Metodi iterativi

26 febbraio 2008

1 Introduzione

Sia A una matrice reale avente n righe ed n colonne, b un vettore colonna avente n righe e si supponga di voler risolvere il sistema lineare $Ax = b$. Come noto, se il determinante della matrice è diverso da 0 (cioè la matrice A è non singolare) allora il problema $Ax = b$ ha una ed una sola soluzione.

Ricordiamo che in Matlab/Octave la soluzione può essere calcolata con il metodo LU, utilizzando il comando \. Un esempio:

```
>> A=[1 2 4; 2 4 16; 3 9 81];
>> b=ones(3,1);
>> x=A\b
>> norm(A*x-b)
ans =  9.9301e-16
>> det(A)
ans = -24.000
```

Uno dei principali problemi del metodo LU è legato all'alto costo computazionale. Se A è una generica matrice quadrata di ordine n infatti necessitano circa

$$O\left(\frac{n^3}{3} + \frac{n^2}{2}\right)$$

operazioni moltiplicative, che possono risultare eccessive nel caso di matrici di grandi dimensioni. Per ovviare a questo problema si usano metodi iterativi (stazionari) del tipo

$$x^{(k+1)} = P\,x^{(k)} + c,\ \ k = 0, 1, \ldots$$

con P dipendente da A e c dipendente da A e b (ma non da k). A differenza dei metodi diretti (come ad esempio il metodo LU), in genere un metodo iterativo stazionario convergente calcola usualmente solo un approssimazione della soluzione x (a meno di una tolleranza prefissata). Se m è il numero di iterazioni necessarie, visto che ogni iterazione ha un costo $O(n^2)$ dovuto al prodotto matrice-vettore $P\,x^{(k)}$, ci si augura che il costo computazionale $O(m\,n^2)$ del metodo iterativo sia di gran lunga inferiore a $O(\frac{n^3}{3} + \frac{n^2}{2})$ di un metodo diretto quale LU.

1.1 I metodi di Jacobi, Gauss-Seidel e SOR

Sia $A = M - N$ con M non singolare, un generico *metodo iterativo stazionario* e' del tipo

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b. \tag{1}$$

La matrice $P = M^{-1}N$ è usualmente chiamata *matrice di iterazione* del metodo iterativo stazionario definito da M, N. Osserviamo che posto $c = M^{-1}b$, il metodo sopracitato è ovviamente tystazionario essendo

$$x^{(k+1)} = Px^{(k)} + c \tag{2}$$

con P e c indipendenti da k.

Questa definizione dei metodi stazionari, forse un po' astratta, ha il vantaggio di offrire una rappresentazione compatta degli stessi ed è comunemente utilizzata in letteratura.
Sia ora $A = D - E - F$ con D matrice diagonale, E, F rispettivamente triangolare inferiore e superiore con elementi diagonali nulli.

Nel caso del **metodo di Jacobi** (1845) si ha

$$M = D,\ N = E + F \tag{3}$$

e quindi

$$P = M^{-1}N = D^{-1}(E+F) = D^{-1}(D-D+E+F) = D^{-1}(D-A) = I - D^{-1}A \tag{4}$$

Si osservi che se D è non singolare allora il metodo di Jacobi, almeno in questa versione di base, non può essere utilizzato visto che in (7) non ha senso la scrittura D^{-1}.

Qualora sia $a_{ii} \neq 0$ per ogni $i = 1, \ldots, n$, il metodo di Jacobi può essere descritto come

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)})/a_{ii},\ i = 1, \ldots, n. \tag{5}$$

Un codice Matlab/Octave del metodo di Jacobi,fornito in internet presso il sito di Netlib

http://www.netlib.org/templates/matlab/

è il seguente

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

%  -- Iterative template routine --
%     Univ. of Tennessee and Oak Ridge National Laboratory
%     October 1, 1993
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
%     Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
```

```
%     1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%
% [x, error, iter, flag]  = jacobi(A, x, b, max_it, tol)
%
% jacobi.m solves the linear system Ax=b using the Jacobi Method.
%
% input   A        REAL matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
%
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it

  iter = 0;                                       % initialization
  flag = 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]=size(A);
  [ M, N ] = split( A , b, 1.0, 1 );       % matrix splitting

  for iter = 1:max_it,                            % begin iteration

     x_1 = x;
     x   = M \ (N*x + b);                         % update approximation

     error = norm( x - x_1 ) / norm( x );         % compute error
     if ( error <= tol ), break, end              % check convergence

  end

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

Il codice di `jacobi` utilizza una funzione `split` che serve per calcolare le matrici M, N che definiscono l'iterazione del metodo di Jacobi:

```
function [ M, N, b ] = split( A, b, w, flag )
%
% function [ M, N, b ] = split_matrix( A, b, w, flag )
%
```

```
% split.m sets up the matrix splitting for the stationary
% iterative methods: jacobi and sor (gauss-seidel when w = 1.0 )
%
% input   A       DOUBLE PRECISION matrix
%         b       DOUBLE PRECISION right hand side vector (for SOR)
%         w       DOUBLE PRECISION relaxation scalar
%         flag    INTEGER flag for method: 1 = jacobi
%                                          2 = sor
%
% output  M       DOUBLE PRECISION matrix
%         N       DOUBLE PRECISION matrix such that A = M - N
%         b       DOUBLE PRECISION rhs vector ( altered for SOR )

  [m,n] = size( A );

  if ( flag == 1 ),                   % jacobi splitting

     M = diag(diag(A));
     N = diag(diag(A)) - A;

  elseif ( flag == 2 ),               % sor/gauss-seidel splitting

     b = w * b;
     M =  w * tril( A, -1 ) + diag(diag( A ));
     N = -w * triu( A,  1 ) + ( 1.0 - w ) * diag(diag( A ));

  end;

% END split.m
```

Ricordiamo che la funzione `split` non coincide con quella predefinita nelle ultime releases di Matlab/Octave. Qualora la funzione `split` che vogliamo utilizzare sia salvata della directory corrente, una volta richiamata, i workspace di Matlab/Octave utilizzano proprio questa e non quella descritta per altri usi in Matlab/Octave. Inoltre per quanto riguarda `tril` e `triu` in `split` dall'help di Matlab si capisce che estraggono rispettivamente la parte triangolare inferiore e superiore di una matrice:

```
>> help tril

 TRIL Extract lower triangular part.
    TRIL(X) is the lower triangular part of X.
    TRIL(X,K) is the elements on and below the K-th diagonal
    of X .  K = 0 is the main diagonal, K > 0 is above the
    main diagonal and K < 0 is below the main diagonal.

    See also TRIU, DIAG.

>> help triu
```

```
 TRIU Extract upper triangular part.
    TRIU(X) is the upper triangular part of X.
    TRIU(X,K) is the elements on and above the K-th diagonal of
    X.  K = 0 is the main diagonal, K > 0 is above the main
    diagonal and K < 0 is below the main diagonal.

    See also TRIL, DIAG.

>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> tril(A)
ans =
     1     0     0
     4     5     0
     7     8     9
>> triu(A)
ans =
     1     2     3
     0     5     6
     0     0     9
>>
```

La routine `jacobi` è scritta da esperti di algebra lineare e si interrompe quando la norma 2 dello step relativo

$$\frac{\|x^{(k+1)} - x^{(k)}\|_2}{\|x^{(k+1)}\|_2}$$

è inferiore ad una tolleranza `tol` prefissata oppure un numero massimo di iterazioni `max_it` è raggiunto. Ricordiamo che se $v = (v_i)_{i=1,\dots,n}$ è un elemento di $\mathbb{R}^n$ allora

$$\|v\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}.$$

Problema: cosa succede quando la matrice diagonale estratta da A è singolare? cosa succede quando partendo da $x_0 \neq 0$, si ha per qualche indice $k > 0$ che $x_k = 0$?

Un altro metodo di particolare interesse é quello di **Gauss-Seidel** (1874) per cui

$$M = D - E, N = F \tag{6}$$

e quindi

$$P = M^{-1}N = (D - E)^{-1}F \tag{7}$$

Similmente al metodo di Jacobi, possiamo riscrivere più semplicemente anche Gauss-Seidel come

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) / a_{ii}. \tag{8}$$

Da (8) si capisce perchè tale metodo è noto anche come *metodo delle sostituzioni successive*.

Per accelerare la cosidetta *velocitá di convergenza* si introducono, per un opportuno parametro ω, la versione *rilassata* del metodo di Jacobi

$$x^{(k+1)} = (I - \omega D^{-1} A) x^{(k)} + \omega D^{-1} b \tag{9}$$

e di Gauss-Seidel

$$x^{(k+1)} = \left(\frac{D}{\omega} - E\right)^{-1} \left(\left(\frac{1}{\omega} - 1\right) D + F\right) x^{(k)} + \left(\frac{D}{\omega} - E\right)^{-1} b. \tag{10}$$

L'idea di fondo di questi metodi rilassati è la seguente [2, p. 261]. Ogni metodo precedentemente esposto può essere scritto come

$$x^{(k+1)} = x^{(k)} + r^{(k)}$$

ove $r^{(k)}$ è la correzione da apportare per passare da $x^{(k)}$ a $x^{(k+1)}$. Nei metodi rilassati, se $r^{(k)}$ è la correzione di Jacobi o Gauss-Seidel, si considera quale correzione $w \cdot r^{(k)}$ e quindi

$$x^{(k+1)} = x^{(k)} + w \cdot r^{(k)}.$$

Si osservi che i metodi di Jacobi e Gauss-Seidel si ottengono rispettivamente da (9) e (10) per la scelta $\omega = 1$. Esistono delle buone scelte di tale parametro ω detto di *rilassamento*? La risposta è affermativa. Sia $\rho(P)$ il massimo degli autovalori in modulo della matrice di iterazione $P = M^{-1}N$ (il cosidetto *raggio spettrale*). Si dimostra che un metodo iterativo (stazionario) definito da P converge per ogni vettore iniziale x_0 se e solo se $\rho(P) < 1$. Se

$$R(P) = -\ln(\rho(P))$$

è la cosidetta *velocità di convergenza asintotica* del metodo iterativo relativo a P, si può dimostrare che il numero di iterazioni k necessarie per ridurre l'errore di un fattore ϵ verifica la disuguaglianza

$$k \geq \frac{-\ln(\epsilon)}{R(P)}.$$

Conseguentemente minore è $\rho(P)$ necessariamente è maggiore $R(P)$ e si può *stimare* il numero di iterazioni per ridurre l'errore di un fattore ϵ. Si desidera quindi cercare metodi con $\rho(P)$ più piccolo possibile.

La versione di Gauss-Seidel con la scelta del parametro ω è nota in letteratura come **SOR**, acronimo di *successive over relaxation*. Una versione di SOR scaricabile presso il sito di Netlib [6] è la seguente

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

%  -- Iterative template routine --
%     Univ. of Tennessee and Oak Ridge National Laboratory
%     October 1, 1993
```

```
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
%     Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
%     1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%
% [x, error, iter, flag]  = sor(A, x, b, w, max_it, tol)
%
% sor.m solves the linear system Ax=b using the
% Successive Over-Relaxation Method (Gauss-Seidel method when omega = 1 ).
%
% input   A        REAL matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         w        REAL relaxation scalar
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
%
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it

  flag = 0;                                   % initialization
  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 );          % matrix splitting

  for iter = 1:max_it                         % begin iteration

     x_1 = x;
     x   = M \ ( N*x + b );                   % update approximation

     error = norm( x - x_1 ) / norm( x );     % compute error
     if ( error <= tol ), break, end          % check convergence

  end
  b = b / w;                                  % restore rhs

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

Come per il metodo di Jacobi, il processo si interrompe quando la norma 2 dello step relativo

$$\frac{\|x^{(k+1)} - x^{(k)}\|_2}{\|x^{(k+1)}\|_2}$$

è inferiore ad una tolleranza `tol` prefissata oppure un numero massimo di iterazioni `max_it` è raggiunto.

Per ulteriori dettagli si consulti ad esempio [3, p. 313-339].

1.2 Convergenza dei Jacobi, Gauss-Seidel ed SOR

Lo studio della convergenza dei metodi di Jacobi, Gauss-Seidel ed SOR è un proposito complicato e ci limiteremo a citare, senza dimostrazione, alcuni classici risultati [2, p. 231-315].

Il metodo di Jacobi risulta convergente in uno dei seguenti casi [2, p. 247]:

1. A è a predominanza diagonale in senso stretto;
2. A è a predominanza diagonale ed è irriducibile;
3. A è a predominanza diagonale in senso stretto per colonne;
4. A è a predominanza diagonale per colonne ed è irriducibile.

Il metodo di Gauss-Seidel risulta convergente in uno dei seguenti casi [2, p. 249]:

1. A è a predominanza diagonale in senso stretto.
2. Sia A una matrice simmetrica definita positiva, non singolare con elementi principali $a_{i,i} \neq 0$. Allora Gauss-Seidel è convergente se e solo se A è definita positiva.

Per matrici tridiagonali (a blocchi) $A = (a_{i,j})$ con componenti diagonali non nulle, i metodi di Jacobi e Gauss-Seidel sono o entrambi convergenti o divergenti e il tasso di convergenza del metodo di Gauss-Seidel è il doppio di quello del metodo di Jacobi (il che vuol dire che *asintoticamente* sono necessarie metà iterazioni del metodo di Gauss-Seidel per ottenere la stessa precisione del metodo di Jacobi).

Ricordiamo che

1. A è a predominanza diagonale (per righe) se per ogni $i = 1, \ldots, n$ risulta

$$|a_{i,i}| \geq \sum_{j=1, j\neq s}^{n} |a_{i,j}|$$

e per almeno un indice s si abbia

$$|a_{s,s}| > \sum_{j=1, j\neq s}^{n} |a_{s,j}|.$$

Ad esempio la matrice

$$A = \begin{pmatrix} 4 & -4 & 0 \\ -1 & 4 & -1 \\ 0 & -4 & 4 \end{pmatrix}$$

è a predominanza diagonale (per righe).

2. A è a predominanza diagonale in senso stretto (per righe) se per ogni $i = 1, \ldots, n$ risulta

$$|a_{i,i}| > \sum_{j=1, j\neq i}^{n} |a_{i,j}|.$$

Ad esempio la matrice

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

è a predominanza diagonale in senso stretto (per righe).

3. A è a predominanza diagonale per colonne (in senso stretto) se A^T è a predominanza diagonale per righe (in senso stretto).

4. A è tridiagonale se $a_{i,j} = 0$ per $|i - j| > 1$. Ad esempio la matrice

$$A = \begin{pmatrix} 4 & -1 & 0 & \ldots & 0 \\ -1 & 4 & -1 & \ldots & 0 \\ 0 & -1 & 4 & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & -1 \\ 0 & 0 & \ldots & -1 & 4 \end{pmatrix}$$

è tridiagonale.

5. A è definita positiva se e solo se i suoi autovalori sono positivi.

La matrice

$$A = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}$$

è definita positiva come si può vedere usando i seguenti comandi Matlab/Octave

```
>> A=[4 -1 0; -1 4 -1; 0 -1 4]
A =
     4    -1     0
    -1     4    -1
     0    -1     4
>> eig(A)
ans =
    2.5858
```

```
    4.0000
    5.4142
>>
```

6. A di ordine $n \geq 2$ è riducibile se esiste una matrice di permutazione Π e un intero k con $0 < k < n$, tale che

$$B = \Pi A \Pi^T = \begin{pmatrix} A_{1,1} & A_{1,2} \\ 0 & A_{2,2} \end{pmatrix}$$

in cui $A_{1,1} \in C^{k \times k}$, $A_{2,2} \in C^{(n-k)\times(n-k)}$. Se A non è riducibile si dice che A è irriducibile.

2 Matrici simmetriche definite positive: il metodo del gradiente coniugato

Il metodo del gradiente coniugato (di cui forniremo solo il codice e alcuni brevi indicazioni) fu descritto nel 1952 da Hestenes e Stiefel ma per quanto destasse subito l'interesse dell'ambiente matematico non venne molto utilizzato fino al 1971, quando Reid suggerì il suo utilizzo per la risoluzione di sistemi sparsi (cioè con molte componenti nulle) di grandi dimensioni [2].

Se A è una matrice simmetrica e definita positiva di ordine n, si può dimostrare che il metodo è convergente e fornisce in aritmetica esatta la soluzione del sistema $Ax = b$ in al massimo n iterazioni. Questo teorema tradisce un po' le attese, sia perchè in generale i calcoli non sono compiuti in aritmetica esatta, sia perchè in molti casi della modellistica matematica n risulta essere molto alto. Comunque si può dimostrare [2, p. 279] in queste ipotesi che se

$$\|x\|_A = \sqrt{x^T A x}$$

e

$$e_k = x^* - x^{(k)}$$

allora

$$\|e_k\|_A \leq \left(\frac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1}\right)^{2k} \|e_0\|_A.$$

Questo risultato stabilisce che la convergenza del gradiente coniugato è lenta qualora si abbiano alti numeri di condizionamento

$$K_2(A) := \|A\|_2 \|A^{-1}\|_2 = \frac{\max_i |\lambda_i|}{\min_j |\lambda_j|}$$

(ove al solito $\{\lambda_i\}$ sono gli autovalori di A). Esistono varie versioni di questa disuguaglianza. Ad esempio in [7, p. 151]:

$$\|e_k\|_A \leq \left(\frac{2c^k}{1 + 2c^k}\right) \|e_0\|_A$$

dove
$$c := \frac{\sqrt{K_2(A)} - 1}{\sqrt{K_2(A)} + 1}.$$

L'analisi del metodo è piuttosto complessa. Qualora interessati si confronti con [1, p. 562-569], [2, p. 272-283], [3, p. 340-356], [7, p. 145-153].

Per quanto riguarda il codice del Gradiente Coniugato, un esempio è il file `cg.m` tratto da Netlib [6]:

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

%  -- Iterative template routine --
%     Univ. of Tennessee and Oak Ridge National Laboratory
%     October 1, 1993
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, Berry, Chan, Demmel, Donato, Dongarra,
%     Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications,
%     1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%
%  [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
%
% cg.m solves the symmetric positive definite linear system Ax=b
% using the Conjugate Gradient method with preconditioning.
%
% input   A        REAL symmetric positive definite matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         M        REAL preconditioner matrix
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
%
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it

  flag = 0;                                 % initialization
  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

  for iter = 1:max_it                       % begin iteration
```

```
    z  = M \ r;
    rho = (r'*z);

    if ( iter > 1 ),                       % direction vector
       beta = rho / rho_1;
       p = z + beta*p;
    else
       p = z;
    end

    q = A*p;
    alpha = rho / (p'*q );
    x = x + alpha * p;                    % update approximation vector

    r = r - alpha*q;                      % compute residual
    error = norm( r ) / bnrm2;            % check convergence
    if ( error <= tol ), break, end

    rho_1 = rho;

  end

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

% END cg.m
```

Osserviamo che il procedimento itera finchè un numero massimo di iterazioni è raggiunto oppure la norma 2 del residuo (relativo)

$$\frac{\|b - Ax^{(k)}\|_2}{\|b\|_2}$$

immagazzinata nella variabile `error` risulta inferiore ad una tolleranza prefissata `tol`. In questo caso il criterio d'arresto del metodo del gradiente coniugato è diverso da quello dello step relativo utilizzato nelle precedenti versioni di `Jacobi` ed `SOR`.

3 Un esperimento numerico

Consideriamo il sistema lineare $Ax = b$ dove A è la matrice tridiagonale a blocchi (di Poisson)

$$A = \begin{pmatrix} B & -I & 0 & \dots & 0 \\ -I & B & -I & \dots & 0 \\ 0 & -I & B & \dots & \dots \\ 0 & \dots & \dots & \dots & -I \\ 0 & 0 & \dots & -I & B \end{pmatrix}$$

con

$$B = \begin{pmatrix} 4 & -1 & 0 & \dots & 0 \\ -1 & 4 & -1 & \dots & 0 \\ 0 & -1 & 4 & \dots & \dots \\ 0 & \dots & \dots & \dots & -1 \\ 0 & 0 & \dots & -1 & 4 \end{pmatrix}$$

La matrice A è facilmente esprimibile utilizzando la funzione `makefish` scaricabile in [6]

```
function mat = makefish(siz);
% make a Poisson matrix

leng = siz*siz;
dia = zeros(siz,siz);
off = -eye(siz,siz);
 for i=1:siz, dia(i,i)=4; end;
 for i=1:siz-1, dia(i,i+1)=-1; dia(i+1,i)=-1; end;
mat = zeros(leng,leng);
 for ib=1:siz,
  mat(1+(ib-1)*siz:ib*siz,1+(ib-1)*siz:ib*siz) = dia; end;
 for ib=1:siz-1,
  mat(1+(ib-1)*siz:ib*siz,1+ib*siz:(ib+1)*siz) = off;
  mat(1+ib*siz:(ib+1)*siz,1+(ib-1)*siz:ib*siz) = off; end;
return;
```

Vediamo un esempio:

```
>> makefish(3)

ans =

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

>>
```

che evidentemente è una matrice di Poisson con B matrice quadrata di ordine 3

```
B =   4    -1     0
     -1     4    -1
      0    -1     4
```

Per ulteriori dettagli sulle origini della matrice di Poisson, si considerino ad esempio [1, p. 557], [2, p. 283], [3, p. 334]. Le matrici di Poisson sono evidentemente simmetriche, tridiagonali a blocchi, diagonalmente dominanti e dal primo e dal secondo teorema di Gerschgorin [2, p. 76-80], [3, p. 955] si può provare che sono non singolari. In particolare si può mostrare che A è definita positiva. Per accertarsene, calcoliamo il minimo autovalore della matrice di Poisson con $B \in \mathcal{M}_5$, semplicemente digitando sulla shell di Matlab-Octave

```
>> A=makefish(5);
>> m=min(eig(A))
m =
    0.5359
>>
```

Tale matrice di Poisson non è malcondizionata essendo

```
>> A=makefish(5);
>> cond(A)
ans =
   13.9282
>>
```

Poniamo ora

```
b=ones(size(A,1),1);
```

e risolviamo il sistema $Ax = b$ digitando

```
x_sol=A\b;
```

Nota la soluzione esatta confrontiamo i vari metodi risolvendo il sistema lineare con un numero massimo di iterazioni `maxit` e una tolleranza `tol` come segue

```
maxit=200; tol=10^(-8);
```

A tal proposito consideriamo l'm-file

```
demo_algebra_lineare.m
```

contenente il codice

```
maxit=200; tol=10^(-8);

siz=5;
A = makefish(siz);      % MATRICE DI POISSON.
b=ones(size(A,1),1);    % TERMINE NOTO.

x_sol=A\b;              % SOLUZIONE ESATTA. METODO LU.

norm_x_sol=norm(x_sol);
if norm(x_sol) == 0
```

```
    norm_x_sol=1;
end

x=zeros(size(b));      % VALORE INIZIALE.

% JACOBI.
[x_j, error_j, iter_j, flag_j]  = jacobi(A, x, b, maxit, tol);

fprintf('\t \n [JACOBI  ] [STEP REL., NORMA 2]: %2.2e [REL.ERR.]:
%2.2e',error_j,norm(x_j-x_sol)/norm_x_sol);
fprintf('\t \n            [ITER.]: %3.0f [FLAG]: %1.0f \n',iter_j,flag_j);


% GAUSS-SEIDEL.
w=1;
[x_gs, error_gs, iter_gs, flag_gs]  = sor(A, x, b, w, maxit, tol);

fprintf('\t \n [GAU.SEI.] [STEP REL., NORMA 2]: %2.2e [REL.ERR.]:
%2.2e',error_gs,norm(x_gs-x_sol)/norm_x_sol);
fprintf('\t \n            [ITER.]: %3.0f [FLAG]: %1.0f
\n',iter_gs,flag_gs);

% SOR.
w_vett=0.8:0.025:2;

for index=1:length(w_vett)
    w=w_vett(index);
   [x_sor, error_sor(index), iter_sor(index), flag_sor(index)]  = sor(A,
x, b, w, maxit, tol);
   relerr(index)=norm(x_sor-x_sol)/norm_x_sol;
end

[min_iter_sor, min_index]=min(iter_sor);

fprintf('\t \n [SOR OTT.] [STEP REL., NORMA 2]: %2.2e [REL.ERR.]:
%2.2e',error_sor(min_index),relerr(min_index));
fprintf('\t \n            [ITER.]: %3.0f [FLAG]: %1.0f [w]: %2.3f
\n',min_iter_sor,flag_sor(min_index),w_vett(min_index));

plot(w_vett,iter_sor,'r-');

% GRADIENTE CONIUGATO.
M=eye(size(A));
[x_gc, error_gc, iter_gc, flag_gc] = cg(A, x, b, M, maxit, tol);

fprintf('\t \n [GRA.CON.] [STEP REL., NORMA 2]: %2.2e [REL.ERR.]:
%2.2e',error_gc,norm(x_gc-x_sol)/norm_x_sol);
fprintf('\t \n            [ITER.]: %3.0f [FLAG]: %1.0f
\n',iter_gc,flag_gc);
```

Lanciamo la demo nella shell di Matlab-Octave e otteniamo

```
>> demo_algebra_lineare

 [JACOBI   ] [STEP REL., NORMA 2]: 8.73e-009 [REL.ERR.]: 5.65e-008
             [ITER.]: 116 [FLAG]: 0

 [GAU.SEI.] [STEP REL., NORMA 2]: 9.22e-009 [REL.ERR.]: 2.76e-008
             [ITER.]:  61 [FLAG]: 0

 [SOR OTT.] [STEP REL., NORMA 2]: 2.31e-009 [REL.ERR.]: 1.10e-009
             [ITER.]:  21 [FLAG]: 0 [w]: 1.350

 [GRA.CON.] [STEP REL., NORMA 2]: 4.41e-017 [REL.ERR.]: 2.21e-016
             [ITER.]:   5 [FLAG]: 0
>>
```

Una breve analisi ci dice che

1. Come previsto dalla teoria, il metodo di Gauss-Seidel converge in approssimativamente metà iterazioni di Jacobi;

2. Il metodo SOR ha quale costante quasi ottimale $w = 1.350$;

3. Il metodo del gradiente coniugato converge in meno iterazioni rispetto agli altri metodi (solo 5 iterazioni, ma si osservi il test d'arresto differente). Essendo la matrice di Poisson di ordine 25, in effetti ciò accade in meno di 25 iterazioni come previsto. Vediamo cosa succede dopo 25 iterazioni:

   ```
   >> maxit=25; tol=0;
   >> siz=5; A = makefish(siz); b=ones(size(A,1),1);
   >> [x_gc, error_gc, iter_gc, flag_gc] = cg(A, x, b, M, maxit, tol);
   >> error_gc
   error_gc =
     3.6287e-039
   >>
   ```

   Il residuo relativo, seppur non nullo è molto piccolo.

Un punto delicato riguarda la scelta del parametro ω ottimale (cioè minimizzante il raggio spettrale di SOR). Sia questo valore uguale a ω^*. Nel nostro codice abbiamo calcolato per forza bruta ω^+, tra i numeri reali $\omega^+ \leq 2$ del tipo $w_j = 0.8 + j \cdot 0.025$ quello per cui venivano compiute meno iterazioni.

E' possibile calcolare ω^* matematicamente? Nel caso della matrice di Poisson la risposta è affermativa. Da [3, Teor.5.10, p.333]

$$\omega^* = \frac{2}{1 + \sqrt{1 - \rho^2(B_J)}}$$

e il raggio spettrale della matrice di iterazione vale $\omega^* - 1$. dove $\rho(S)$ è il massimo degli autovalori in modulo della matrice S (il cosidetto raggio spettrale) e B_J la matrice di iterazione di Jacobi. Vediamo di calcolare questo valore nel caso della sopracitata matrice di Poisson. Dalla teoria, con ovvie notazioni,

$$B_J = I - D^{-1}A$$

e quindi

```
>> format long;
>> D=diag(diag(A));
>> BJ=eye(size(A))-inv(D)*A;
>> s=eig(BJ);
>> s_abs=abs(s);
>> rho=max(s_abs);
>> w=2/(1+sqrt(1-rho^2))
w =
    1.33333333333333
>> maxit=50; tol=10^(-8);
>> b=ones(size(A,1),1);
>> [x_sor, error_sor, iter_sor, flag_sor]  = sor(A, x, b, w, maxit, tol);
>> iter_sor
iter_sor =
    22
>>
```

Si rimane un po' sorpresi dal fatto che per $w = 1.350$ il numero di iterazioni fosse inferiore di quello fornito dal valore ottimale teorico $w^* = 1.333\ldots$. Il fatto è che questo è ottenuto cercando di massimizzare la velocità asintotica di convergenza. Purtroppo questo minimizza una stima del numero di iterazioni k minime da compiere e non quello effettivo.

Abbiamo detto che un punto chiave è la grandezza del raggio spettrale delle matrici di iterazione e che è desiderabile che questo numero oltre ad essere strettamente minore di uno sia il più piccolo possibile. Vediamo i raggi spettrali dei metodi esposti.

Salviamo in `raggispettrali.m` il seguente programma principale

```
maxit=50; tol=0;

siz=5;
A = makefish(siz);      % MATRICE DI POISSON.
b=ones(size(A,1),1);    % TERMINE NOTO.

[ M, N ] = split( A , b, 1.0, 1 ); % JACOBI.
P=inv(M)*N;
rho_J=max(abs(eig(P)));
fprintf('\n \t [RAGGIO SPETTRALE][JACOBI]: %2.15f',rho_J);

[ M, N, b ] = split( A, b, 1, 2 ); % GS.
```

```
P=inv(M)*N;
rho_gs=max(abs(eig(P)));
fprintf('\n \t [RAGGIO SPETTRALE][GAUSS-SEIDEL]: %2.15f',rho_gs);


D=diag(diag(A));
E=-(tril(A)-D);
F=-(triu(A)-D);
w=1.350;
M=D/w-E; N=(1/w-1)*D+F;
P=inv(M)*N;
rho_sor=max(abs(eig(P)));
fprintf('\n \t [RAGGIO SPETTRALE][SOR BEST]: %2.15f',rho_sor);

w=1.33333333333333;
[ M, N, b ] = split( A, b, w, 2 ); % SOR OPT.
M=D/w-E; N=(1/w-1)*D+F;
P=inv(M)*N;
rho_sor_opt=max(abs(eig(P)));
fprintf('\n \t [RAGGIO SPETTRALE][SOR OPT]: %2.15f',rho_sor_opt);
```

Di seguito:

```
>> raggispettrali
          [RAGGIO SPETTRALE][JACOBI]: 0.866025403784438
          [RAGGIO SPETTRALE][GAUSS-SEIDEL]: 0.750000000000000
          [RAGGIO SPETTRALE][SOR BEST]: 0.350000000000001
          [RAGGIO SPETTRALE][SOR OPT]: 0.333333380707781
>>
```

Il valore del raggio spettrale della matrice di iterazione del metodo SOR per parametro ottimale, per quanto visto anticipatamente vale $\omega^* - 1$, e l'esperimento numerico lo conferma.
Abbiamo poi osservato che in questo caso la velocità di convergenza del metodo di Gauss-Seidel è il doppio di quella di Jacobi. Poste B_{GS}, B_J le rispettive matrici di iterazione, e detta R la velocità di convergenza, osserviamo che da

$$\begin{aligned} R(B_J) &:= -\ln(\rho(B_J)) && (11)\\ R(B_{GS}) &:= -\ln(\rho(B_{GS})) && (12)\\ R(B_{GS}) &:= 2R(B_J) && (13) \end{aligned}$$

si ha

$$-\ln(\rho(B_{GS})) = R(B_{GS}) = 2R(B_J) = -2\ln(\rho(B_J)) = -\ln(\rho(B_J))^2$$

da cui essendo il logaritmo una funzione invertibile

$$\rho(B_{GS}) = (\rho(B_J))^2.$$

Il raggio spettrale della matrice di iterazione di Gauss-Seidel coincide quindi col quadrato di quella di Jacobi ed infatti come è facile verificare

```
>> 0.866025403784438^2
ans =
   0.75000000000000
>>
```

Al momento non consideriamo il metodo del gradiente coniugato poichè non è di tipo stazionario.

4 Facoltativo: Il metodo di Richardson

Come visto nella precedente lezione i metodi di Jacobi e di Gauss-Seidel e le loro versioni *rilassate* sono metodi iterativi del tipo

$$Mx^{(k+1)} = Nx^{(k)} + b, \tag{14}$$

per opportune scelte della matrici M, N tali che

$$A = M - N. \tag{15}$$

Se

$$r^{(k)} = b - Ax^{(k)} \tag{16}$$

è il *residuo* alla k-sima iterazione allora da (14) e (15)

$$M(x^{(k+1)} - x^{(k)}) = Nx^{(k)} + b - Mx^{(k)} = b - Ax^{(k)} = r^{(k)} \tag{17}$$

Per un opportuno parametro di accelerazione $\alpha > 0$ (da non confondersi con quello di SOR), si può fornire un'ovvia generalizzazione del metodo (17)

$$M(x^{(k+1)} - x^{(k)}) = \alpha\, r^{(k)},\ k \geq 0. \tag{18}$$

Evidentemente (17) corrisponde alla scelta $\alpha = 1$.

Il parametro $\alpha > 0$ viene scelto cosí da minimizzare il raggio spettrale della matrice di iterazione. In questo caso si vede che da

$$M(x^{(k+1)} - x^{(k)}) = \alpha\,(b - Ax^{(k)}) \tag{19}$$

necessariamente

$$Mx^{(k+1)} = Mx^{(k)} + \alpha\,(b - Ax^{(k)}) = (M - \alpha A)x^{(k)} + \alpha\, b, \tag{20}$$

e quindi la matrice di iterazione diventa

$$R_\alpha = M^{-1}(M - \alpha\, A) = I - \alpha\, M^{-1} A. \tag{21}$$

Se $M^{-1}A$ è definita positiva e λ_{min} e λ_{max} sono rispettivamente il minimo e massimo autovalore di $M^{-1}A$, allora il valore ottimale del parametro α è

$$\alpha_{\text{ott}} = \frac{2}{\lambda_{min} + \lambda_{max}} \tag{22}$$

ed in corrispondenza si ha che la matrice di iterazione $R_{\alpha_{\text{ott}}}$ ha raggio spettrale

$$\alpha_{\text{ott}} = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{min} + \lambda_{max}} \tag{23}$$

Si osservi che la scelta di α non dipende dall'iterazione; di conseguenza (18) definisce il cosidetto *metodo di Richardson stazionario*, per distinguerlo dal *metodo di Richardson non stazionario*

$$M(x^{(k+1)} - x^{(k)}) = \alpha_k (b - Ax^{(k)}). \tag{24}$$

con α_k che non è necessariamente costante.

Se $M^{-1}A$ é definita positiva, una classica scelta di α_k è

$$\alpha_k = \frac{{r^{(k)}}^T z^{(k)}}{{z^{(k)}}^T A z^{(k)}}. \tag{25}$$

5 Facoltativo: Esercizi sui metodi di Richardson

Il metodo di Richardson si puó implementare come segue:

1. assegnato $x^{(0)}$, si ponga $r^{(0)} = b - Ax^{(0)}$.
2. si considera per $k \geq 0$ lo schema:

$$\begin{cases} Mz^{(k)} = r^{(k)}; \\ x^{(k+1)} = x^{(k)} + \alpha z^{(k)}; \\ r^{(k+1)} = r^{(k)} - \alpha A z^{(k)}; \end{cases} \tag{26}$$

Il metodo di Richardson con parametro α_k definito da (25) è una semplice variante di (26) in cui α_k viene assegnato subito dopo aver calcolato $z^{(k)}$.

1. Implementare in Matlab le due routines

   `richardson`

   e

   `richardsonmodificato`

   definito da (25).

2. Dato il problema $Ax = b$ definito dallo schema alle differenze utilizzato nella lezione precedente per risolvere un problema di Poisson, definire i metodi di Richardson associati ai metodi di Jacobi e Gauss-Seidel. Applicare i rispettivi metodi di Richardson ottimali e osservare sia i (possibili) miglioramenti in termini di errore dopo un numero prefissato di iterazioni sia il miglior raggio spettrale della matrice di iterazione. Utilizzare il residuo assoluto in norma 2

$$r^{(k)} := \|b - Ax^{(k)}\|_2 < \text{toll} \tag{27}$$

quale test di arresto. Implementare in alternativa una versione basata sul residuo relativo in norma 2

$$r_{\text{rel}}^{(k)} := \|b - Ax^{(k)}\|_2 / \|b\|_2 < \text{toll}, \tag{28}$$

o lo step relativo in norma 2

$$\Delta_{\text{rel}}^{(k)} := \|x^{(k+1)} - x^{(k)}\|_2 / \|b\|_2 < \text{toll}. \tag{29}$$

3. Data la matrice di Hilbert di ordine 12 (che denotiamo con H_{12}), calcolarne gli autovettori. E' questa matrice definita positiva? Ricordiamo che H_{12} puó essere generata dal comando Matlab `hilb(12)`. Usare la routine Matlab `cond` per calcolarne il numero di condizionamento $K_2(A)$ in norma 2, cioé la quantitá
$$K_2(A) = \|A\|_2 \|A^{-1}\|_2. \tag{30}$$
E' una matrice definita positiva? Si puó affermare che i metodi di Jacobi e Gauss-Seidel sono convergenti? Applicare i metodi Jacobi, Gauss-Seidel e le rispettive versioni ottimali per risolvere il sistema $H_{12}x = b$ con $b_i = \int_0^1 x^{i-1}$ per $i = 1, \ldots, 12$. Si osservi l'errore compiuto sapendo che la soluzione del problema e' il primo vettore della base canonica di R^n (per convincersene scrivere la riga di comando Matlab $x = A\backslash b$ ove $A = H_{12}$). Eseguire lo stesso esercizio per H_{20} e $b \in R^{20}$ definita come sopra.

4. Si consideri la matrice a blocchi
$$A = \begin{pmatrix} B & -I & 0 \\ -I & B & -I \\ 0 & -I & B \end{pmatrix} \tag{31}$$
dove I é la matrice identica e
$$B = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & 0 \\ 0 & -1 & 4 \end{pmatrix} \tag{32}$$
Applicare i metodi di Jacobi, Gauss-Seidel e i rispettivi metodi di Richardson (nella versione stazionaria a parametro ottimale, e non stazionaria con parametro α_k definito in (25)) per calcolare la soluzione del problema $Ax = b$, dove $b = (1, 0, \ldots, 0) \in R^9$. E' possibile stabilire a priori che i metodi di Jacobi e Gauss-Seidel sono convergenti? Calcolare il raggio spettrale delle matrici di iterazione di tali metodi utilizzando il comando `eig` di Matlab . Fissata una tolleranza di tol= 10^{-8}, arrestare le iterazioni quando il residuo relativo in norma 2 é inferiore di*tol*. Quale di questi metodi converge in meno iterazioni? Questo risultato è coerente con il valore del raggio spettrale delle matrici di iterazione?

6 Facoltativo: Altre matrici interessanti. La matrice di Hilbert.

Per vedere alcuni comandi di base aiutiamoci con delle matrici predefinite in Matlab/Octave. Digitiamo nella shell di Matlab/Octave >> `help elmat`. In Matlab 6.5 abbiamo

```
>> help elmat

  Elementary matrices and matrix manipulation.

  Elementary matrices.
    zeros       - Zeros array.
    ones        - Ones array.
    eye         - Identity matrix.
    repmat      - Replicate and tile array.
    rand        - Uniformly distributed random numbers.
    randn       - Normally distributed random numbers.
    linspace    - Linearly spaced vector.
    logspace    - Logarithmically spaced vector.
    freqspace   - Frequency spacing for frequency response.
    meshgrid    - X and Y arrays for 3-D plots.
    :           - Regularly spaced vector and index into matrix.

  ...

  Specialized matrices.
    compan      - Companion matrix.
    gallery     - Higham test matrices.
    hadamard    - Hadamard matrix.
    hankel      - Hankel matrix.
    hilb        - Hilbert matrix.
    invhilb     - Inverse Hilbert matrix.
    magic       - Magic square.
    pascal      - Pascal matrix.
    rosser      - Classic symmetric eigenvalue test problem.
    toeplitz    - Toeplitz matrix.
    vander      - Vandermonde matrix.
    wilkinson   - Wilkinson's eigenvalue test matrix.
```

Questo ci dice che Matlab ha predefinito un set di matrici di particolare interesse. Se possibile si suggerisce di provare i metodi che andremo ad introdurre con una matrice facente parte della `gallery` di Matlab. Ciò non appare possibile nelle recenti releases di Octave, come GNU Octave 2.1.73. Da Matlab 6.5

```
>> help gallery

 GALLERY Higham test matrices.
    [out1,out2,...] = GALLERY(matname, param1, param2, ...)
```

```
    takes matname, a string that is the name of a matrix family, and
    the family's input parameters. See the listing below for available
    matrix families. Most of the functions take an input argument
    that specifies the order of the matrix, and unless otherwise
    stated, return a single output.
    For additional information, type "help private/matname", where matname
    is the name of the matrix family.

    cauchy   Cauchy matrix.
    chebspec Chebyshev spectral differentiation matrix.
    chebvand Vandermonde-like matrix for the Chebyshev polynomials.
    chow     Chow matrix -- a singular Toeplitz lower Hessenberg matrix.
    circul   Circulant matrix.

    ...

    poisson  Block tridiagonal matrix from Poisson's equation (sparse).
    prolate  Prolate matrix -- symmetric, ill-conditioned Toeplitz matrix.
    randcolu Random matrix with normalized cols and specified singular
values.
    randcorr Random correlation matrix with specified eigenvalues.
    randhess Random, orthogonal upper Hessenberg matrix.
    rando    Random matrix with elements -1, 0 or 1.
    randsvd  Random matrix with pre-assigned singular values and specified
             bandwidth.
    redheff  Matrix of 0s and 1s of Redheffer.
    riemann  Matrix associated with the Riemann hypothesis.
    ris      Ris matrix -- a symmetric Hankel matrix.
    smoke    Smoke matrix -- complex, with a "smoke ring" pseudospectrum.
    toeppd   Symmetric positive definite Toeplitz matrix.
    toeppen  Pentadiagonal Toeplitz matrix (sparse).
    tridiag  Tridiagonal matrix (sparse).
    triw     Upper triangular matrix discussed by Wilkinson and others.
    wathen   Wathen matrix -- a finite element matrix (sparse, random
entries).
    wilk     Various specific matrices devised/discussed by Wilkinson.
             (Two output arguments)

    GALLERY(3) is a badly conditioned 3-by-3 matrix.
    GALLERY(5) is an interesting eigenvalue problem.  Try to find
    its EXACT eigenvalues and eigenvectors.

    See also MAGIC, HILB, INVHILB, HADAMARD, WILKINSON, ROSSER, VANDER.
```

7 Facoltativo: Altre matrici interessanti. La matrice di Hilbert.

Rivediamo gli esperimenti in una recente release di Octave, come GNU Octave 2.1.73.

```
octave:12> makefish(3)
ans =

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

octave:13> A=makefish(5);
octave:14> m=min(eig(A))
m = 0.53590
octave:15> cond(A)
ans = 13.928
octave:16> b=ones(size(A,1),1);
octave:17> demo_algebra_lineare


 [JACOBI  ] [STEP REL., NORMA 2]: 8.73e-09 [REL.ERR.]: 5.65e-08
            [ITER.]: 116 [FLAG]: 0

 [GAU.SEI.] [STEP REL., NORMA 2]: 9.22e-09 [REL.ERR.]: 2.76e-08
            [ITER.]:  61 [FLAG]: 0

 [SOR OTT.] [STEP REL., NORMA 2]: 2.31e-09 [REL.ERR.]: 1.10e-09
            [ITER.]:  21 [FLAG]: 0 [w]: 1.350

 [GRA.CON.] [STEP REL., NORMA 2]: 4.67e-17 [REL.ERR.]: 1.85e-16
            [ITER.]:   5 [FLAG]: 0
octave:18> format long;
octave:19> D=diag(diag(A));
octave:20> size(D)
ans =

  25  25
octave:21> BJ=eye(size(A))-inv(D)*A;
octave:22> s=eig(BJ);
octave:23> s_abs=abs(s);
octave:24> rho=max(s_abs);
octave:25> w=2/(1+sqrt(1-rho^2))
```

```
w = 1.33333333333333
octave:26> maxit=50; tol=10^(-8);
octave:27> b=ones(size(A,1),1);
octave:28> [x_sor,error_sor,iter_sor,flag_sor]=sor(A,x,b,w,maxit,tol);
octave:29> iter_sor
iter_sor = 22
octave:30> raggispettrali

         [RAGGIO SPETTRALE][JACOBI]: 0.866025403784439
         [RAGGIO SPETTRALE][GAUSS-SEIDEL]: 0.750000000000000
         [RAGGIO SPETTRALE][SOR BEST]: 0.350000000000000
         [RAGGIO SPETTRALE][SOR OPT]: 0.333333380472264
octave:31>  0.866025403784439^2
ans = 0.750000000000001
octave:32>
```

References

[1] K. Atkinson, *Introduction to Numerical Analysis*, Wiley, 1989.

[2] D. Bini, M. Capovani e O. Menchi, *Metodi numerici per l'algebra lineare*, Zanichelli, 1988.

[3] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.

[4] S.D. Conte e C. de Boor, *Elementary Numerical Analysis, 3rd Edition*, Mc Graw-Hill, 1980.

[5] The MathWorks Inc., *Numerical Computing with Matlab*, `http://www.mathworks.com/moler`.

[6] Netlib, `http://www.netlib.org/templates/matlab/`.

[7] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.

[8] A. Suli e D. Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.