Calcolo Numerico

(A.A. 2012-2013)

Esercitazioni

Lezioni n. 12,13

Metodi iterativi per sistemi lineari

14, 16-05-2013

Sistemi lineari - Metodo iterativi

$$A\,X = B \Leftrightarrow X = C\,X + Q$$

Se $\overline{X} \in \mathbf{R}^n$ è **soluzione** di $A\,X = B$ allora è **punto unito** di $\Phi = C\,X + Q$:

$$\boxed{A\,\overline{X} = B \Leftrightarrow \overline{X} = C\,\overline{X} + Q}$$

Il **punto unito** $\overline{X} = [\overline{x}_1, \overline{x}_2, ..., \overline{x}_n]^T$, può essere **approssimato** generando la successione

$$\begin{cases} X^{(k)} = \mathbf{C}\,X^{(k-1)} + Q & k = 1, 2, \ldots \\ X^{(0)} \in \mathbf{R}^n & \text{dato} \end{cases}$$

$$\Rightarrow \quad x_i^{(k)} = \sum_{j=1}^{n} c_{ij}\, x_j^{(k-1)} + q_i \qquad i = 1, \ldots, n$$

La matrice $\mathbf{C} \in \mathbf{R}^{n \times n}$ è chiamata **matrice di iterazione**.

Convergenza: $\lim_{k \to \infty} \|E^{(k)}\| = 0 \iff \lim_{k \to \infty} X^{(k)} = \overline{X}$

Costruzione di metodi iterativi

Splitting di A: $A = M + N$ dove M è una matrice **invertibile**.

$$AX = B \rightarrow (M+N)X = B \rightarrow MX = -NX + B \rightarrow$$

$$\rightarrow X = -M^{-1}NX + M^{-1}B \Rightarrow \boxed{C = -M^{-1}N \qquad Q = M^{-1}B}$$

Una possibile **decomposizione** di A è

$$\boxed{A = L + D + U}$$

Jacobi: $M = D,\ N = L + U \Rightarrow \begin{cases} C_J = -D^{-1}(L+U) \\ Q_J = D^{-1}B \end{cases}$

Gauss-Seidel: $M = D + L,\ N = U \Rightarrow \begin{cases} C_{GS} = -(D+L)^{-1}U \\ Q_{GS} = (D+L)^{-1}B \end{cases}$

Convergenza dei metodi di Jacobi e Gauss-Seidel

$$\begin{cases} \textbf{C.S.:} & \|C_J\|_1, \|C_J\|_\infty < 1 \quad \text{e} \quad \|C_{GS}\|_1, \|C_{GS}\|_\infty < 1 \\ \textbf{C.N.S.:} & \rho(C_J) < 1 \quad \text{e} \quad \rho(C_{GS}) < 1 \end{cases}$$

Attenzione: Le **condizioni di convergenza** per le matrici di iterazione C_J e C_{GS} vanno **verificate** di volta in volta.

- Per alcune matrici A potrebbe convergere **solo uno** dei due metodi.
- Se convergono entrambi i metodi, quello di **Gauss-Seidel** converge **più velocemente**.

Criterio d'arresto e velocità asintotica di convergenza

Se il metodo iterativo è **convergente**, si arresta il procedimento quando

$$\|X^{(k+1)} - X^{(k)}\| < \varepsilon \qquad \varepsilon\text{: } \textbf{tolleranza prefissata}$$

Stima a priori: il numero di iterazioni K necessario affinché $\|E^{(K)}\| < \varepsilon$, è dato da

$$K > \log\left(\frac{(1 - \|C\|)\,\varepsilon}{\|X^{(1)} - X^{(0)}\|}\right)\frac{1}{\log\|C\|}$$

L'errore si riduce di un fattore 10^{-m} all'iterazione

$$K \simeq -\frac{m}{\text{Log } \rho(C)}$$

Velocità asintotica di convergenza: $V = -\text{Log } \rho(C)$

Convergenza per matrici A con struttura speciale

Matrici diagonalmente dominanti:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad i = 1, 2, \ldots, n$$

(diagonale dominanza per righe)

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ji}| \quad i = 1, 2, \ldots, n$$

(diagonale dominanza per colonne)

Condizione sufficiente di convergenza:

Teorema. Se A è **diagonalmente dominante** per righe o per colonne, i **metodi di Jacobi** e **Gauss-Seidel** sono entrambi **convergenti** per qualunque scelta dell'approssimazione iniziale $X^{(0)}$.

Per esempio, se A è **diagonalmente dominante per righe** si ha

$$\|C_J\|_\infty = \max_{1 \leq i \leq n} \left| \sum_{\substack{j=1 \\ j \neq i}}^{n} -\frac{a_{ij}}{a_{ii}} \right| \leq \max_{1 \leq i \leq n} \left(\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \right) \frac{1}{|a_{ii}|} < 1$$

Matrici (simmetriche) definite positive:

Una matrice quadrata $A \in \mathbf{R}^{n\times n}$ **simmetrica** è **definita positiva** se

$$X^T A X = \sum_{i,j=1}^{n} a_{ij} x_i x_j > 0 \qquad \forall X \in \mathbf{R}^n$$

Per riconoscere se una matrice è definita positiva si può usare il **criterio di Sylvester**:

> Affinché una matrice $A \in \mathbf{R}^{n\times n}$ **simmetrica** sia **definita positiva**, è **necessario e sufficiente** che
>
> $$\det A_k > 0 \qquad k = 1, 2, \ldots, n,$$
>
> dove A_k sono le sottomatrici principali di testa di A.

Condizione sufficiente di convergenza:

> **Teorema.** Se A è (simmetrica) **definita positiva**, il **metodo di Gauss-Seidel** è **convergente** per qualunque scelta dell'approssimazione iniziale $X^{(0)}$.

Esercizio 1

Dato il sistema

$$\begin{cases} 10x_1 + x_2 + x_3 &= 2 \\ x_1 + \alpha x_2 + x_3 &= 1 \\ 2x_1 + x_2 + \beta x_3 &= 3 \end{cases}$$

dipendente dai parametri α e β. Stabilire per quali valori dei parametri α e β i metodi iterativi di **Jacobi** e **Gauss-Seidel** convergono sicuramente per ogni scelta del vettore iniziale $\mathbf{X}^{(0)}$.

Condizione sufficiente perchè un metodo iterativo converga rispetto ad una norma di matrici $\|\cdot\|$ è che la norma della matrice di iterazione C sia strettamente minore di 1, cioè $\|C\| < 1$.

Sia $A = \begin{pmatrix} 10 & 1 & 1 \\ 1 & \alpha & 1 \\ 2 & 1 & \beta \end{pmatrix}$ la matrice dei coefficienti del sistema,

$$L = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 1 & 0 \end{pmatrix}, \; D = \begin{pmatrix} 10 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & \beta \end{pmatrix} \quad \text{e} \quad U = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix},$$

allora la matrice di iterazione del metodo di Jacobi C_J è data da

$$C_J = -D^{-1}(L + U),$$

cioè

$$C_J = \begin{pmatrix} 0 & -\frac{1}{10} & -\frac{1}{10} \\ -\frac{1}{\alpha} & 0 & -\frac{1}{\alpha} \\ -\frac{2}{\beta} & -\frac{1}{\beta} & 0 \end{pmatrix}$$

con $\alpha \neq 0$ e $\beta \neq 0$.

Considerando la $\|\cdot\|_\infty$ risulta

$$\|C_J\|_\infty = \max\{\frac{2}{10}, \frac{2}{|\alpha|}, \frac{3}{|\beta|}\} < 1 \quad \Leftrightarrow \quad \frac{2}{|\alpha|} < 1 \quad \wedge \quad \frac{3}{|\beta|} < 1,$$

cioè

$$|\alpha| > 2 \quad \wedge \quad |\beta| > 3.$$

Si osserva che alla stessa conclusione si giunge imponendo che la matrice A del sistema sia a diagonale dominante per righe.

Considerando, invece, la $\|\cdot\|_1$ risulta

$$\|C_J\|_1 = \max\{\frac{1}{|\alpha|} + \frac{2}{|\beta|}, \frac{1}{10} + \frac{1}{|\beta|}, \frac{1}{10} + \frac{1}{|\alpha|}\} < 1$$

$$\Leftrightarrow \quad \begin{cases} \frac{1}{|\alpha|} + \frac{2}{|\beta|} < 1 \\ \frac{1}{10} + \frac{1}{|\beta|} < 1 \\ \frac{1}{10} + \frac{1}{|\alpha|} < 1 \end{cases}$$

$$\Leftrightarrow \quad \begin{cases} \frac{|\beta|+2|\alpha|}{|\alpha||\beta|} < 1 \\ |\beta| > \frac{10}{9} \\ |\alpha| > \frac{10}{9} \end{cases}$$

da cui $\frac{10}{3} < |\beta| + 2|\alpha| < |\alpha||\beta|$.

E' opportuno notare che in questo caso non si arriva alla stessa conclusione imponendo la dominanza diagonale per colonne.

Inoltre, scegliendo $\alpha = \frac{9}{4}$ e $\beta = \frac{13}{4}$ risulta $\|C_J\|_\infty < 1$ mentre $\|C_J\|_1 > 1$. Infatti,

$$\frac{13}{4} + 2\frac{9}{4} = \frac{31}{4} = 7.75 > \frac{9}{4}\frac{13}{4} = \frac{117}{16} \approx 7.3125.$$

In questo caso, per esempio, sono necessarie almeno

$$K = 184$$

iterazioni per ottenere un errore $\|E^{(K)}\|_\infty$ tra due approssimazioni successive inferiore a $0.5 \cdot 10^{-5}$ avendo scelto $\mathbf{X}^{(0)} = [0 \quad 0 \quad 0]^T$ come approssimazione iniziale.

Infatti risulta

$$\|E^{(K)}\|_\infty < \varepsilon$$

quando

$$K > \log\left(\frac{(1-\|C\|_\infty)\,\varepsilon}{\|X^{(1)}-X^{(0)}\|_\infty}\right)\frac{1}{\log\|C\|_\infty}$$

dove $\|C\|_\infty = \frac{12}{13}$, $\varepsilon = 0.5\cdot 10^{-5}$,

$$\|\mathbf{X}^{(1)}-\mathbf{X}^{(0)}\|_\infty = \|\mathbf{X}^{(1)}\|_\infty = \|\mathbf{Q_J}\|_\infty = \frac{12}{13}$$

e $\mathbf{Q} = D^{-1}B = [\frac{b_1}{a_{11}} \quad \frac{b_2}{a_{22}} \quad \frac{b_3}{a_{33}}]^T = [\frac{2}{10} \quad \frac{4}{9} \quad \frac{12}{13}]^T$,

con $\mathbf{B} = [b_1 \quad b_2 \quad b_3]^T = [2 \quad 1 \quad 3]^T$ vettore dei termini noti del sistema.

Viceversa, se si pone $\alpha = 4$ e $\beta = 3$, il metodo di Jacobi non soddisfa la condizione sufficiente rispetto alla norma $\|\cdot\|_\infty$ mentre converge sicuramente rispetto alla norma $\|\cdot\|_1$. Inoltre

$$\|E^{(K)}\|_1 < 0.5 \cdot 10^{-5}$$

se

$$K > 166.03$$

avendo scelto $\mathbf{X}^{(0)} = [0 \quad 0 \quad 0]^T$

ed essendo $\|X^{(1)}\|_1 = \| \begin{bmatrix} \frac{2}{10} & \frac{1}{4} & \frac{1}{3} \end{bmatrix}^T \|_1 = \frac{2}{10} + \frac{1}{4} + \frac{1}{3} = \frac{47}{60}$

e $\|C\|_1 = \frac{11}{12}$

La matrice di iterazione del metodo di Gauss-Seidel è data da

$$C_{GS} = -(L + D)^{-1}U,$$

con

$$(L + D)^{-1} = \begin{pmatrix} \frac{1}{10} & 0 & 0 \\ -\frac{1}{10\alpha} & \frac{1}{\alpha} & 0 \\ \frac{1}{10\beta}\left(\frac{1}{\alpha} - 2\right) & -\frac{1}{\alpha\beta} & \frac{1}{\beta} \end{pmatrix}.$$

Allora

$$C_{GS} = \begin{pmatrix} 0 & -\frac{1}{10} & -\frac{1}{10} \\ 0 & \frac{1}{10\alpha} & -\frac{9}{10\alpha} \\ 0 & -\frac{1}{10\alpha\beta} + \frac{1}{5\beta} & \frac{9}{10\alpha\beta} + \frac{1}{5\beta} \end{pmatrix}.$$

$$\|C_{GS}\|_\infty = \max\{\frac{2}{10}, \frac{1}{|\alpha|}, \frac{|2\alpha - 1| + |9 + 2\alpha|}{10|\alpha||\beta|}\} < 1$$

se

$$\begin{cases} |\alpha| > 1 \\ |2\alpha - 1| + |9 + 2\alpha| < 10|\alpha||\beta| \end{cases}$$

Si osserva che $\alpha = \frac{9}{4}$ e $\beta = \frac{13}{4}$ soddisfano la condizione precedente.

Quindi il metodo di Gauss-Seidel sicuramente converge per ogni scelta dell'approssimazione iniziale.

Scegliendo $\mathbf{X}^{(0)} = [0 \quad 0 \quad 0]^T$, sono necessarie

$$K = 16$$

affinchè $\|E^{(K)}\|_\infty < \varepsilon = 0.5 \cdot 10^{-5}$.

Infatti, $\|C_{GS}\|_\infty = \frac{4}{9}$

mentre

$$\|\mathbf{X}^{(1)} - \mathbf{X}^{(0)}\|_\infty = \|\mathbf{X}^{(1)}\|_\infty = \|\mathbf{Q}_{GS}\|_\infty =$$

$$= \|(\mathbf{D} + \mathbf{L})^{-1}\mathbf{B}\|_\infty = \|[\frac{1}{5} \quad \frac{16}{45} \quad \frac{404}{585}]^T\|_\infty = \frac{404}{585}$$

dove

$$(L+D)^{-1} = \begin{pmatrix} \frac{1}{10} & 0 & 0 \\ -\frac{2}{45} & \frac{4}{9} & 0 \\ -\frac{28}{585} & -\frac{16}{117} & \frac{4}{13} \end{pmatrix}.$$

Si osserva che il **metodo di Gauss-Seidel converge molto più velocemente del metodo di Jacobi**

Ripetere per $\|C_{GS}\|_1$.

Esercizio 2

Dato il sistema

$$\begin{cases} x_1 + 4x_2 = 5 \\ 3x_1 + x_2 + x_3 = 2 \\ 2x_2 + 4x_3 = 20 \end{cases}$$

stabilire se è possibile risolverlo usando il metodo di Jacobi per ogni scelta dell'approssimazione iniziale.

Soluzione

La matrice dei coefficienti del sistema è $A = \begin{pmatrix} 1 & 4 & 0 \\ 3 & 1 & 1 \\ 0 & 2 & 4 \end{pmatrix}$.

Si osserva subito che A non è a diagonale dominante nè per righe nè per colonne.

La matrice di iterazione del metodo di Jacobi è

$$C_J = \begin{pmatrix} 0 & -4 & 0 \\ -3 & 0 & -1 \\ 0 & -\frac{1}{2} & 0 \end{pmatrix}$$

per la quale risulta

$$\|C_J\|_\infty = \max\{4, 4, 1/2\} = 4 > 1$$

$$\|C_J\|_1 = \max\{3, 9/2, 1\} = 3 > 1$$

$$\rho(C_J) = \max_i |\lambda_i| = 5/\sqrt{2} > 1$$

infatti $det(C_J - \lambda I) = -\lambda^3 + \lambda/2 + 12\lambda = \lambda(-\lambda^2 + 25/2) = 0 \Leftrightarrow \lambda_1 = 0, \lambda_{2,3} = \pm 5/\sqrt{2}$ e quindi $\max_i |\lambda_i| = 5/\sqrt{2}$.

Quindi il metodo di Jacobi non converge.

Tuttavia, scambiando la prima e la seconda equazione del sistema, si ottiene un sistema equivalente la cui matrice dei coefficienti è data da

$$\hat{A} = \begin{pmatrix} 3 & 1 & 1 \\ 1 & 4 & 0 \\ 0 & 2 & 4 \end{pmatrix}$$

alla quale è associata la seguente matrice di iterazione del metodo di Jacobi

$$\hat{C}_J = \begin{pmatrix} 0 & -\frac{1}{3} & -\frac{1}{3} \\ -\frac{1}{4} & 0 & 0 \\ 0 & -\frac{1}{2} & 0 \end{pmatrix}$$

per la quale risulta

$$\|\widehat{C}_J\|_\infty = \max\{\frac{2}{3}, \frac{1}{4}, \frac{1}{2}\} = \frac{2}{3} < 1$$

$$\|\widehat{C}_J\|_1 = \max\{\frac{1}{4}, \frac{5}{6}, \frac{1}{3}\} = \frac{5}{6} < 1$$

$$\rho(\widehat{C}_J) = \max_i |\lambda_i| = 0.4257 < 1.$$

In questo caso il metodo di Jacobi converge per ogni scelta della approssimazione iniziale.

Inoltre, la velocità di convergenza del metodo è

$$-\log(\rho(\widehat{C}_J)) = -\log(0.4257) = 0.8540$$

e l'errore di approssimazione si riduce di un fattore 10^{-m} all'iterazione

$$K \approx -\frac{m}{\log(\rho(\widehat{C}_J))} = \frac{m}{0.8540}.$$

Esercizio 3

Scrivere una funzione Matlab che riceva in input una matrice e ne calcoli la norma spettrale (variabile di output) senza utilizzare le funzioni di Matlab predefinite per il calcolo della norma di matrici.

Si confronti il risultato con quello ottenuto usando le funzione predefinite di Matlab.

Soluzione

```
function [n_spettrale] = norma_spettrale(A)
% function [n_spettrale] = norma_spettrale(A)
% calcola la radice quadrata del raggio spettrale della matrice A'A
%
% INPUT
% A = matrice
%
% OUTPUT
% n_spettrale = radice quadrata del massimo degli autovalori
%               della matrice A'A


autovalori = eig(A'*A); % Calcolo degli autovalori
rho = max(abs(autovalori));
n_spettrale = sqrt(rho);
```

Dal Command Window

```
>> A = rand(3);
>> disp(A)
    0.4103    0.3529    0.1389
    0.8936    0.8132    0.2028
    0.0579    0.0099    0.1987
>> n_spettrale = norma_spettrale(A);
>> disp(n_spettrale)
    1.3485
    >> n_matlab = norm(A,2);
>> disp(n_matlab)
    1.3485
```

Calcolo degli autovalori in Matlab

V=eig(A)

V è il vettore degli autovalori della matrice A

[V,D] = eig(A)

V è la matrice degli autovettori di A,

D è matrice diagonale degli autovalori di A

Cosa restituisce la funzione eigs(A,k)? (Usare lo help di Matlab!)

Esercizio 4

1. Scrivere una funzione Matlab **CS_iterativi** che riceva in input una matrice A e una variabile di tipo char che vale $'J'$ o $'GS'$, rispettivamente Jacobi e Gauss-Seidel, e restituisca in output una variabile logica cs che vale 1 se il metodo iterativo scelto soddisfa la condizione sufficiente per la convergenza per ogni scelta della approssimazione iniziale e 0 altrimenti. La funzione deve anche stampare un messaggio di output relativo alla convergenza del metodo.

2. Scrivere una funzione Matlab **CNES_iterativi** che abbia le stesse variabili di input della funzione **CS_iterativi** ma che restituisca in output una variabile logica cnes che vale 1 se il metodo iterativo scelto soddisfa la condizione necessaria e sufficiente per la convergenza e 0 altrimenti.

Soluzione

```
function [cs] = CS_iterativi(A,tipo)
% verifica le condizioni sufficienti per la  convergenza dei metodi di
% Jacobi e Gauss-Seidel per la soluzione di sistemi aventi A come matrice
% dei coefficienti
%
% INPUT
% A = matrice dei coefficienti del sistema
% tipo = variabile char. Se tipo = 'J', si applica il metodo di
%        Jacobi. Se tipo = 'GS', si applica il metodo di Gauss-Seidel
%
% OUTPUT
% cs = variabile logica. Se cs = 1, le condizioni sufficienti sono
%      verificate. Se cs  = 0, le condizioni sufficienti non
%      sono verificate
%
%
```

```
if tipo == 'J'
   Minv = inv(diag(diag(A)));
   L = tril(A,-1);
   U = triu(A,1);
   CJ = -Minv*(L+U);
   norma_1 = norm(CJ,1);
   norma_inf = norm(CJ,inf);
   test = 0;
   if norma_1 < 1
     fprintf('il metodo di Jacobi converge rispetto alla norma 1 \n')
     cs = 1;
     test = 1;
   end
   if norma_inf < 1
     fprintf('il metodo di Jacobi converge rispetto alla norma infinito')
     cs = 1;
     test = 1;
   end
```

```
if test  == 0
  cs = 0;
  fprintf('la matrice di iterazione del metodo di Jacobi non...
   soddisfa la condizione sufficiente')
end
```

```
elseif tipo == 'GS'
   L = tril(A);
   U = triu(A,1);
   Minv = inv(L);
   CGS = -Minv*U;
   norma_1 = norm(CGS,1);
   norma_inf = norm(CGS,inf);
   test = 0;
   if norma_1 < 1
    fprintf('il metodo di Gauss-Seidel converge rispetto alla norma 1\n')
    cs = 1;
    test = 1;
   end
   if norma_inf < 1
    fprintf('il metodo di Gauss-Seidel converge rispetto alla...
     norma infinito')
    cs = 1;
    test = 1;
   end
```

```
   if test == 0
     cs = 0;
     fprintf('la matrice di iterazione del metodo di Gauss-Seidel...
      non soddisfa la condizione sufficiente')
   end
else
 fprintf('la variabile tipo non corrisponde ai metodi...
  di Jacobi o Gauss-Seidel')
    cs = [];
end
```

```
function [cnes] = CNES_iterativi(A,tipo)
% verifica la condizione necessaria e  sufficiente per
% la  convergenza dei metodi di Jacobi e Gauss-Seidel
% per la soluzione di sistemi aventi A come matrice
% dei coefficienti
%
% INPUT
% A = matrice dei coefficienti del sistema
% tipo = variabile char. Se tipo = 'J', si applica il metodo
         di Jacobi. Se tipo = 'GS',
%        si applica il metodo di Gauss-Seidel
%
% OUTPUT
% cnes = variabile logica. Se cnes = 1, la condizione
%        necessaria e sufficiente e' verificata.
%        Se cnes  = 0, la condizione necessaria e
%        sufficiente non e' verificata
%
%
%
```

```
if tipo == 'J'
   Minv = inv(diag(diag(A)));
   L = tril(A,-1);
   U = triu(A,1);
   CJ = -Minv*(L+U);
   raggio_spettrale = max(abs(eig(CJ)));
   if raggio_spettrale < 1
       fprintf('il metodo di Jacobi converge')
       cnes = 1;
   else
       cnes = 0;
       fprintf('il metodo di Jacobi non converge')
   end
```

```
elseif tipo == 'GS'
   L = tril(A);
   U = triu(A,1);
   Minv = inv(L);
   CGS = -Minv*U;
   raggio_spettrale = max(abs(eig(CGS)));
   if raggio_spettrale < 1
       fprintf('il metodo di Gauss-Seidel converge')
       cnes = 1;
   else
       cnes = 0;
       fprintf('il metodo di Gauss-Seidel non converge')
   end
else
    fprintf('la variabile tipo non corrisponde ai metodi...
     di Jacobi o Gauss-Seidel')
    cnes = [];
end
```

Dal Command Window

```
>> alpha = 9/4;
>> beta=13/4;
>> A = [10 1 1  ; 1 alpha 1; 2 1 beta]

A =

   10.0000    1.0000    1.0000
    1.0000    2.2500    1.0000
    2.0000    1.0000    3.2500

>> [cs] = CS_iterativi(A,'J');
il metodo di Jacobi converge rispetto alla norma infinito>>

[cs] = CS_iterativi(A,'GS');
il metodo di Gauss-Seidel converge rispetto alla norma 1
il metodo di Gauss-Seidel converge rispetto alla norma infinito>>
```

```
>> alpha = 4;
>> beta=3;
>> A = [10 1 1  ; 1 alpha 1; 2 1 beta]

A =

    10     1     1
     1     4     1
     2     1     3

>> [cs] = CS_iterativi(A,'J');
il metodo di Jacobi converge rispetto alla norma 1
>> [cs] = CS_iterativi(A,'GS');
il metodo di Gauss-Seidel converge rispetto alla norma 1
il metodo di Gauss-Seidel converge rispetto alla norma infinito>>
```

Verifichiamo ora la condizione necessaria e sufficiente

```
>> alpha = 9/4;
>> beta=13/4;
>> A = [10 1 1  ; 1 alpha 1; 2 1 beta]

A =

   10.0000    1.0000    1.0000
    1.0000    2.2500    1.0000
    2.0000    1.0000    3.2500

>> [cnes] = CNES_iterativi(A,'J');
il metodo di Jacobi converge>>
>> [cnes] = CNES_iterativi(A,'GS');
il metodo di Gauss-Seidel converge>>
```

```
>> alpha = 4;
>> beta=3;
>> A = [10 1 1  ; 1 alpha 1; 2 1 beta]

A =

    10     1     1
     1     4     1
     2     1     3

>> [cnes] = CNES_iterativi(A,'J');
il metodo di Jacobi converge>>
>> [cnes] = CNES_iterativi(A,'GS');
il metodo di Gauss-Seidel converge>>
```

Esercizio 5

- Scrivere una funzione Matlab **jacobi.m** che implementi il metodo iterativo di Jacobi. La funzione, oltre alla matrice $\mathbf{A}$, al termine noto $\mathbf{b}$ del sistema e al vettore approssimazione iniziale $X0$, deve ricevere in input la precisione ε richiesta alla soluzione e il numero massimo di iterazioni consentite max_iter. Le variabili di output della funzione sono l'approssimazione della soluzione prodotta X, l'errore massimo ad ogni iterazione ERR e il numero di iterazioni effettuate $iter$. La funzione deve stampare l'errore ad ogni iterazione e un messaggio relativo alla convergenza del metodo.

- Scrivere una funzione Matlab **gauss_seidel.m** che implementi il metodo di Gauss-Seidel. Le variabili di input e di output sono le stesse della funzione **jacobi**.

- Si consideri il seguente sistema

$$\begin{cases} 4x_1 + +2x_2 - 2x_3 &= 14 \\ -2x_1 + 5x_2 + 3x_3 &= \text{-3} \\ 2x_1 - x_2 + 4x_3 &= \text{-8.5} \end{cases}$$

e lo si risolva con il metodo di Jacobi e quello di Gauss Seidel usando il vettore $\mathbf{X}^{(0)} = [1 \quad 1 \quad 1]^T$ e richiedendo una precisione non inferiore a 10^{-6} da raggiungere in meno di 2000 iterazioni. Si confrontino i risultati con la soluzione data dal solutore di Matlab.

Si sostituisca il coefficiente di x_1 della prima equazione con 1 e si ripeta l'esercizio. Commentare i risultati.

Soluzione

```
function [X, ERR, iter] = jacobi(A,b,X0,eps,max_iter)
% function [X] = jacobi(A,b,X0,eps,max_iter)
% Risolve un sistema lineare con il metodo di Jacobi
%
% Input:
% A = matrice dei coefficienti del sistema
% B = vettore dei termini noti
% epsilon = accuratezza della soluzione
% max_iter = numero massimo di iterazioni consentite
%
% Output:
% X = vettore soluzione
% ERR = vettore dell'errore massimo per ogni iterazione
% iter = numero di iterazioni eseguite
```

```
% Calcolo delle dimensioni della matrice
dimA = size(A);
n = dimA(1);

% Costruzione matrice di iterazione
D = diag(diag(A));
L = tril(A,-1);
U = triu(A,1);
Minv = inv(D);
CJ = -Minv*(L+U);
QJ = Minv*b;
```

```
% Calcolo autovalori e verifica C.N.S. di convergenza
rhoCJ = max(abs(eig(CJ)))
if (rhoCJ >= 1)
    error(Attenzione: ==>> rho > 1, il metodo non converge)
    return
end
```

```
% Ciclo iterativo
err = 100;
iter = 0;
ERR = [];
while (err>eps & iter<= max_iter)
    X = CJ*X0+QJ;
    err = norm(X-X0,inf);
    ERR = [ERR err];
    X0 = X;
    iter = iter + 1;
end
disp('ERRORE')
fprintf('%3.15f\n',ERR)
if (iter > max_iter) & (err>eps)
    fprintf('Il metodo non ha raggiunto l''accuratezza richiesta ...
    dopo %7d iterazioni',max_iter)
end
```

In alternativa

```
function [X, ERR, iter] = jacobi_2(A,b,X0,eps,max_iter)
% function [X] = jacobi_2(A,b,X0,eps,max_iter)
% Risolve un sistema lineare con il metodo di Jacobi
%
% Input:
% A = matrice dei coefficienti del sistema
% B = vettore dei termini noti
% epsilon = accuratezza della soluzione
% max_iter = numero massimo di iterazioni consentite
%
% Output:
% X = vettore soluzione
% ERR = vettore dell'errore massimo per ogni iterazione
% iter = numero di iterazioni eseguite
```

```
% Calcolo delle dimensioni della matrice
dimA = size(A);
n = dimA(1);
X0 = X0(:)';

% Costruzione matrice di iterazione
Minv = inv(diag(diag(A)));
L = tril(A,-1);
U = triu(A,1);
CJ = -Minv*(L+U);

% Calcolo autovalori e verifica C.N.S. di convergenza
rhoCJ = max(abs(eigs(CJ,1))) % eigs(MAT,m) restituisce gli m
                             % autovalori piu'grandi in valore
                             % assoluto della matrice MAT
if (rhoCJ >= 1)
    error('Attenzione: ==>> rho > 1, il metodo non converge')
    return
end
```

```
% Ciclo iterativo
err = 100;, iter = 1;, ERR = [];
while (err>eps & iter<= max_iter)
 for i = 1:n
     X(i) = (-sum(A(i,1:i-1).*X0(1:i-1))-...
              sum(A(i,i+1:end).*X0(i+1:end))+b(i))/A(i,i);
 end
 err = norm(X-X0,inf);
 ERR = [ERR err];
 X0 = X;
 iter = iter + 1;
end
```

```
fprintf('%3.15f\n',ERR)
if (iter >= max_iter) & (err>eps)
    fprintf('Il metodo non ha raggiunto l''accuratezza richiesta...
     dopo %7d iterazioni',max_iter)
end
```

```
function [X, ERR, iter] = gauss_seidel(A,b,X0,eps,max_iter)
% function [X] = gauss_seidel(A,b,X0,eps,max_iter)
% Risolve un sistema lineare con il metodo di Gauss Seidel
%
% Input:
% A = matrice dei coefficienti del sistema
% B = vettore dei termini noti
% epsilon = accuratezza della soluzione
% max_iter = numero massimo di iterazioni consentite
%
% Output:
% X = vettore soluzione
% ERR = vettore dell'errore massimo per ogni iterazione
% iter = numero di iterazioni eseguite
```

```
% Calcolo delle dimensioni della matrice
dimA = size(A);
n = dimA(1);
X0 = X0(:)';

% Costruzione matrice di iterazione
L = tril(A,);
Minv = inv(L);
U = triu(A,1);
C_GS = -Minv*(U);

% Calcolo autovalori e verifica C.N.S. di convergenza
rhoCGS = max(abs(eig(C_GS)))
if (rhoCGS >= 1)
    error('Attenzione: ==>> rho > 1, il metodo non converge')
    return
end
```

```
% Ciclo iterativo
err = 100;, iter = 1;, ERR = [];
while (err>eps & iter<= max_iter)
 for i = 1:n
  X(i)=(-sum(A(i,1:i-1).*X(1:i-1)-...
        A(i,i+1:end).*X0(i+1:end))+n(i))/A(i,i);
 end
 err = norm(X-X0,inf);
 ERR = [ERR err];
 X0 = X;
 iter = iter + 1;
end,

fprintf('%3.15f\n',ERR)
if (iter > max_iter) & (err>eps)
    fprintf('Il metodo non ha raggiunto l''accuratezza richiesta...
     dopo %7d iterazioni',max_iter)
end
```

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

A =

     4     2    -2
    -2     5     3
     2    -1     4

>> b = [14 -3 -8.5]'

b =

   14.0000
   -3.0000
   -8.5000
```

```
>> [X, ERR, iter] = jacobi(A,b,[1 1 1]',1*10^-6,2000);

rhoCJ =

    0.7909

ERRORE
3.375000000000000
3.025000000000000
2.362500000000001
1.635000000000001
1.496250000000000
0.929500000000000
0.920000000000000
0.609700000000000
0.541725000000000
0.383445000000000
0.317573000000001
```

0.231347000000000
0.212980800000000
0.141649650000000
0.136747380000000
0.101036590000000
0.084248038000000
0.068395719000000
0.049738780800000
0.044285193900000
0.027989769480000
0.027509407940000
0.017930043160000
0.016384662534000
0.011397279018000
0.009315356243400
0.006951628820000
0.006228825022800
0.004059633705920
0.004043103111768

0.00291146982236 8
0.00251758869798 2
0.00199572699417 8
0.00150330106711 3
0.00130656018475 5
0.00085719831814 5
0.00082029580557 8
0.00052579834984 7
0.00049403603024 0
0.00033783824130 8
0.00028447114443 2
0.00020828455069 5
0.00018162795310 5
0.00012306171046 5
0.00011921020566 9
0.00008359071352 3
0.00007502422454 6
0.00005805656420 9
0.00004529711588 3

0.000038440890698
0.000026155716167
0.000024392798587
0.000015374488585
0.000014852072011
0.000009986782485
0.000008656670015
0.000006223024281
0.000005279910011
0.000003718642557
0.000003505109600
0.000002390619110
0.000002229576345
0.000001683558886
0.000001360853058
0.000001127799683
0.000000795325089

```
>> X

X =

   1.500000
   1.500000
  -2.500000

>> iter

iter =

    66
```

Risolvendo con Gauss-Seidel

```
>> [X, ERR, iter] = gauss_seidel(A,b,[1 1 1]',1*10^-6,2000);
rhoCGS =
   0.38729833462074
ERRORE
4.82500000000000
2.09000000000000
1.02950000000000
0.45621875000000
0.20882656250000
0.04642179687500
0.02320231562500
0.01014886738281
0.00471307764648
0.00106986259814
0.00052277481365
0.00022567485961
```

```
0.00010634059009 1
0.00002463609560 7
0.00001177542698 7
0.00000501609410 3
0.00000239866636 7
0.00000056685238 6

>> X'

ans =

   1.500000
   1.500000
  -2.500000

>> iter

iter =

    18
```

Con il solutore di Matlab

```
>> X=A\b

X =

   1.5000
   1.5000
  -2.5000
```

Il metodo di Gauss-Seidel produce una soluzione con un numero inferiore di iterazioni (18) rispetto al metodo di Jacobi (66), infatti il raggio spettrale della matrice di iterazione del metodo di Gauss-Seidel è minore del raggio spettrale della matrice di iterazione del metodo di Jacobi, essendo $\rho(C_{GS}) \approx 0.3873$ e $\rho(C_J) = 0.7909$. Inoltre, risulta

$$E_J^{(66)} \approx 4 \cdot 10^{-7}$$

$$E_{GS}^{(18)} \approx 2 \cdot 10^{-7}$$

Sostituendo con 1 il coefficienti di x_1 della prima equazione, si ottiene il seguente risultato

```
>> A = [1 2 -2; -2 5 3; 2 -1 4]
A =
     1     2    -2
    -2     5     3
     2    -1     4
>> [X, ERR, iter] = gauss_seidel(A,b,[1 1 1]',1*10^-6,2000);
rhoCGS =
   1.28197051490249
??? Error using ==> gauss_seidel
Attenzione: ==>> rho > 1, il metodo non converge

>> [X, ERR, iter] = jacobi(A,b,[1 1 1]',1*10^-6,2000);
rhoCJ =
   1.41081086544963
??? Error using ==> jacobi
Attenzione: ==>> rho > 1, il metodo non coverge
```

Per ognuno dei metodi il raggio di convergenza è maggiore di 1 e quindi il metodo iterativo non converge alla soluzione.

Esercizio

Scrivere una funzione Matlab def_pos.m che riceva in input una matrice A e stabilisca se è una matrice simmetrica e definita positiva usando il criterio di Sylvester. La matrice deve restituire come output una variabile logica dp che valga 1 nel caso la risposta sia affermativa, 0 viceversa. La funzione deve inoltre stampare opportuni messaggi relativi al risultato ottenuto.

Soluzione

```
function [df] = def_pos(A)
%function [df] = def_pos(A)
% stabilisce se la matrice quadrata A e'' simmatrica e definita positiva
%
% INPUT
% A = matrice quadrata
%
% OUTPUT
% df = variabile logica. df = 1 se A e'' simmetrica e definita positiva,
%                              0 altrimenti

% controlla se A e'' una matrice quadrata
[m,n] = size(A);
if m ~=n
    error('la matrice A deve essere quadrata!!!')
end
```

```
if A==A' % controllo se A e'' simmetrica
    i = 1;
    while (i<=m) & (det(A(1:i,1:i))>0)  % controllo prima la dimensione
                                        % e poi il determinante!!!
       i = i+1;
    end
    if i == m+1
       df = 1;
       disp('la matrice e'' simmetrica e definita positiva')
    else
       df = 0;
       disp('la matrice e'' simmetrica ma non e'' definita positiva')
    end
else
    disp('la matrice non e'' simmetrica')
    df = 0;
end
```

Dal Command Window

```
>> A = diag(ones(10,1))+diag(.5*ones(9,1),-1)+diag(.5*ones(9,1),1);
>> df = def_pos(A);
la matrice e' simmetrica e definita positiva
>> disp(df)
     1
```

Nel caso della passeggiata aleatoria bisogna risolvere il sistema lineare

$$\begin{cases} 2p_1 & -p_2 & & & & & = & 1 \\ -p_1 & +2p_2 & -p_3 & & & & = & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & & \\ & & & -p_{N-3} & +2p_{N-2} & -p_{N-1} & = & 0 \\ & & & & -p_{N-2} & +2p_{N-1} & = & 0 \end{cases}$$

nelle incognite p_i, $i = 1, \ldots, N-1$.

La soluzione esatta è $p_i = 1 - \frac{i}{N}$, $i = 0, \ldots, N$.

```
N = input('inserisci la dimensione del problema N = ')
d = 2*ones(1,N-1);
a = -1*ones(1,N-2);
s = -1*ones(1,N-2);
b = zeros(N-1,1);
b(1) = 1;
A = diag(d)+diag(a,-1)+diag(a,1);
```

```
x_true = 1-(1:N-1)/N;

[X_J,ERR_J,iter_J] = jacobi(A,b,zeros(size(b)),.5*10^-3,50);
err_J = max(abs(X_J'-x_true))

[X_GS,ERR_GS,iter_GS] = gauss_seidel(A,b,zeros(size(b)),.5*10^-3,50);
err_GS = max(abs(X_GS-x_true))
```

Per il **metodo di Jacobi** si ha:

N	$\|C_J\|_1$	$\|C_J\|_\infty$	$\rho(C_J)$	$\|X^{(50)} - X^{(49)}\|_\infty$	$\|E^{(50)}\|_\infty$	decimali esatti
11	1	1	0.9595	$0.64 \cdot 10^{-2}$	$0.77 \cdot 10^{-1}$	0
21	1	1	0.9888	$0.95 \cdot 10^{-2}$	0.36	0
51	1	1	0.9981	$0.95 \cdot 10^{-2}$	0.68	0
101	1	1	0.9995	$0.95 \cdot 10^{-2}$	0.81	0

per il **metodo di Gauss-Seidel** si ha:

N	$\|C_{GS}\|_1$	$\|C_{GS}\|_\infty$	$\rho(C_{GS})$	$\|X^{(50)} - X^{(49)}\|_\infty$	$\|E^{(50)}\|_\infty$	decimali esatti
11	0.9990	0.9980	0.9206	$0.69 \cdot 10^{-3}$	$0.80 \cdot 10^{-2}$	1
21	1	1	0.9778	$0.42 \cdot 10^{-2}$	0.18	0
51	1	1	0.9962	$0.44 \cdot 10^{-2}$	0.55	0
101	1	1	0.9990	$0.44 \cdot 10^{-2}$	0.73	0

Strutture dati

Cell permette di collezionare in un 'unica variabile oggetti di vario tipo (vettori, matrici, variabili numeriche o logiche, caratteri o stringhe) ma anche variabili dello stesso tipo ma di dimensione diversa.

Le variabili di tipo cell si definiscono tra parentesi graffe. Le componenti si elencano una dopo l' altra e separate da virgole

C = {componente1, componente2, ... , componenteN};

C{i} estrae la i-esima componente di C

C(i) indica la i-esima componente di C

Esempio:

```
>> C = {ciao, [4 3 1 2],3,[1 7 2; 0 5 8; 1 0 9]};
% definisce una variabile cell composta da 4 elementi:
% una stringa, un vettore, un numero e una matrice.

>> whos
Name      Size              Bytes  Class
  C       1x4                488  cell array

>>C{2}  % estrae il secondo elemento della variabile di tipo cell C
ans =
     4     3     1     2
```

Nota: una variabile di tipo Cell contiene copie di variabili e non puntatori a variabili.

Quindi se $C = \{A, B\}$, con A, B due generiche variabili, il contenuto di C non cambia se in seguito A e B sono modificate

Esempio:

```
>> A = [2 5 3 6];
>> B = [1 5; 8 9;3 6];
>> C = {A,B};
>> B = 1;
>> C{2}
ans =
     1     5
     8     9
     3     6
```

C può essere inizializzata con il comando C = cell(m,n)

C è composta da mxn matrici vuote

Celldisp(C): visualizza il contenuto di una cell

Cellplot(C): disegna il contenuto di una cell

Le celle possono essere annidate, cioè un elemento di una cell può essere esso stesso una cell.

Esempio:

```
C = cell(1,4);
A = [1 2 3; 4 5 6];
v = [1 4 6];
C{1} = {A,v};
C{2} = 4;
C{4} = stringa;
```

Per estrarre il contenuto della matrice A si digita il comando

```
>>C{1}{1}
```

gli indici sono ordinati da sinistra verso destra, dalla cell più esterna a quella più interna

Alcune funzioni

Varargin = variabile di tipo cell contenente le variabili di input di una funzione.

Si usa quando gli input di una funzione possono variare (esistono parametri di default)

L' uso di questa funzione richiede un ordine preciso delle variabili di input che devono essere assegnate all' inizio della funzione combinata con il comando nargin.

Nargin = numero degli elementi in varargin

Oss: In modo analogo, per lo output si usa varargout combinata con nargout.

Esempio:

```
function [R] = confronta(x,varargin)
% stabilisce se ci sono elementi in x il cui valore supera un valore
% fissato T. Se si, assegna alla variabile R il valore 1 altrimenti 0
if nargin == 1 % se il numero delle variabili di input e'' 1,
               % allora assegna un valore di default a T
   T = 256;
else
   T = varargin{1}; % altrimenti assegna a T il primo elemento di varagin
end
% seleziona gli elementi di x il cui valore e'' maggiore di T
ind = find(x>T);
if isempty(ind),
   R = 0,
else
   R=1;
end
```

Il comando switch

Switch-case-Otherwise:

```
switch    nome_variabile
      case    valore1
                primo blocco di istruzioni
      case    valore2
              secondo blocco di istruzioni
              ......
      otherwise
                ultimo blocco di istruzioni
end
```

Se il valore di nome_variabile è valore1 viene eseguito il primo blocco di istruzioni. Se è valore2, il secondo blocco, e cos via, altrimenti, se non è alcuno dei valori elencati, esegue l' ultimo blocco di istruzioni.

Esempio:

```
a = rem(b,3);          % resto della divisione per 3
switch  a
  case 0
      c = b/3;
  case 1
      c = (b-a)/3;
  otherwise
      c = b-a/2;
end
```