Algebra lineare numerica in Matlab per Ingegneria dell'Energia
Esercizi risolti. [1]

A. Sommariva[2]

Abstract

Fattorizzazione LU, risoluzione di sistemi lineari, metodi iterativi, esempi.

Ultima revisione: 20 dicembre 2018

1. La routine `fattorizzazione_LU`

Il seguente pseudocodice

function [L,A]=fattorizzazione_LU(A)

for *k=1,...,n-1* **do**
  $l_{k,k} = 1$ **for** *i=k+1,...,n* **do**
    $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$
    for *j=k,...,n* **do**
      $a_{i,j} = a_{i,j} - l_{i,k} a_{k,j}$
    end
  end
end
$l_{n,n} = 1$

calcola la fattorizzazione LU di una matrice $A = (a_{i,j})$ data in input, e in output offre le matrici triangolari $L = (l_{i,j})$ e $U = (u_{i,j})$ che corrisponde alla matrice A alla fine del processo (che per questo appare pure in output).

Lo si implementi in Matlab mediante la function

`fattorizzazione_LU`

ricordando che n è il numero di righe e colonne di A.

1.1. Risoluzione

Salviamo la seguente routine.

```
function [L,A]=fattorizzazione_LU(A)

% oggetto:
% fattorizzazione LU senza pivoting della matrice A.
%
% input:
% A: matrice n x n;
%
% output:
% L: matrice triangolare inferiore con elementi diag. ...
     uguali a 1;
% A: matrice triangolare superiore che e' la U della ...
     fattorizzazione A=LU.

n=size(A,1);
L=zeros(n,n);
for k=1:n-1
    L(k,k)=1;
    for i=k+1:n
        L(i,k)=A(i,k)/A(k,k);
        for j=k:n
            A(i,j)=A(i,j)-L(i,k)*A(k,j);
        end
    end
end
L(n,n)=1;
```

Il file è essenzialmente quanto scritto nel pseudocodice, con l'aggiunta

- del comando `n=size(A,1);` per determinare il numero di righe (e di colonne) di A;
- dell'assegnazione `L=zeros(n,n);` per *inizializzare* la matrice L, ovvero dare una valore a tutte le sue componenti prima di eseguire il processo.

2. La routine `fattorizzazione_LU` e la soluzione di sistemi lineari

Si scriva uno script `demo_eliminazione_gaussiana` che definita la matrice

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

e il termine noto

$$b = \begin{pmatrix} 3 \\ 4 \\ 3 \end{pmatrix}$$

calcoli la soluzione del sistema lineare $Ax = b$, risolvendo i due sistemi triangolari citati in precedenza (con P uguale alla matrice identica), mediante il comando $\backslash$ di Matlab.

La soluzione corretta è

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

2.1. *Risoluzione*

```
function demo_eliminazione_gaussiana

% esempio relativo all'eliminazione gaussiana.
A=[2 1 0; 1 2 1; 0 1 2];
b=[3 4 3]';

[L,U]=fattorizzazione_LU(A);
y=L\b;
x=U\y
```

Il file necessita di pochi commenti.

- Abbiamo descritto la matrice A e il termine noto b mediante

```
A=[2 1 0; 1 2 1; 0 1 2];
b=[3 4 3]';
```

  Si osservi che per quanto riguarda A, il passaggio da una riga alla successiva è dovuto alla presenza di `;`.

- Nell'ultima assegnazione non abbiamo messo il `;` così da scrivere in command-window i valori assunti da `x` e verificare che offrono la soluzione esatta.

2.2. *Le routines* `metodo_GS` *e* `demo_GS`

Si modifichino i files `metodo_jacobi` e `demo_jacobi`, così da definire `metodo_GS` e `demo_GS`, che al posto di Jacobi, implementino e testino il metodo di Gauss-Seidel. A ta proposito, si usi la sua descrizione matriciale, coinvolgente le matrici M e N.

Nello svolgimento della routine, si utilizzino i comandi Matlab `tril` e `triu` (aiutarsi con l'help di Matlab).

2.2.1. *Svolgimento*

Il file `metodo_GS` è uguale a quello di `metodo_jacobi` eccetto per quanto riguarda le matrici P e N.

```
function [x,errs,iter,flag]=metodo_GS(A,x,b,maxit,tol)

% oggetto:
% la routine risolve il sistema lineare Ax=b mediante
% il metodo di Gauss-Seidel
%
% input:
% A: matrice quadrata non singolare
% x: approssimazione iniziale della soluzione
%     (vettore colonna)
% b: termine noto (vettore colonna)
% maxit: numer massimo di iterazioni
% tol: tolleranza del metodo di Gauss-Seidel
%
% output:
% x: approssimazione della soluzione, fornita dal
%      metodo di Gauss-Seidel.
% errs: norme dell'errore
%                  norm(x_new-x_old)/norm(x_new)
%     al variare delle iterazioni, ovvero lo step
%     relativo.
% iter: numero di iterazioni del metodo
% flag: 0: si e' raggiunta l'approssimazione
%             desiderata della soluzione
%       1: non si e' raggiunta l'approssimazione
%             desiderata della soluzione

% inizializzazioni
flag=0;


% --- le matrici P, N per Gauss-Seidel ---

P=tril(A,-1)+diag(diag(A));
% se il determinante di P e' uguale a "1", allora
% il metodo di Gauss-Seidel non si puo' applicare in
%  quanto qualche componente A(k,k)=0;
% in effetti P e' una matrice triangolare e
% il determinante di P e' il prodotto degli elementi
% sulla diagonale per cui e' nullo sse qualche
% A(k,k)=0;
if det(P) == 0
    errs=[]; iter=0; flag=1;  return;
end
N=-triu(A,1);


% --- iterazioni del metodo di Gauss-Seidel ---

for iter=1:maxit
    x_old=x;
    x=P\(N*x_old+b);   % nuova iterazione
    % calcolo step relativo
    errs(iter)=norm(x-x_old)/norm(x);
    % se error e' suff. piccolo si esce dalla routine
    if (errs(iter)≤tol), return, end
end

% se abbiamo raggiunto questo punto abbiamo fatto
%  troppe iterazioni senza successo e quindi
% poniamo "flag=1".
flag=1;
```

Vediamo i punti relativi alla definizione di `P` e `N`, mediante `tril` e `triu`.

```
>> 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.

    Reference page for tril
    Other functions named tril

>> 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.

    Reference page for triu
    Other functions named triu

>>
```

Ricordiamo che $A = D - E - F$ con

1. D matrice diagonale,
2. E triangolare inferiore,
3. F triangolare superiore.

Allora il metodo di Gauss-Seidel corrisponde a scegliere, se D è invertibile, $P = D - E$, $N = F$.

Seguendo l'help di `tril` e `triu`, con un po' di pazienza ricaviamo

- D è `diag(diag(A);`
- E è `-tril(A,-1);`
- F è `-triu(A,+1);`

Per convincersene, da command-window

```
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> % estrazione della parte diagonale
>> D=diag(diag(A))
D =
     1     0     0
     0     5     0
     0     0     9
>> % estrazione della matrice triang. inf.
>> % di A, con diag. nulla, cambiata di segno.
>> E=-tril(A,-1)
E =
     0     0     0
    -4     0     0
    -7    -8     0
>> % estrazione della matrice triang. sup.
>> % di A, con diag. nulla, cambiata di segno.
>> F=-triu(A,+1)
F =
     0    -2    -3
     0     0    -6
     0     0     0
>> % verifichiamo che A=D-E-F.
>> D-E-F
ans =
     1     2     3
     4     5     6
     7     8     9
>>
```

Detto questo, da

```
D=diag(diag(A),

E=-tril(A,-1),

F=-triu(A,+1)
```

abbiamo

- $P = D - E$ e quindi scriveremo

```
P=diag(diag(A)+tril(A,-1)
```

- $N = -F$ e quindi otteniamo

```
N=triu(A,+1)
```

La routine `demo_GS` è essenzialmente `demo_jacobi` eccetto per la chiamata della routine `metodo_GS`.

Dalla sua applicazione ricaviamo

```
>> demo_GS

   dimensione matrice:    100
   numero iterazioni :    137
   flag              :      0
   residuo relativo  : 2.71e-06

>>
```

Si osservi che in questo caso, il metodo di Gauss-Seidel ha effettuato circa metà delle iterazioni fatte dal metodo di Jacobi, per soddisfare lo stesso criterio di arresto.