Soluzione di un sistema di equazioni lineari

1 giugno 2008

1 Soluzione di un sistema di equazioni lineari

Si consideri il sistema lineare di m equazioni in n incognite

$$\begin{array}{l} a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + \ldots + a_{1,n} \cdot x_n = b_1 \\ a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 + \ldots + a_{2,n} \cdot x_n = b_2 \\ a_{3,1} \cdot x_1 + a_{3,2} \cdot x_2 + \ldots + a_{3,n} \cdot x_n = b_3 \\ \ldots \\ a_{m,1} \cdot x_1 + a_{n,2} \cdot x_2 + \ldots + a_{n,n} \cdot x_n = b_m \end{array} \tag{1}$$

Definita la matrice

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ a_{3,1} & a_{3,2} & \ldots & a_{3,n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,n} \end{pmatrix}$$

e i vettori colonna delle incognite x e dei termini noti b

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \ldots \\ x_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \ldots \\ b_m \end{pmatrix}$$

il sistema di equazioni (3) si può riscrivere come

$$A \cdot x = b$$

dove $\cdot$ è l'usuale prodotto matrice-vettore per cui $(A \cdot x)_i = \sum_{j=1}^{n} a_{i,j} \cdot x_j$ con $(A \cdot x)_i$ la i-sima componente del vettore $A \cdot x$.

1.1 Risolubilità di un sistema di equazioni lineari

Ci si pone il problema se esista e sia unica la soluzione del sistema lineare $A \cdot x = b$. A tal proposito distinguiamo alcuni casi

1. $m = n$: in tal caso la matrice A è quadrata di ordine n e la soluzione $x^* \in \mathbb{R}$ esiste ed è unica se e solo se la matrice è invertibile (che è equivalente a dire che il determinante della matrice A è diverso da 0); per calcolare in Matlab il determinante di una matrice A si usa la riga di comando `det(A);`

2. $m > n$ oppure $m < n$: questa volta la matrice è rettangolare e per la discussione generale si considera il teorema di Rouchè -Capelli, descritto in

   vspace0.2cm http://it.wikipedia.org/wiki/Teorema_di_Rouch%C3%A9-Capelli

Per il momento, a meno di ipotesi contrarie, considereremo solo matrici quadrate. Inoltre sottintenderemo come d'uso il $\cdot$ nel prodotto tra matrici con matrici e matrici con vettori. Così scriveremo AB invece di $A \cdot B$ e Ax invece di $A \cdot x$.

1.2 Metodo numerico di risoluzione del sistema lineare, con la fattorizzazione LU

Supponiamo che esistano due matrici quadrate di ordine n, diciamo L ed U, rispettivamente triangolare inferiore e superiore, tali che $A = LU$. Questa scomposizione di A in fattori, è nota come fattorizzazione LU (L sta per *lower*, U sta per *upper*). Ricordiamo che una matrice quadrata $A = (a_{i,j})$ è triangolare inferiore se del tipo

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,n} \\ 0 & 0 & \dots & a_{3,n} \\ \dots & 0 & \dots & \dots \\ 0 & 0 & 0 & a_{n,n} \end{pmatrix}$$

mentre è triangolare inferiore se ha la forma

$$A = \begin{pmatrix} a_{1,1} & 0 & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & 0 \\ a_{3,1} & a_{3,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix}$$

Ciò significa che tutte le componenti rispettivamente sotto e sopra la diagonale sono uguali a 0.

Essendo $A = LU$ e contemporaneamente $Ax = b$, necessariamente $LUx = b$. Posto $y = Ux$, si ha $Ly = b$. Se y^* è la soluzione di questo sistema lineare di tipo triangolare allora $Ax = b$ se e solo se $Ux = y^*$. Quindi una volta risolto il sistema lineare di tipo triangolare superiore $Ux = y^*$, si è calcolata pure la soluzione del sistema $Ax = b$. Visivamente si risolvono di seguito

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

In altre parole, se A è una matrice invertibile e $A = LU$, si può calcolare l'unica soluzione del sistema lineare $Ax = b$ risolvendo due sistemi lineari di tipo triangolare. Osserviamo che essendo L triangolare inferiore, il vettore y^* può essere facilmente ricavato con il metodo delle *sostituzioni in avanti*. In modo simile, essendo U triangolare superiore, noto y^*, il vettore x^* può essere facilmente ricavato con il metodo delle *sostituzioni all'indietro*.

Notiamo che non tutte le matrici posseggono una fattorizzazione LU. Ad esempio per la matrice (di permutazione)

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

non esistono L, U con le proprietà sopra citate tali che $A = LU$.

Comunque, qualora esista tale fattorizzazione si può provare che senza ulteriori richieste, esistono infinite fattorizzazioni LU. Sia ad esempio $D_1 = \alpha I$, $D_2 = \alpha^{-1} I$, per $\alpha \neq 0$. Se $A = LU$, per $L_1 = D_1 L$ e $U_1 = D_2 U$ abbiamo pure $A = L_1 U_1$ con L_1, U_1 che sono rispettivamente triangolare inferiore e superiore.

Si dimostra che se A ammette una fattorizzazione LU, essa è unica se si assume che L abbia tutte le componenti diagonali $l_{i,i}$ uguali a 1 (fattorizzazione di Doolittle) o, alternativamente, che U abbia tutte le componenti diagonali $u_{i,i}$ uguali a 1 (fattorizzazione di Crout). Vediamo quale delle due alternative è implementata in Matlab. Eseguiamo il comando `help LU` dalla shell di Matlab/Octave. Otteniamo:

```
>> help LU

 LU     LU factorization.
    [L,U] = LU(X) stores an upper triangular matrix in U and a
    "psychologically lower triangular matrix" (i.e. a product
    of lower triangular and permutation matrices) in L, so
    that X = L*U. X can be rectangular.

    [L,U,P] = LU(X) returns lower triangular matrix L, upper
    triangular matrix U, and permutation matrix P so that
    P*X = L*U.

    LU(X), with one output argument, returns the output from
    LAPACK'S DGETRF or ZGETRF routine.

    LU(X,THRESH) controls pivoting in sparse matrices, where
    THRESH is a pivot threshold in [0,1].  Pivoting occurs
    when the diagonal entry in a column has magnitude less than
    THRESH times the magnitude of any sub-diagonal entry in that
    column.  THRESH = 0 forces diagonal pivoting.  THRESH = 1 is
    the default.

    See also LUINC, QR, RREF.

>>
```

Non dice molto, eccetto che la sintassi è

```
[L,U,P] = LU(X)
```

In questo caso $PA = LU$ con P matrice di permutazione (vedremo dopo il significato). Proviamo quale esempio

```
>> A=[ 7 8 9; 1 2 3; 4 5 6]

A =
     7     8     9
     1     2     3
     4     5     6

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

L =
    1.0000         0         0
    0.1429    1.0000         0
    0.5714    0.5000    1.0000


U =
    7.0000    8.0000    9.0000
         0    0.8571    1.7143
         0         0    0.0000

P =
     1     0     0
     0     1     0
     0     0     1

>>
```

Come previsto la matrice L è triangolare inferiore, U triangolare superiore e il semplice comando `L*U` mostra che in effetti

```
>> L*U
ans =
     7     8     9
     1     2     3
     4     5     6
>>
```

Visto che P è la matrice identica allora $PA = A$ che evidentemente coincide con LU.

Osserviamo che la diagonale di L ha componenti uguali a 1. Supporremo quindi sia $l_{i,i} = 1$ per $i = 1, \ldots, n$.

1.2.1 Un implementazione Matlab del metodo LU

Vediamo un implementazione in Matlab di quanto visto, per risolvere il problema $Ax = b$, dove al solito supporremo A quadrata e non singolare. Come detto precedentemente si tratta di calcolare, se esiste, la fattorizzazione $A = LU$ della matrice A per poi risolvere i problemi rispettivamente in avanti e all'indietro

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Definiamo le routines

```
function [x]=sostit_indietro(U,b)
% Risoluzione di un sistema lineare Ux=b con le sostituzioni
% all'indietro: versione per righe.

n = size(U,1);
x = zeros(n,1);
x(n,1) = b(n)/U(n,n);
for k=(n-1):-1:1
  x(k,1)=(b(k) - sum(U(k,k+1:n).*x(k+1:n,1)'))/U(k,k);
end;


function [x]=sostit_avanti(L,b)
% Risoluzione di un sistema lineare Lx=b con le sostituzioni in
% avanti: versione per righe.

n = size(L,1);
x = zeros(n,1);
x(1,1) = b(1)/L(1,1);
for k=2:n
  x(k,1)=(b(k) - sum(L(k,1:k-1).*x(1:k-1,1)'))/L(k,k);
end;
```

Nota. Non è molto chiaro il senso del ciclo for nelle due routines `sostit_avanti`, `sostit_indietro`. Vediamo il caso di una sostituzione in avanti, applicandola alla risoluzione del sistema lineare $Ax = b$, dove $b = [1;1;1]$ e

$$\begin{matrix} 1/2 & 0 & 0 \\ 2 & 1/6 & 0 \\ 4 & 8 & 2 \end{matrix} \tag{2}$$

cioè al problema

$$\begin{matrix} (1/2)\cdot x_1 = 1 \\ 2\cdot x_1 + (1/6)\cdot x_2 = 1 \\ 4\cdot x_1 + 8\cdot x_2 + 2\cdot x_3 = 1 \end{matrix} \tag{3}$$

Essendo la matrice A triangolare inferiore, usiamo l'algoritmo della sostituzione in avanti, eseguendo la chiamata

```
[x]=sostit_avanti(A,b)
```

- Analizziamo la prima riga

  ```
  n = size(L,1);
  ```

  Essendo A una matrice 3×3, abbiamo $n = 3$.

- La riga

  ```
  x(1,1) = b(1)/L(1,1);
  ```

  non fa altro che risolvere l'equazione

  $$(1/2) \cdot x_1 = 1$$

  e pone in $x(1,1)$ il valore $1/(1/2) = 2$.

- Entriamo nel ciclo for

  ```
  for k=2:n
    x(k,1)=(b(k) - sum(L(k,1:k-1).*x(1:k-1,1)'))/L(k,k);
  end;
  ```

  per $k = 2$. Se facessimo i conti a mano vorremmo risolvere la seconda equazione per sostituzione come

  ```
  2*x_1 + (1/6)*x_2  =1
  2*2   + (1/6)*x_2  =1
  4     + (1/6)*x_2  =1
  (1/6)*x_2  =1-4
  (1/6)*x_2  =-3
  x_2        =(-3)/(1/6)
  x_2        =(-18)
  ```

  Vediamo ora il codice. Per $k = 2$, si ha, essendo in Matlab `1:1=1` e `'` l'operatore di trasposizione, ricordato che nella chiamata $L = A$, $A(2,1) = 2$, $A(2,2) = (1/6)$, $x(1,1) = 2$

  ```
  x(k,1)=(b(k) - sum(L(k,1:k-1).*x(1:k-1,1)'))/L(k,k);
  x(2,1)=(b(2) - sum(L(2,1:2-1).*x(1:2-1,1)'))/L(2,2);
  x(2,1)=(b(2) - sum(L(2,1).*x(1,1)'))/L(2,2);
  x(2,1)=(1 - sum(A(2,1).*x(1,1)'))/A(2,2);
  x(2,1)=(1 - sum(2*2))/(1/6);
  x(2,1)=(-3)/(1/6);
  x(2,1)=(-18);
  ```

- Entriamo nel ciclo for

  ```
  for k=2:n
    x(k,1)=(b(k) - sum(L(k,1:k-1).*x(1:k-1,1)'))/L(k,k);
  end;
  ```

per $k = 3$. Se facessimo i conti a mano vorremmo risolvere la terza equazione per sostituzione (per x_1=2 e x_2=18) come

```
4*x_1+8*x_2+2*x_3=1
(4*2+8*18)+2*x_3=1
2*x_3=1-(4*2+8*18)
x_3=( 1-(4*2+8*18))/2
x_3=(1-sum([(4*2) (8*18)]))/2
x_3=(1-sum([8 144]))/2
x_3=(1-152)/2
x_3=-151/2
```

Vediamo cosa fa il ciclo for per $k = 3$, ricordando che in Matlab 1:2=[1 2], $L = A$, $L(3,1:2) = A(3,1:2) = [4\ 8]$, $x(1:2,1)' = [2\ 18]$, $L(3,3) = 2$, $b(3) = 1$

```
x(k,1)=(b(k) - sum(L(k,1:k-1).*x(1:k-1,1)'))/L(k,k);
x(3,1)=(b(3) - sum(L(3,1:3-1).*x(1:3-1,1)'))/L(3,3);
x(3,1)=(b(3) - sum(L(3,1:2).*x(1:2,1)'))/L(3,3);
x(3,1)=(1 - sum([4 8].*[2 18]))/2;
x(3,1)=(1 - sum([4*2 8*18]))/2;
x(3,1)=(1 - sum([8 144]))/2;
x(3,1)=(1 - 152)/2;
x(3,1)=(-151)/2;
```

Qualora A possegga una fattorizzazione LU, le matrici L ed U possono essere ottenute tramite la seguente funzione Matlab/Octave

```
%  Fattorizzazione LU "sul posto" della matrice A (A=L*U) con la
%  sequenza "kji"
%
%  "sul posto" significa che i fattori L ed U sono ancora contenuti in A .
%
function [A] = fattLU(A)

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

Come detto nell'help, *fattorizzazione sul posto* vuol dire che durante la fattorizzazione LU, l'immagazzinamento parziale delle matrici L ed U non è eseguito in due ulteriori matrici L ed U, ma nello spazio precedentemente usato rispettivamente nella parte triangolare inferiore e superiore della matrice A. Questo era importante anni fa per poter risolvere sistemi lineari di non piccole dimensioni pur essendo ai limiti della memoria del computer.

Una variante più complessa e stabile è quella vista con il comando `lu` di Matlab/Octave.

Come commentato nel codice, L ed U si ottengono dalla matrice A in output prendendo la parte triangolare inferiore di A e la parte triangolare superiore (imponendo che gli elementi diagonali di U siano uguali a 1).

Vediamo un esempio

```
>> A=hilb(3)
A =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
>> b=[1; 0; 0]
b =
     1
     0
     0
>> % RISOLVIAMO IL PROBLEMA Ax=b CON IL METODO LU.
>> [A1] = fattLU(A)
A1 =
    1.0000    0.5000    0.3333
    0.5000    0.0833    0.0833
    0.3333    1.0000    0.0056
>>
>> % L E' LA MATRICE CHE SI OTTIENE DALLA PARTE TRIANGOLARE
>> % INFERIORE, TOGLIENDO LA DIAGONALE E AGGIUNGENDO LA
>> % MATRICE IDENTICA.
>> % tril(A) E' LA MATRICE TRIANGOLARE INFERIORE ESTRATTA
>> % ED INCLUDE LA DIAGONALE.
>>
>> L=tril(A1)-diag(diag(A1))+eye(3);
>>
>> % triu(A) E' LA MATRICE TRIANGOLARE SUPERIORE ESTRATTA
>> % ED INCLUDE LA DIAGONALE.
>> U=triu(A1);
>>
>> A
A =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
>> L*U
ans =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
>>
>> [y]=sostit_avanti(L,b);
>> [x]=sostit_indietro(U,y);
>> A*x-b
```

```
ans =
  1.0e-015 *
         0
   -0.8882
    0.8882
>> x
x =
    9.0000
  -36.0000
   30.0000
>>
```

La scrittura

```
>> A*x-b
ans =
  1.0e-015 *
         0
   -0.8882
    0.8882
```

indica il vettore colonna

$$1 \cdot 10^{-15} \begin{pmatrix} 0.0000 \\ -0.8882 \\ 0.8882 \end{pmatrix}.$$

Il comando `diag(diag(A1))` calcola la matrice diagonale associata alla matrice A1. Così

```
>> A1=[1 2 3; 4 5 6; 7 8 9]
A1 =
     1     2     3
     4     5     6
     7     8     9
>> diag(A1)
ans =
     1
     5
     9
>> diag(diag(A1))
ans =
     1     0     0
     0     5     0
     0     0     9
>>
```

Ricordiamo che `triu` e `tril` estraggono la parte triangolare superiore e inferiore di una matrice. Vediamo un esempio di queste due funzioni

```
>> A=hilb(3)
A =
    1.0000    0.5000    0.3333
    0.5000    0.3333    0.2500
    0.3333    0.2500    0.2000
>> triu(A)
ans =
    1.0000    0.5000    0.3333
         0    0.3333    0.2500
         0         0    0.2000
>> tril(A)
ans =
    1.0000         0         0
    0.5000    0.3333         0
    0.3333    0.2500    0.2000
>>
```

1.3 Tecnica del pivoting

Proviamo il seguente esperimento

```
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

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

L =

    1.0000         0         0
    0.1429    1.0000         0
    0.5714    0.5000    1.0000


U =

    7.0000    8.0000    9.0000
         0    0.8571    1.7143
         0         0    0.0000


P =

     0     0     1
     1     0     0
```

```
     0     1     0

>>
```

notiamo che Matlab/Octave forniscono una fattorizzazione $PA = LU$ con P che non è la matrice identica. Viene da chiedersi quali siano i vantaggi. In molti casi questo è dovuto alla maggiore stabilità dell'algoritmo ma a volte per problemi strettamente matematici: come anticipato, non tutte le matrici A posseggono una fattorizzazione $A = LU$.
Ad esempio la matrice (detta di *permutazione*)

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

non possiede la fattorizzazione LU (lo si verifichi manualmente).

Se proviamo la fattorizzazione LU in Matlab (senza pivoting) otteniamo

```
>> A=[0 1; 1 0]
A =
     0     1
     1     0
>> [L,U]=lu(A)
L =
     0     1
     1     0
U =
     1     0
     0     1
>> L*U
ans =
     0     1
     1     0
>>
```

ma evidentemente la matrice L non è triangolare inferiore. Bisogna quindi sempre indicare chi sia la matrice P, avendo

```
>> A=[0 1; 1 0]
A =
     0     1
     1     0
>> [L,U,P]=lu(A)
L =
     1     0
     0     1
U =
     1     0
     0     1
P =
     0     1
```

```
     1     0

>> P*A
ans =
     1     0
     0     1
>> L*U
ans =
     1     0
     0     1
>>
```

Ci si domanda, se $PA = LU$, è possibile modificare il metodo LU in un metodo che chiameremo *metodo LU con pivoting*, per risolvere sistemi lineari del tipo $Ax = b$? Osserviamo che se $Ax = b$ allora $PAx = Pb$ da cui $LUx = Pb$. Quindi basta risolvere i due sistemi triangolari

$$\begin{cases} Ly = Pb \\ Ux = y. \end{cases}$$

Vediamone un esempio che ben illustra il metodo LU con pivoting. Si consideri il sistema lineare (tratto da [5], p.127)

$$\begin{aligned} x_1 + 2\,x_2 + x_3 + 4\,x_4 &= 13 \\ 2\,x_1 + 0\,x_2 + 4\,x_3 + 3\,x_4 &= 28 \\ 4\,x_1 + 2\,x_2 + 2\,x_3 + x_4 &= 20 \\ -3\,x_1 + x_2 + 3\,x_3 + 2\,x_4 &= 6 \end{aligned} \tag{4}$$

Riscritto matricialmente per

$$A = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 2 & 0 & 4 & 3 \\ 4 & 2 & 2 & 4 \\ -3 & 1 & 3 & 2 \end{pmatrix} \tag{5}$$

e

$$b = \begin{pmatrix} 13 \\ 28 \\ 20 \\ 6 \end{pmatrix} \tag{6}$$

si cerca l'unico x^* tale che $Ax^* = b$.
Usando il backslash \ solo per risolvere sistemi triangolari, il problema può essere risolto come segue

```
>> A=[1 2 1 4; 2 0 4 3; 4 2 2 4; -3 1 3 2];
>> b=[13; 28; 20; 6];
>> [L,U,P]=lu(A)
L =
    1.0000         0         0         0
   -0.7500    1.0000         0         0
```

```
    0.5000   -0.4000    1.0000         0
    0.2500    0.6000   -0.4583    1.0000
U =
    4.0000    2.0000    2.0000    4.0000
         0    2.5000    4.5000    5.0000
         0         0    4.8000    3.0000
         0         0         0    1.3750
P =
     0     0     1     0
     0     0     0     1
     0     1     0     0
     1     0     0     0
>> % PA=LU ALLORA LUx = PAx = Pb.
>> b1=P*b;
>>
>> % PRIMO SISTEMA TRIANGOLARE.
>> y=L\b1;
>>
>> % SECONDO SISTEMA TRIANGOLARE.
>> x=U\y
x =
    1.6364
   -6.2727
    2.0909
    5.4545
>>
>> % CONFRONTO TRA A*x E b.
>> A*x
ans =
   13.0000
   28.0000
   20.0000
    6.0000
>> b
b =
    13
    28
    20
     6
>>
```

Se in alternativa utilizziamo le routines di sostituzione in avanti e all'indietro

```
>> A=[12 1 4; 2 0 4 3; 4 2 2 4; -3 1 3 2];
>> [L,U,P]=lu(A);
>> %PA=LU. Ax=b; ALLORA PAx=Pb. ALLORA LUx=Pb.
>> b1=P*b;
>> [y]=sostit_avanti(L,b1);
>> [x]=sostit_indietro(U,y)
x =
```

```
    1.6364
   -6.2727
    2.0909
    5.4545
>> A*x-b
ans =
  1.0e-014 *
    0.3553
         0
         0
         0
>>
```

2 Facoltativo: un'implementazione dell'eliminazione gaussiana con pivoting

Vediamo ora compe indipendentemente dal comando Matlab/Octave \ si modifichi `fattLU` per ottenere una routine che risolva col metodo LU (anche detto di *eliminazione gaussiana*) con pivoting il sistema $Ax = b$. Da un'occhiata veloce al codice di `fattLU` si capisce subito che un primo problema è l'assegnazione

```
A(k+1:n,k)=A(k+1:n,k)/A(k,k);
```

in cui il termine $A(k,k)$ al denominatore può essere nullo. Se la matrice iniziale A è invertibile, visto che le trasformazioni apportate non cambiano il valore assoluto del determinante (cosa non ovvia), al passo k-simo è sicuramente non nullo uno dei termini al di sotto della diagonale e nella k-sima colonna, cioè

$$A(k,k),\ A(k+1,k),\ \ldots,A(n,k).$$

Se così non fosse si potrebbe provare che la matrice non è invertibile, il che è assurdo in quanto abbiamo supposto $\det A \neq 0$. Visivamente saremmo in una condizione del tipo

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \ldots & a_{2,n} \\ 0 & 0 & 0 & \ldots & a_{3,n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & a_{m,n} \end{pmatrix}$$

ed essendo le prime tre righe linearmente dipendenti il determinante di A è necessariamente 0.

Sappiamo quindi che se anche $A(k,k) = 0$, esiste almeno un indice $j \in k+1,\ldots,n$ tale che $A(j,k) \neq 0$. Tra tutti gli indici j tali che $A(j,k) \neq 0$, si scelga per motivi di stabilità j^* per cui è massimo $|A(j,k)|$. A questo punto si inverta la riga k-sima di A con la j^*-sima e si proceda con l'algoritmo della fattorizzazione LU. Così facendo il processo della fattorizzazione LU può essere portato a termine ma si verifica che

invece di $A = LU$ si ha $PA = LU$ con P matrice di permutazione cioè del tipo $P^{-1} = P^T$ ove al solito P^T è la trasposta della matrice P.

Nota la fattorizzazione $PA = LU$, se $Ax = b$ allora $PAx = Pb$ e quindi $LUx = Pb$. Posto $\hat{b} = Pb$, abbiamo $LUx = \hat{b}$ che può essere risolto come abbiamo visto prima via sostituzioni all'indietro e all'avanti.

Consideriamo un codice che implementa in Matlab tale metodo LU (con pivoting). Una versione più sofisticata è usata da Matlab nelle sue funzioni di algebra lineare e la descrizione che segue ha solo interesse didattico.

```
function [x,L,U,P]=lupivot(A,b)

%-------------------------------------
% INPUT:
% A matrice.
% b termine noto.
%
% OUTPUT:
% x soluzione del sistema Ax=b.
% A matrice elaborata dalla funzione.
% r vettore degli scambi.
%-------------------------------------

[n,n]=size(A);
x=zeros(n,1);
y=zeros(n,1);
c=zeros(1,n);
r=1:n;

for p=1:n-1

    % Calcola il pivot relativo alla "p"-sima colonna.
    [max1,j]=max(abs(A(p:n,p)));

    % Scambia righe "p" e "j".
    c=A(p,:);
    A(p,:)=A(j+p-1,:);
    A(j+p-1,:)=c;
    d=r(p);
    r(p)=r(j+p-1);
    r(j+p-1)=d;

    if A(p,p) == 0
       fprintf('\n \t [WARNING]: A NON INVERTIBILE');
       break
    end

    for k=p+1:n
        mult=A(k,p)/A(p,p);
        A(k,p)=mult;
        A(k,p+1:n)=A(k,p+1:n)-mult*A(p,p+1:n);
```

```
    end

end

% Calcola "y".
y(1)=b(r(1));
for k=2:n
    y(k)=b(r(k))-A(k,1:k-1)*y(1:k-1);
end

% Calcola "x".
x(n)=y(n)/A(n,n);
for k=n-1:-1:1
    x(k)=(y(k)-A(k,k+1:n)*(k+1:n))/A(k,k);
end

% Calcoliamo P, L, U.
U=(tril(A'))'; L=A-U+eye(size(A));

P=zeros(size(A));
for k=1:n
    P(k,r(k))=1;
end
```

Vediamo l'esempio $Ax = b$ con A e b definiti rispettivamente da (), (). Dalla shell di Matlab digitiamo

```
>> A=[1 2 1 4; 2 0 4 3; 4 2 2 4; -3 1 3 2];
>> b=[13; 28; 20; 6];
>> [x,L,U,P]=lupivot(A,b)
```

Si ottiene

```
>> [x,L,U,P]=lupivot(A,b)

x =

    1.6364
   -6.2727
    2.0909
    5.4545


L =

    1.0000         0         0         0
   -0.7500    1.0000         0         0
    0.5000   -0.4000    1.0000         0
```

```
    0.2500    0.6000   -0.4583    1.0000


U =

    4.0000    2.0000    2.0000    4.0000
         0    2.5000    4.5000    5.0000
         0         0    4.8000    3.0000
         0         0         0    1.3750


P =

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

Se digitiamo `>>P*A-L*U` otteniamo la matrice nulla, e quindi $PA = LU$. Certamente P è una matrice di permutazione e lo conferma il fatto che se digitiamo `>>P*P'` otteniamo la matrice identica e quindi l'inversa di P è proprio la sua trasposta. Vediamo se in effetti $b - Ax$ è piccolo. Se fosse $x = x^*$ allora sarebbe $b - Ax^* = 0$. Scriviamo `>>b-A*x` e otteniamo

```
>> b-A*x

ans =
  1.0e-014 *

    0.1776
         0
         0
         0
```

La soluzione è esatta eccetto per la prima componente in cui viene fatto un errore di circa 10^{-15}. Come già detto, potevamo illustrare il problema della risoluzione dei sistemi lineari basandoci su un appropriato comando Matlab, ma non avremo capito la struttura della implementazione (la corrispondente funzione Matlab non è leggibile come file). Comunque,

```
>> A=[1 2 1 4; 2 0 4 3; 4 2 2 4; -3 1 3 2];
>> b=[13; 28; 20; 6];
>> x1=A\b

x1 =
    1.6364
   -6.2727
    2.0909
    5.4545
```

```
>> b-A*x1

ans =
  1.0e-014 *

   -0.3553
         0
   -0.3553
    0.1776

>>
```

3 Facoltativo. Qualche osservazione: complessità computazionale e calcolo del determinante

1. Una prima questione riguarda il numero di flops per risolvere un generico sistema lineare $Ax = b$ con A matrice $n \times n$. Si prova che questa quantità è $O(n^3/3)$ (cf. [18]).

2. Una seconda osservazione riguarda il calcolo del determinante di una matrice [20]. Si può vedere facilmente che essendo $PA = LU$ con P una matrice di permutazione, L triangolare inferiore con elementi diagonali uguali a 1 e U triangolare superiore e che

   - come già visto, la matrice di permutazione P ha la proprietà che $P^{-1} = P$ (cf. [17]) e quindi moltiplicando ambo a sinistra di $PA = LU$ per $P^{-1} = P$ ricaviamo $P^{-1}PA = P^{-1}LU = PLU$ da cui $A = PLU$;
   - per il teorema di Binet [19] si ha $\det(AB) = \det(A)\det(B)$, da cui

     $$\det(A) = \det(PLU) = \det(P)\det(LU) = \det(P)\det(L)\det(U);$$

   - il determinante di P è 1 o -1 a seconda degli scambi effettuati durante il pivoting (quantità facilmente calcolabile utilizzando una variabile di flag nell'implementazione della fattorizzazione LU con pivoting);
   - il determinante L è 1 (una matrice triangolare ha determinante uguale al prodotto dei termini diagonali che in questo caso sono tutti uguali a 1);
   - il determinante di $U = (u_{j,k})$ è $\prod_{k=1}^{n} u_{k,k}$ (una matrice triangolare ha determinante uguale al prodotto dei termini diagonali);

   necessariamente

   $$\det(A) = \det(P)\det(U) = \pm \prod_{k=1}^{n} u_{k,k}.$$

   La complessità computazionale in questo caso è così $O(n^3/3)+O(n) = O(n^3/3)$ cioè essenzialmente quella dovuta alla fattorizzazione LU, da paragonarsi con quella della regola di Laplace che è $O(n!)$ (cf. [18]).

4 Esercizio: fattorizzazione di Cholesky

Data una matrice di ordine n simmetrica definita positiva A (cioè $A_{i,j} = A_{j,i}$ e $x^T Ax > 0$ se $x \neq 0$) si dimostra che esiste una matrice R triangolare superiore, tale che $A = R^T R$ e $r_{i,i} = 1$ per $i = 1,\ldots,n$. Supposto $A = LU$, $D := diag(U)$ si dimostra che $R = D^{-1/2}U$ (cf. [2, p.524], [4, p.278], [7], [8, p.93], [14], [3]).

Vediamo i dettagli. Si prova (non facile) che una matrice simmetrica ha una fattorizzazione $A = LU$. Se D è la matrice diagonale estratta da U si ponga $\tilde{R} = D^{-1}U$. Si mostra che se A è simmetrica allora $\tilde{R} = L^T$ e quindi

$$A = LU = LDD^{-1}U = LD\tilde{R} = LDL^T = LD^{1/2}D^{1/2}L^T.$$

Si ponga

$$R = D^{1/2}L^T = D^{1/2}\tilde{R} = D^{-1/2}U.$$

Allora, ricordando che per una matrice diagonale $D = D^T$

$$R^T = (D^{1/2}L^T)^T = L(D^{1/2})^T = LD^{1/2}$$

da cui

$$A = R^T R.$$

In alcuni manuali, come [4, p.278], la matrice A viene fattorizzata come SS^T con però S triangolare inferiore. In questo caso

$$S = R^T = LD^{1/2}.$$

Figure 1: Andrè Louis Cholesky (1875-1918).

Si dimostra che A è definita positiva se e solo se i suoi autovalori sono positivi. Gli autovalori di A si possono calcolare mediante il comando `eig`. Ad esempio la matrice simmetrica

$$A = \begin{pmatrix} 5 & 11 \\ 11 & 25 \end{pmatrix}$$

è definita positiva in quanto

```
>> A=[5 11; 11 25]
A =
     5    11
    11    25
>> eig(A)
ans =
    0.1339
   29.8661
>>
```

1. Come verificare che la matrice è simmetrica? Ricordiamo che se $\|\cdot\|$ è una norma matriciale allora
$$\|A\| = 0 \Leftrightarrow A = 0.$$
Poichè la matrice è simmetrica se e solo se $A^T = A$ cioè $A^T - A = 0$ (qui 0 è la matrice nulla). Di conseguenza $A^T = A$ se e solo se $\|A^T - A\| = 0$. Ricordiamo che per ottenere la cosidetta norma 2 di una matrice A basta effettuare la chiamata `norm(A);`

2. Esistono vari metodi alternativi al ciclo for per calcolare se tutti gli elementi di un vettore sono positivi. Vediamo alcuni esempi. Usiamo il comando `all`

```
>> help all

 ALL    True if all elements of a vector are nonzero.
    For vectors, ALL(V) returns 1 if none of the elements of the
    vector are zero. Otherwise it returns 0. For matrices,
    ALL(X) operates on the columns of X, returning a row vector
    of 1's and 0's. For N-D arrays, ALL(X) operates on the first
    non-singleton dimension.

    ALL(X,DIM) works down the dimension DIM.  For example, ALL(X,1)
    works down the first dimension (the rows) of X.

    See also ANY.

>> A=[5 11; 11 25]
A =
     5    11
    11    25
>> v=eig(A)
v =
    0.1339
   29.8661
>> u=(v <= 0)
u =
     0
```

```
     0
>> all(u)
ans =
     0
>> % LA RISPOSTA all(u) = 0 SIGNIFICA CHE u NON HA
>> % ELEMENTI MAGGIORI DI 0, E QUINDI NON E' MAI
>> % VERO CHE CI SIA UN COMPONENTE DI v CHE SIA MINORE
>> % DI 0. RICORDIAMO CHE 0 STA PER FALSE, 1 PER TRUE.
```

Vediamo un'alternativa.

```
>> A=[5 11; 11 25]
A =
     5    11
    11    25
>> v=eig(A)
v =
    0.1339
   29.8661
>> u=(v > 0)
ans =
     1
     1
>> % SE LA LUNGHEZZA DEL VETTORE u CORRISPONDE AL
>> % NUMERO DI OCCORRENZE DI COMPONENTI
>> % DI u CHE VALGONO 1, ALLORA TUTTE LE
>> % COMPONENTI DI u VALGONO 1 E QUINDI
>> % OGNI COMPONENTE DI v E' POSITIVA, DA CUI
>> % sum(u)-length(u)=0.
>> sum(u)-length(u)
ans =
     0
>> % IN ALTERNATIVA GUARDO CHE NESSUNA SIA MINORE
>> % O UGUALE A 0.
>> u1=(v <= 0)
u1 =
     0
     0
>> % SICCOME I VALORI DI u1 SONO 0 0 1, ALLORA SE
>> % LA SOMMA DI COMPONENTI DI u1 VALE 0,
>> % VUOL DIRE CHE IL TEST "(v <= 0)" NON E' MAI
>> % VERIFICATO PER ALCUNA COMPONENTE DI v.
>> sum(u1)
ans =
     0
>>
```

Come già detto la matrice diagonale estratta da A si ottiene come `diag(diag(A))` e non `diag(A)`. Se A è una matrice, per ottenere A^k basta eseguire il comando Matlab `A^k`.

Eseguire una funzione Matlab che verifichi se una matrice generica A sia simmetrica e definita positiva ed in tal caso esegua la fattorizzazione di Cholesky. Verificare che la quantità

```
norm(A-R'*R,inf)
```

sia piccola. Ricordiamo che come si vede dall'help

```
>> help norm

 NORM   Matrix or vector norm.
    For matrices...
      NORM(X) is the largest singular value of X, max(svd(X)).
      NORM(X,2) is the same as NORM(X).
      NORM(X,1) is the 1-norm of X, the largest column sum,
                      = max(sum(abs(X))).
      NORM(X,inf) is the infinity norm of X, the largest row sum,
                      = max(sum(abs(X'))).
      NORM(X,'fro') is the Frobenius norm, sqrt(sum(diag(X'*X))).
      NORM(X,P) is available for matrix X only if P is 1, 2, inf or 'fro'.

    For vectors...
      NORM(V,P) = sum(abs(V).^P)^(1/P).
      NORM(V) = norm(V,2).
      NORM(V,inf) = max(abs(V)).
      NORM(V,-inf) = min(abs(V)).

    See also COND, RCOND, CONDEST, NORMEST.

 Overloaded methods
    help ss/norm.m
    help lti/norm.m
    help frd/norm.m
    help idmodel/norm.m
```

Se la norma di una matrice A di ordine n è vicina a 0 allora $A \approx 0_n$, dove 0_n indica la matrice nulla di ordine n.
Si testi la bontà del programma relativamente alla matrice

$$A = \begin{pmatrix} 5 & 11 \\ 11 & 25 \end{pmatrix}.$$

Suggerimento: è bene usare `lu` di Matlab/Octave o si deve preferire `fattLU`? Perchè ?

Per aiutare la risoluzione facciamo un esempio:

```
>> A=hilb(2)
A =
    1.0000    0.5000
    0.5000    0.3333
```

```
>> [L,U]=lu(A)
L =
    1.0000         0
    0.5000    1.0000
U =
    1.0000    0.5000
         0    0.0833
>> D=diag(U)
D =
    1.0000
    0.0833
>> D=diag(diag(U))
D =
    1.0000         0
         0    0.0833
>> sqrtD=D^(1/2)
sqrtD =
    1.0000         0
         0    0.2887
>> R=inv(sqrtD)*U
R =
    1.0000    0.5000
         0    0.2887
>> R'*R
ans =
    1.0000    0.5000
    0.5000    0.3333
>> S=R'
S =
    1.0000         0
    0.5000    0.2887
>> S*S'
ans =
    1.0000    0.5000
    0.5000    0.3333
>>
```

5 Esercizio facoltativo: sulla matrice di Hilbert

Data una matrice A il suo numero di condizionamento (in norma 2) è

$$\mu = \|A\|_2 \|A^{-1}\|_2.$$

Sapendo che la norma 2 di una matrice A si calcola con il comando `norm(A,2)`, calcolare il numero di condizionamento della matrice di Hilbert $A = (a_{i,j}) = \frac{1}{i+j-1}$ con $i, j = 1, ldots, 5$. Confrontare il risultato con `cond(hilb(5))` che vale approssimativamente $4.7661e+005$. Riprovare lo stesso esperimento per $n = 25$.

6 Facoltativo. Cenni storici

Il metodo di eliminazione gaussiana probabilmente era noto prima di Gauss. Nel molto interessante [9], si riferisce fosse già noto ai Babilonesi nel 300 a.c. e ai cinesi nel 200 a.c. Per capire come i Babilonesi risolvessero questi problemi si consulti [1, p.16]. In [13], si asserisce sia parte del *Chapter Eight, Rectangular Arrays*, del testo cinese *Jiuzhang suanshu* anche noto come *The Nine Chapters on the Mathematical Art* [12], in cui si studiano 18 problemi con 2 o 5 equazioni. La prima referenza a tale libro è datata 179ac.

Figure 2: Carl Friedrich Gauss (1777-1855).

Successivamente è presente in modo primitivo in certi studi di Cardano, Leibniz, McLaurin, Bezout e Laplace. In particolare si afferma:

Il termine determinante fu introdotto per primo da Gauss nel Disquisitiones arithmeticae (1801) dedicato allo studio di forme quadratiche. Egli usò tale termine perchè il determinante determina le proprietà della forma quadratica. Comunque il concetto è diverso da quello nostro. Nello stesso lavoro Gauss dispone i coefficienti delle sue forme quadratiche in schieramenti rettangolari. Egli descrive la moltiplicazione di matrice (che pensa come composizione non raggiungendo quindi il concetto di algebra matriciale) e l'inversa di una matrice nel particolare contesto degli schieramenti di coefficienti delle forme quadratiche. L'eliminazione Gaussiana, che apparve dapprima nel testo cinese Nove Capitoli dell'Arte Matematica scritto nel 200 A.C., fu usata da Gauss nel suo lavoro dedicato allo studio dell'orbita dellŠasteroide Pallas. Usando le osservazioni di Pallas fra il 1803 ed il 1809, Gauss ottenne un sistema di sei equazioni lineari in sei incognite. Gauss fornì un metodo sistematico per risolvere tali equazioni che è precisamente lŠeliminazione Gaussiana sulla matrice dei coefficienti.

7 Online

Per ulteriori dettagli sulla fattorizzazione LU si consultino i siti

1. http://it.wikipedia.org/wiki/Fattorizzazione_LU
2. http://en.wikipedia.org/wiki/LU_decomposition

3. http://en.wikipedia.org/wiki/Gaussian_elimination
4. http://en.wikipedia.org/wiki/The_Nine_Chapters_on_the_Mathematical_Art
5. http://math.fullerton.edu/mathews/n2003/BackSubstitutionMod.html
6. http://www.math.tamu.edu/d̃allen/masters/egypt_babylon/babylon.pdf
7. http://it.wikipedia.org/wiki/Gauss
8. http://it.wikipedia.org/wiki/Decomposizione_di_Cholesky
9. http://it.wikipedia.org/wiki/Andr%C3%A9-Louis_Cholesky
10. http://it.wikipedia.org/wiki/Matrice_permutativa
11. http://it.wikipedia.org/wiki/Teorema_di_Binet
12. http://it.wikipedia.org/wiki/Determinante

References

[1] G.D. Allen, http://www.math.tamu.edu/d̃allen/masters/egypt_babylon/babylon.pdf.

[2] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, 1989.

[3] C. Brezinski, http://www.rehseis.cnrs.fr/calculsavant/Textes/biographied'andr.html.

[4] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.

[5] J.H. Mathews e K.D. Fink, *Numerical Methods using Matlab*, Prentice Hall, 1999.

[6] The MathWorks Inc., *Numerical Computing with Matlab*, http://www.mathworks.com/moler.

[7] A. Quarteroni e F. Saleri, *Introduzione al calcolo scientifico*, Springer Verlag, 2006.

[8] G. Rodriguez, *Algoritmi Numerici*, Pitagora Editrice, (2008).

[9] WEB, http://www.na.iac.cnr.it/even/matrici.htm.

[10] Wikipedia (Fattorizzazione LU), http://it.wikipedia.org/wiki/Fattorizzazione_LU.

[11] Wikipedia (LU Decomposition), http://en.wikipedia.org/wiki/LU_decomposition.

[12] Wikipedia (The Nine Chapters), http://en.wikipedia.org/wiki/The_Nine_Chapters_on the_Mathematical_Art.

[13] Wikipedia (Gaussian elimination),
http://en.wikipedia.org/wiki/Gaussian_elimination.

[14] Wikipedia (Decomposizione di Cholesky),
http://it.wikipedia.org/wiki/Decomposizione_di_Cholesky.

[15] Wikipedia (Cholesky), http://it.wikipedia.org/wiki/Andr%C3%A9-Louis_Cholesky.

[16] Wikipedia (Gauss), http://it.wikipedia.org/wiki/Gauss.

[17] Wikipedia (Matrice permutativa),
http://it.wikipedia.org/wiki/Matrice_permutativa.

[18] Wikipedia (Algoritmo),
http://it.wikipedia.org/wiki/Algoritmo.

[19] Wikipedia (Teorema di Binet),
http://it.wikipedia.org/wiki/Teorema_di_Binet.

[20] Wikipedia (Determinante),
http://it.wikipedia.org/wiki/Determinante.