01
   9.990739146298653e-01
   9.990745793559509e-01
   9.990761285321503e-01
         ….
   9.999090317264413e-01
   9.999381981140697e-01
   9.999618074601277e-01
iterG =
     3020
```

El método de Gauss-Seidel se detiene antes del número máximo de iteraciones. Es decir, esta solución corresponde a un residual relativo menor a 10^{-5}.

```
>> [x,iterGM]=itermeth(A,b,x0,5000,1.e-5)       % SOLUCIÓN CON MÉTODO DEL GRADIENTE
x =
   9.999718833726567e-01
   9.999544336615868e-01
   9.999330181913602e-01
         ….
   9.993427819680506e-01
   9.993410787578588e-01
   9.993410787578592e-01
   9.993427819680507e-01
         ….
   9.999330181913604e-01
   9.999544336615868e-01
   9.999718833726567e-01
iterGM =
     4939
```

El método del gradiente es más lento que Gauss-Seidel, pero llega a una solución más precisa.
Ahora usamos un pre-condicionador P, que es la siguiente matriz 100 x 100

```
>> P                                        % P ES UNA MATRIZ 100 x 100
P =
   2  -1   0   0   0   0   0   0   0   0
  -1   2  -1   0   0   0   0   0   0   0
   0  -1   2  -1   0   0   0   0   0   0
   0   0  -1   2  -1   0   0   0   0   0
   0   0   0  -1   2  -1   0   0   0   0
   0   0   0   0  -1   2  -1   0   0   0
   0   0   0   0   0  -1   2  -1   0   0
   0   0   0   0   0   0  -1   2  -1   0
   0   0   0   0   0   0   0  -1   2  -1
   0   0   0   0   0   0   0   0  -1   2
```

Note que esta matriz es simétrica y definida positiva (verifique!). Con P se aplica el método de gradiente pre-condicionado.

```
                    % SOLUCIÓN CON MÉTODO DEL GRADIENTE PRE-CONDICIONADO
>> [x,iterPGM]=itermeth(A,b,x0,5000,1.e-5,P)
x =
    9.999985716869449e-01
    1.000000286593749e+00
    9.999966686104097e-01
         ….
    9.999933483838740e-01
    9.999933483838741e-01
    9.999933483838742e-01
    9.999933483838744e-01
         ….
    9.999966686104114e-01
    1.000000286593750e+00
    9.999985716869455e-01
iterPGM =
   22
```

Con el pre-condicionador se acelera la convergencia. Al calcular los números de condición de A y P^(-1)*A se tiene:

```
>> cond(A,2)
ans =
  1.3058e+03
>> cond(inv(P)*A,2)
ans =
   9.4777
```

Finalmente podemos calcular directamente la solución. El Matlab selecciona el mejor método con el comando \ . En este caso, el Matlab usa probablemente una descomposición de Cholesky.

```
>> x=A\b
x =
    1.000000000000001e+00
    1.000000000000001e+00
    1.000000000000002e+00
         ….
    1.000000000000019e+00
    1.000000000000019e+00
    1.000000000000019e+00
         ….
```

1.000000000000003e+00
1.000000000000002e+00
1.000000000000001e+00

CONCLUSIÓN:
Los métodos iterativos con precondicionamiento son una opción alternativa a los métodos directos. Sobre todo, cuando la matriz A del sistema Ax=b es mal condicionada.

Table 5.4. Errors obtained using the preconditioned gradient method (PG), the preconditioned conjugate gradient method (PCG), and the direct method implemented in the MATLAB command \ for the solution of the Hilbert system. For the iterative methods also the number of iterations is reported

		\	PG		PCG	
n	$K(\mathrm{A}_n)$	Error	Error	Iter	Error	Iter
4	1.55e+04	7.72e-13	8.72e-03	995	1.12e-02	3
6	1.50e+07	7.61e-10	3.60e-03	1813	3.88e-03	4
8	1.53e+10	6.38e-07	6.30e-03	1089	7.53e-03	4
10	1.60e+13	5.24e-04	7.98e-03	875	2.21e-03	5
12	1.70e+16	6.27e-01	5.09e-03	1355	3.26e-03	5
14	6.06e+17	4.12e+01	3.91e-03	1379	4.32e-03	5

Figura 1. Fuente: Quarteroni, Ejemplo 5.16