Francesca Mazzia
Dipartimento Interuniversitario di Matematica
Università di Bari

MATLAB:Soluzione Sistemi Lineari.

Soluzione sistemi triangolari

La seguente funzione risolve i sistemi triangolari inferiori

```
function b = soll(L,b)
%%
%% dati di input
%%        L matrice triangolare inferiore
%%        b termine noto
%% output
%%        b soluzione
n = length(b);
b(1) = b(1)/L(1,1);
for i=2:n
  b(i) = (b(i) - L(i,1:i-1)*b(1:i-1))/L(i,i);
end
```

Per eseguirla in Matlab eseguiamo le seguenti istruzioni:

```
>> L = [ 1  0 0 0
         2 4  0 0
         3 6 5 0
```

```
         1 2 3 4]

L =

     1     0     0     0
     2     4     0     0
     3     6     5     0
     1     2     3     4

>> b = [1;2;3;4]

b =

     1
     2
     3
     4

>> x=soll(L,b)

x =

    1.0000
         0
         0
    0.7500

>> L*x

ans =

     1
     2
     3
     4

>> b
```

```
b =

     1
     2
     3
     4

>> L*x-b

ans =

     0
     0
     0
     0

>> norm(L*x-b,'inf')

ans =

     0

>>
```

Algoritmo di eliminazione di Gauss.

Eseguiamo in Matlab tutti i passaggi per costruire la fattorizzazione

```
>> clear P A L
>>A = [0     2     4     2
     6     8     9     8
     4     0     0     5
     6     2     4     7];

>>A0 = A;

>>P = eye(4);
```

```
>>b = [ 1; 2; 3; 4];

% scegliamo il pivot

>>[elm,indm]=max(abs(A(1:4,1)))

elm =

     6


indm =

     2
% scambiamo le righe di A
>>A([1 indm],:)=A([indm,1],:)

A =

     6     8     9     8
     0     2     4     2
     4     0     0     5
     6     2     4     7

% scambiamo le righe di P e di b
>>P([1 indm],:)=P([indm,1],:)

P =

     0     1     0     0
     1     0     0     0
     0     0     1     0
     0     0     0     1

>>b([1 indm])=b([indm,1])

b =
```

```
     2
     1
     3
     4

% calcolo gli elementi di L :  L(2:4,1)=A(2:4,1)/A(1,1)
>>L(2:4,1)=A(2:4,1)/A(1,1)

L =

         0
         0
    0.6667
    1.0000

% aggiorno la matrice A
%  aggiorno la seconda riga
>>A(2,:)=A(2,:)-L(2,1)*A(1,:)

A =

     6     8     9     8
     0     2     4     2
     4     0     0     5
     6     2     4     7

% aggiorno la terza riga
>>A(3,:)=A(3,:)-L(3,1)*A(1,:)

A =

    6.0000    8.0000    9.0000    8.0000
         0    2.0000    4.0000    2.0000
         0   -5.3333   -6.0000   -0.3333
    6.0000    2.0000    4.0000    7.0000

% aggiorno la quarta  riga
>>A(4,:)=A(4,:)-L(4,1)*A(1,:)
```

```
A =

    6.0000    8.0000    9.0000    8.0000
         0    2.0000    4.0000    2.0000
         0   -5.3333   -6.0000   -0.3333
         0   -6.0000   -5.0000   -1.0000

% aggiorno il vettore b

>>b(2:4)=b(2:4)-L(2:4,1)*b(1)

b =

    2.0000
    1.0000
    1.6667
    2.0000
% adesso ripeto il procedimento sulla sottomatrice A(2:4,2:4)
% scelgo l'elemento pivotale
>>[elm,indm]=max(abs(A(2:4,2)))

elm =

     6


indm =

     3

>>indm=indm+1

indm =

     4

%scambio le righe di A
```

```
>>A([2 indm],:)=A([indm 2],:)

A =

    6.0000    8.0000    9.0000    8.0000
         0   -6.0000   -5.0000   -1.0000
         0   -5.3333   -6.0000   -0.3333
         0    2.0000    4.0000    2.0000

% scambio le righe di P
>>P([2 indm],:)=P([indm 2],:)

P =

     0     1     0     0
     0     0     0     1
     0     0     1     0
     1     0     0     0


% scambio le righe di b
>>b([2 indm],:)=b([indm 2])

b =

    2.0000
    2.0000
    1.6667
    1.0000
% scambio le righe di L
>>L([2 indm],1)=L([indm 2],1)


L =

         0
    1.0000
```

```
    0.6667
         0


% calcolo gli elementi della seconda colonna di L
>>L(3:4,2)=A(3:4,2)/A(2,2)

L =

         0         0
         0         0
    0.6667    0.8889
    1.0000   -0.3333

% aggiorno la matrice A:  terza e quarta riga
>>A(3:4,:)=A(3:4,:)-L(3:4,2)*A(2,:)

A =

    6.0000    8.0000    9.0000    8.0000
         0   -6.0000   -5.0000   -1.0000
         0         0   -1.5556    0.5556
         0         0    2.3333    1.6667

% aggiorno il vettore b
>>b(3:4)=b(3:4)-L(3:4,2)*b(2)

b =

    2.0000
    2.0000
   -0.1111
    1.6667

% eseguo l'ultimo passaggio
% calcolo elemento pivotale
>>[elm,indm]=max(abs(A(3:4,3)))
```

```
elm =

    2.3333


indm =

     2

>>indm=indm+2

indm =

     4

% scambio le righe di A
>>A([3 indm],:)=A([indm 3],:)

A =

    6.0000    8.0000    9.0000    8.0000
         0   -6.0000   -5.0000   -1.0000
         0         0    2.3333    1.6667
         0         0   -1.5556    0.5556

% scambio le righe di P
>>P([3 indm],:)=P([indm 3],:)


P =

     0     1     0     0
     0     0     0     1
     1     0     0     0
     0     0     1     0

% scambio le righe di b
>>b([3 indm],:)=b([indm 3])
```

```
b =

    2.0000
    2.0000
    1.6667
   -0.1111
% scambio le righe di L
>>L([3 indm],:)=L([indm 3],:)

L =

         0         0
    1.0000         0
         0   -0.3333
    0.6667    0.8889


% calcolo gli elementi di L
>>L(4:4,3)=A(4:4,3)/A(3,3)

L =
         0         0         0
    1.0000         0         0
         0   -0.3333         0
    0.6667    0.8889   -0.6667


% aggiorno la A
>>A(4:4,:)=A(4:4,:)-L(4:4,3)*A(3,:)

A =

    6.0000    8.0000    9.0000    8.0000
         0   -6.0000   -5.0000   -1.0000
         0         0    2.3333    1.6667
         0         0         0    1.6667
```

```
% aggiorno la b
>>b(4)=b(4)-L(4,3)*b(3)

b =

    2.0000
    2.0000
    1.6667
    1.0000

% ho ottenuto un sistema triangolare superiore
% completo la matrice L

>>L(:,4)=0

L =

    0         0         0         0
    1.0000    0         0         0
         0   -0.3333    0         0
    0.6667    0.8889   -0.6667    0

>>L=L+eye(4)

L =

    1.0000         0         0         0
    1.0000    1.0000         0         0
         0   -0.3333    1.0000         0
    0.6667    0.8889   -0.6667    1.0000


% verifico la fattorizzazione
>>P*A0

ans =

     6     8     9     8
```

```
     6     2     4     7
     0     2     4     2
     4     0     0     5

>>L*A

ans =

    6.0000    8.0000    9.0000    8.0000
    6.0000    2.0000    4.0000    7.0000
         0    2.0000    4.0000    2.0000
    4.0000    0.0000    0.0000    5.0000
```

Costruiamo una funzione Matlab che ci permette di calcolare la fattorizzazione LU mediante il metodo di eliminazione di Gauss senza eseguire il pivot

```
function [L,U] = gauss(A)

% ELIMINAZIONE DI GAUSS
% Dati di input:
%   A = matrice quadrata
% Dati di output:
%   L = matrice triangolare inferiore
%   U = matrice triangolare superiore
% tali che A = LU

[n,m] = size(A);

if n ~= m
  error('ERRORE: la matrice di input non e'' quadrata')
end

for k = 1:n-1
```

```
% controllo sulla grandezza dell'elemento diagonale:
  if abs(A(k,k)) <  realmin
    disp('Un elemento diagonale e'' nullo')
    error('Non e'' possibile continuare il procedimento di eliminazione')
  end
  A(k+1:n,k) = A(k+1:n,k)/A(k,k);
  A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - ...
  A(k+1:n,k)*A(k,k+1:n);
end
L = tril(A,-1) + eye(n);
U = triu(A);
```

Problemi dell'algoritmo di fattorizzazione LU senza permutazioni:

1) Non può essere eseguita su matrici come:

```
>> Q=[0 1 0;0 0 1;1 0 0]

Q =

     0     1     0
     0     0     1
     1     0     0

>> A=[1 2 5;-1 -2 -7;1 1 3]

A =

     1     2     5
    -1    -2    -7
     1     1     3
```

Infatti se facciamo

```
>> [L,U]=gauss(Q)
```

e

```
>> [L,U]=gauss(A)
```

abbiamo

```
Un elemento diagonale e' nullo
??? Error using ==> gauss
Non e' possibile continuare il procedimento di eliminazione
```

Possiamo risolvere tale problema utilizzando la funzione Matlab che esegue la fattorizzazione LU con pivot che si chiama lu;

```
>> A=[1 2 5;-1 -2 -7;1 1 3]

A =

     1     2     5
    -1    -2    -7
     1     1     3

>> [L,U,P]=lu(A)

L =

     1     0     0
     1     1     0
    -1     0     1
```

```
U =

     1     2     5
     0    -1    -2
     0     0    -2


P =

     1     0     0
     0     0     1
     0     1     0
```

da cui abbiamo

```
>> P*A

ans =

     1     2     5
     1     1     3
    -1    -2    -7

>> L*U

ans =

     1     2     5
     1     1     3
    -1    -2    -7
```

Analogamente:

```
>> Q=[0 1 0;0 0 1;1 0 0]

Q =

     0     1     0
     0     0     1
     1     0     0

>> [L,U,P]=lu(Q)

L =

     1     0     0
     0     1     0
     0     0     1


U =

     1     0     0
     0     1     0
     0     0     1


P =

     0     0     1
     1     0     0
     0     1     0
```

da cui

```
>> P*Q

ans =
```

```
     1     0     0
     0     1     0
     0     0     1

>> L*U

ans =

     1     0     0
     0     1     0
     0     0     1
```

Vediamo anche come costruire l'algoritmo per calcolare la fattorizzazione LU con il metodo di eliminazione di Gauss con il pivot parziale:

```
function [L,U,P] = gaussp(A)

% ELIMINAZIONE DI GAUSS CON PIVOT PARZIALE
% Dati di input:
%   A = matrice quadrata
% Dati di output:
%   L = matrice triangolare inferiore
%   U = matrice triangolare superiore
%   P = matrice di permutazione
% tali che PA = LU

[n,m] = size(A);
if n ~= m
   error('ERRORE: la matrice di input non e'' quadrata')
end
P = eye(n);
for k = 1:n-1

% ricerca del pivot e permutazione:
% si scambiano le righe k e rp di A e di P
```

```
   [piv,rp] = max(abs(A(k:n,k)));
   rp = rp + k - 1;
   P([k rp],:)=P([rp k],:)
   A([k rp],:)=A([rp k],:)


% controllo sulla grandezza dell'elemento pivotale:

if abs(A(k,k)) <= realmin
      disp('Un elemento pivotale e'' nullo')
      error('La matrice in input e'' singolare')
   end
   A(k+1:n,k) = A(k+1:n,k)/A(k,k);
   A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - A(k+1:n,k)*A(k,k+1:n);
end

L = tril(A,-1) + eye(n);
U = triu(A);
```

Anche mediante tale funzione possiamo risolvere il problema precedente; infatti:

```
>> A=[1 2 5;-1 -2 -7;1 1 3]

A =

     1     2     5
    -1    -2    -7
     1     1     3


>> [L,U,P]=gaussp(A)

L =
```

```
     1     0     0
     1     1     0
    -1     0     1


U =

     1     2     5
     0    -1    -2
     0     0    -2


P =

     1     0     0
     0     0     1
     0     1     0
```

e

```
>> Q=[0 1 0;0 0 1;1 0 0]

Q =

     0     1     0
     0     0     1
     1     0     0

>> [L,U,P]=gaussp(Q)

L =

     1     0     0
     0     1     0
     0     0     1
```

```
U =

     1     0     0
     0     1     0
     0     0     1


P =

     0     0     1
     1     0     0
     0     1     0
```

2) La fattorizzazione LU senza pivot Presenta problemi di instabilità numerica:

$$\begin{cases} 10^{-17}x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{cases}$$

Il problema è ben posto, ed ha come soluzione esatta $x \simeq (1,1)^T$.

Infatti ponendo:

```
 >> A=[1e-17 1;1 1]

A =

   1.0000e-17   1.0000e+00
   1.0000e+00   1.0000e+00

>> b=[1;2]

b =
```

```
     1
     2
```

abbiamo:

```
>> x=A\b

x =

     1
     1
```

Applicando il metodo di Gauss ed operando in virgola mobile in base 2 in doppia precisione, si ottengono i fattori L ed U seguenti:

```
>> [L,U]=gauss(A)


L =

   1.0000e+00             0
   1.0000e+17   1.0000e+00


U =

   1.0000e-17   1.0000e+00
            0  -1.0000e+17
```

Il prodotto LU non è A e infatti:

```
>> A-L*U
```

```
ans =

     0     0
     0     1
```

Ciò significa che la soluzione numerica ottenuta con il metodo di Gauss senza pivot sarà totalmente sbagliata.

Infatti risolvendo $Ly = b$ e $Uxtilde = y$ abbiamo:

```
>> y=L\b

y =

   1.0000e+00
  -1.0000e+17

>> xtilde=U\y

xtilde =

     0
     1
```

Se invece utilizziamo il metodo di Gauss con il pivot parziale abbiamo:

```
 >> A=[1e-17 1;1 1]

A =

   1.0000e-17   1.0000e+00
   1.0000e+00   1.0000e+00
```

```
>> [L,U,P]=gaussp(A)

L =
  1.0000e+000            0
  1.0000e-017  1.0000e+000

U =
     1     1
     0     1

P =
     0     1
     1     0
```

da cui

```
>> P*A-L*U

ans =

     0     0
     0     0

>> y=L\P*b
y =

     2
     1

>> xtilde=U\y
xtilde =

     1
     1
```