Algebra lineare

30 giugno 2007

1 Soluzione di un sistema di equazioni lineari

Si consideri il sistema lineare di m equazioni in n incognite

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

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 () si può riscrivere come

$$A \cdot x = b$$

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

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

   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 $Ax = b$.

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
>>
```

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 questi due metodi. Supponiamo $A = LU$ e di voler risolvere il sistema $Ax = b$.

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

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

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] = fattLUkji(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 i=k+1:n
      A(i,j)=A(i,j)-A(i,k)*A(k,j) ;
    end,
  end
end
```

Una variante più complessa e stabile è quella vista utilizzando 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).

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 $A = LU$. Da un'occhiata veloce al codice di `fattLUkji` 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 $\det A \neq 0$. Sappiamo quindi che se anche $A(k,k) = 0$, esiste almeno un indice $j \in k+1,\dots,n$ 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.

Vediamone un esempio che ben illustra il metodo LU con pivoting. Si consideri il sistema lineare (tratto da [2], p.127)

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

Riscritto matricialmente per

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

e

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

si cerca l'unico x^* tale che $Ax^* = b$.

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)\cdotx(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 i=1:n
    P(i,r(i))=1;
end
```

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

>>
```

2 Online

Per ulteriori dettagli sulla fattorizzazione LU si consultino i siti

http://it.wikipedia.org/wiki/Fattorizzazione_LU
http://en.wikipedia.org/wiki/LU_decomposition
http://math.fullerton.edu/mathews/n2003/BackSubstitutionMod.html

References

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

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

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

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