Calcolo Numerico

(A.A. 2014-2015)

Lab n. 10

Metodi iterativi per sistemi lineari

3 Dicembre 2014

Strutture dati

I **cell arrays** e le **struct** consentono il raggruppamento di tipi di **dati diversi** (vettori, matrici, variabili numeriche o logiche, caratteri o stringhe), ma in relazione fra loro in una sola variabile.

Sono dei contenitori per vari tipi di dati per cui le operazioni matematiche non sono definite.

Strutture dati - cell

I **cell array** sono matrici i cui elementi sono **celle**. Una cella può contenere un qualunque tipo di dato e celle diverse dello stesso array possono contenere dati di tipo diverso. Un **cell array** serve quindi per collezionare un insieme di dati correlati ma con dimensione e tipo differenti.

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

Inizializzazione: **C = cell(m,n)** crea mxn matrici vuote

Definizione: **C = { A1, A2, . . . ; B1, B2, . . . ; . . . }**

Indicizzazione: **D=C{i,j}** assegna alla variabile D l'elemento **(i,j)** di **C**

Content: **E=C(i,j)** assegna la componente **(i,j)** di **C** alla variabile **D** che è un cell array composto da una sola cella

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
```

Per accedere ad un elemento di un vettore all'interno della cella

```
% prendo la II componenente del vettore contenuto nella seconda cella
>> C{2}(2)
```

celldisp(C) : visualizza il contenuto di una cell

cellplot(C): disegna il contenuto di una cell

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
```

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

Strutture dati - struct

Una **struct** è una variabile composta da diversi elementi che vengono chiamati **campi**, e vengono richiamati con una notazione molto simile alla notazione campo degli oggetti java. Anche i campi di una struct possono contenere qualsiasi tipo di dato.

Inizializzazione: **S = struct('nome_campo',{},'nome_campo2',{})**
crea una struct vuota con due campi

Definizione: **S = struct('nome_campo',val,'nome_campo2',val2)**
crea una struct con due campi

Accesso: **S.nome_campo** i campi di una struct vengono specificati utilizzando un punto seguito dal nome del campo

Esempio

```
>> Studente = struct('nome','Alessandro', 'matricola','123');
>> Studente.nome
ans =
Alessandro
```

E' possibile costruire un **array di strutture**, per esempio per gestire un insieme di studenti di un corso.

```
>> Studente(2).nome = 'Anna';
>> Studente(2).matricola = '124';
```

I campi di una struct possono essere dei vettori.

```
>> Studente(1).esami_superati(1) = 'Annalisa';
>> Studente(1).esami_superati(2) = 'Analisi 1';
```

varargin e varargout

La variabile predefinita **varargin** è una variabile di tipo cell che consente di specificare un numero variabile di **parametri di input** in una funzione.

L' uso di questa variabile richiede un **ordine preciso delle variabili di input** che sono assegnate alla funzione e si usa in combinazione con la variabile **nargin** che restituisce il numero delle variabili di input con cui è richiamata la funzione dall'esterno.

Analogamente per le **variabili di output** si usa la variabile di tipo cell **varargout** combinata con la variabile **nargout**.

varargin e varargout

Quindi per scrivere una funzione con un numero variabile di parametri di input/output si usa la sintassi

function [O1,. . . ,ON,**varargout**] = myf(I1,. . . ,IN,**varargin**)

dove

- **I1,. . . ,IN** sono N parametri di input obbligatori
- **varargin** contiene tutti gli input opzionali passati alla funzione
- **O1,. . . ,ON** sono parametri di output obbligatori
- **varargout** contiene tutti gli output opzionali richiesti

Esempio:

```
function [R] = confronta(x,varargin)
% stabilisce se ci sono elementi nel vettore x il cui valore supera una
% soglia 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;
elseif nargin == 2
   T = varargin{1}; % altrimenti assegna a T il I elemento di varargin
else
   error('il numero degli input e'' errato')
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 così via, **altrimenti**, se non è alcuno dei valori elencati, esegue l' **ultimo blocco di istruzioni**.

Esempio:

```
a = rem(n,2);        % resto della divisione per 2
switch  a
  case 0
      disp('n e'' un numero pari')
  case 1
      disp('n e'' un numero dispari')
  otherwise
      disp('n non e'' un numero intero')
end
```

Il comando **switch** si usa anche con variabili di tipo string.

Esercizio 1

Scrivere la funzione Matlab **num_cond** che legga in **input** una matrice quadrata **A** e la variabile numerica **tipo_norma** che può assumere i valori $1, 2$ o inf La funzione restituisca in **output** il numero di condizionamento **A** rispetto alla norma indicata nella variabile tipo_norma.

Se la variabile tipo_norma non viene data in input, la funzione deve calcolare il numero di condizionamento rispetto alla norma infinito.

La funzione deve prevedere come ulteriori variabili di output, se richieste, la matrice inversa di A e la norma di A calcolata rispetto a tipo_norma.

function [C, **varargout**] = num_cond(A, **varargin**)

Esercizio 1

Usiamo la funzione num_cond per calcolare il numero di condizionamento delle seguenti matrici:

$$A = \begin{pmatrix} 1 & 1.01 \\ 0.99 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 101 \\ 99 & 1 \end{pmatrix}$$

```
>> A = [1 1.01; 0.99 1];
>> KA_inf = num_cond(A);
>> [KA_1, IA] = num_cond(A,1);
```

Esercizio 2

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.

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!)

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**.

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

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

Una possibile **decomposizione** di A è

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

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

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

Convergenza dei metodi di Jacobi e Gauss-Seidel

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

Attenzione: Le **condizioni di convergenza** per le matrici di iterazione $\mathrm{C_J}$ e $\mathrm{C_{GS}}$ 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

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

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

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

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

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

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

Esercizio 3

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.

Dal **Command Window**

```
>>  A = [3 1 1; 1 4 0; 0 2 4];
>> [cs] = CS_iterativi(A,'J');
il metodo di Jacobi converge rispetto alla norma 1
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
```

Esercizio 4

- Scrivere una funzione Matlab **jacobi.m** che implementi il **metodo iterativo di Jacobi**. La funzione deve ricevere in **input** la matrice $\mathbf{A}$, il termine noto $\mathbf{b}$ del sistema, il vettore approssimazione iniziale $\mathbf{X0}$, la precisione ε richiesta alla soluzione e il numero massimo di iterazioni consentite $\mathbf{max\_iter}$. Se ε e $\mathbf{max\_iter}$ non sono passati come input definire dei valori di default.

  Le variabile di **output** della funzione sono l'approssimazione della soluzione prodotta $\mathbf{X}$, e, se eventualmente richiesti, l'errore massimo ad ogni iterazione $\mathbf{ERR}$ e il numero di iterazioni effettuate $\mathbf{iter}$. La funzione deve stampare 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** ma sono tutte obbligatorie.

- 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.

Esercizio 4

```
>> A = [4 2 -2; -2 5 3; 2 -1 -4];
>> b = [14 -3 -8.5]'
>> [X, ERR, iter] = jacobi(A,b,[1 1 1]',1*10^-6,2000);
il raggio spettrale vale 0.605
>> X

X =

  7.2143
    -1.3571
    6.0714

>> iter

iter =

    31
```

Risolvendo con Gauss-Seidel

```
>> [X, ERR, iter] = gauss_seidel(A,b,[1 1 1]',1*10^-6,2000);
rhoCGS =
   0.46949
>> X
X =
       7.2143
      -1.3571
       6.0714

>> iter

iter =

    23
```

Con il solutore di Matlab

```
>> X=A\b

X =
       7.2143
      -1.3571
       6.0714
```

Il metodo di Gauss-Seidel produce una soluzione con un numero inferiore di iterazioni (23) rispetto al metodo di Jacobi (31), 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 $\rho(C_{GS}) \approx 0.46949$ e $\rho(C_J) = 0.605$. Inoltre, risulta

$$E_J^{(31)} \approx 7 \cdot 10^{-7}$$

$$E_{GS}^{(23)} \approx 6 \cdot 10^{-7}$$

Osservazione

Per un'implementazione più efficiente dei metodi di Jacobi e Gauss-Seidel, si evita il calcolo dell'inversa della matrice **M**: le matrici di iterazione e i termini noti, **C_J, Q_J, C_GS, Q_GS**, sono calcolate risolvendo i seguenti sistemi lineari:

$$\mathbf{D}\mathbf{C_J} = -(\mathbf{L}+\mathbf{U})$$
$$\mathbf{D}\mathbf{Q_J} = \mathbf{B}$$

$$(\mathbf{D}+\mathbf{L})\mathbf{C_{GS}} = -\mathbf{U}$$
$$(\mathbf{D}+\mathbf{L})\mathbf{Q_{GS}} = \mathbf{B}$$

In **Matlab**:

```
C_J = -D\(L+U);
Q_J = D\b;

C_GS = -(D+L)\U;
Q_GS = (D+L)\b;
```

Dal command window

```
>> A = [2 2 -2; -2 5 3; 2 -1 3];
>> b = [14 -3 -8.5]';
>> [X_J, E_J, n_iter_J] = Jacobi_fast(A,b,zeros(size(b)),10^-6,500);
Error using Jacobi_fast (line 57)
Attenzione: ==>> rho > 1, il metodo non coverge
>> [X_GS,E_GS,n_iter_GS]=gauss_seidel_fast(A,b,zeros(size(b)),10^-6,500);
il raggio spettrale vale 0.632
```

Esercizio 5

Costruire una matrice tridiagonale A di ordine $N = 100$, t.c.

$\mathbf{a(i,i) = 5, i = 1, \ldots, 100}$
$\mathbf{a(i,i-1) = -2, i = 2, \ldots, 100}$
$\mathbf{a(i,i+1) = -2, i = 1, \ldots, 99}$

Scrivere uno script Matlab che risolvi il sistema $\mathbf{Ax = b}$, dove il vettore del termine noto è dato da $\mathbf{b_i = i, i = 1, \ldots, 100}$. In particolare

- risolvere il sistema con il metodo di Gauss, il metodo di Thomas e (se convergenti) con i metodi iterativi di Gauss-Seidel e Jacobi
- qual è l'algoritmo più veloce da un punto di vista computazionale?
- quale dei due metodi iterativi converge più velocemente?
- si riporti in un grafico il numero di iterazioni necessarie ai due metodi iterativi per convergere al variare della tolleranza

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