Calcolo soluzioni di un sistema lineare con metodo iterativo di Jacobi.

```
x = Jacobi(A,b);
x = Jacobi(A,b, TOL);
x = Jacobi(A,b,TOL,MAXITER);
[x, niter] = Jacobi(A,b,TOL,MAXITER);
[x, niter, resrel] = Jacobi(A,b,TOL,MAXITER);
```

Argomenti di input

A - Matrice dei coefficienti

Matrice dei coefficienti del sistema `Ax = b`. Il numero di righe di A dev'essere uguale al numero di elementi di b. A deve essere di tipo `sparse`.

Tipo: `sparse | double`

b - Array dei termini noti

Array dei termini noti del sistema `Ax = b`. Può essere inserito sia come vettore riga, sia come vettore colonna.

Tipo: `single | double`

TOL - Tolleranza richiesta

Il parametro facoltativo `TOL` è del tipo 10^{-t} dove t è il numero di cifre esatte richieste dall'utente. Se non specificato viene impostata la tolleranza di default 10^{-6}.

Esempio: 10^(-12)

Tipo: `single | double`

MAXITER - Numero massimo di iterazioni

Il parametro facoltativo MAXITER indica il numero massimo di iterazioni, se omesso viene impostato al valore di default 500.

Esempio: `20`

Tipo: `single | double`

Argomenti di output

x - Array delle soluzioni

`x` è il vettore contenente le n soluzioni del sistema `Ax = b`.

Tipo: `single | double`

niter - Numero di iterazioni

`niter` è il numero di iterazioni eseguite per calcolare la soluzione `x`.

Tipo: `single | double`

resrel - Residuo relativo

`resrel` è il residuo relativo calcolato utilizzando la formula $\frac{||\, b - A * x \,||}{||\, b \,||}$.

Tipo: `single | double`

Algoritmo

Per la risoluzione di un sistema lineare `Ax = b`, con `A` matrice di tipo *sparse*, l'algoritmo implementa il metodo iterativo di Jacobi.

Correlati

bicg

```
function [x, niter, resrel] = Jacobi(A, b, TOL, MAXITER)

narginchk(2,4);

% controlla ingressi vuoti
if(isempty(A) || isempty(b))
   error("Matrice A o vettore b vuoti.");
end

% controlla se la matrice A è di tipo sparse
if(issparse(A)==0)
    error("La matrice A non è tipo sparse.");
end

[n,m] = size(A);
```

```
% controlla se A è una matrice quadrata.
if(m~=n)
   error("Matrice A non quadrata.");
end

if(nnz(diag(A)) ~= n)
    error("La matrice A ha elementi nulli sulla diagonale. Riordinare la matrice e ries
end

% controlla se b è un vettore.
if(~isvector(b) || isscalar(b) || isa(b,'cell') || isa(b,'table') || isa(b,'struct'))
    error("b non è un vettore.");
end

% controlla se b è un vettore numeric
if(~isnumeric(b))
    error("Il vettore b non è vettore numerico.");
end

I=length(b);

% controlla la lunghezza di b.
if(I~=n)
    error("Il vettore dei termini noti non rispetta le dimensioni di A.");
end

% controlli parametri facoltativi
if(nargin == 2)

    TOL = 10^(-6);
    MAXITER = 500;

elseif(nargin == 3)

    MAXITER = 500;

    if ( (isnumeric(TOL) == 0) || (isreal(TOL) == 0) || (all(isfinite(TOL)) == 0) || (
        TOL = 10^(-6);
        warning("TOL scorretto. Impostato a 10^(-6)");
    end

elseif(nargin == 4)

    if ( (isnumeric(TOL) == 0) || (isreal(TOL) == 0) || (all(isfinite(TOL)) == 0) || (
        TOL = 10^(-6);
        warning("TOL scorretto. Impostato a 10^(-6)");
    end

    % controllo che MAXITER sia reale e finito
    if ( (isnumeric(MAXITER) == 0) || (isreal(MAXITER) == 0) || (all(isfinite(MAXITER)
        MAXITER = 500;
        warning("Numero massimo di iterazioni scorretto. Impostato a 500.");
    end
```

```
    MAXITER = floor(MAXITER);

end

x0 = zeros(n,1);
niter = 0;

c = (b./diag(A));
B = (-1./diag(A)')'.*(A-diag(diag(A)));
x = B*x0 + c;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% LE SEGUENTI LINEE DI CODICE SONO UTILIZZATE SOLO NEL CONFRONTO   %%%%
%%%%%%%% TEMPORALE NEL PARAGRAFO TEMPI DI ESECUZIONE NEL LIVE SCRIPT      %%%%
%%%%%%%% 'Livescript_Jacobi.mlx'                                         %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %%%%
%   P = diag(diag(A));                                                     %%%%
%   N = P - A;                                                             %%%%
%   invP = inv(P);                                                         %%%%
%   B = eye(n,n) - invP*A;                                                 %%%%
%   c = invP*b;                                                            %%%%
%   x = B*x0 + c;                                                          %%%%
%                                                                          %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TOLR = max(TOL*norm(x, "inf"), realmin);

while(norm(x - x0,"inf") >= TOLR && niter < MAXITER )
    x0 = x;
    x = B*x0 + c;
    TOLR = max(TOL*norm(x, "inf"), realmin);
    niter = niter + 1;
end

if(niter == MAXITER)
    warning("Attenzione! È stato raggiunto il numero massimo di iterazioni. Il risulta
end

resrel = norm(b-A*x,"inf")/norm(b,"inf");

end
```