Laboratorio computazionale numerico
Lezione 2

f.poloni&sns.it

2008-11-05

1 Fattorizzazione LU ed eliminazione di Gauss

1.1 Matrice di test

Esercizio 1 (di riscaldamento). Scrivere una funzione testmatrix(n) che, dato n, costruisca la matrice $n \times n$ con n sulla diagonale e uni altrove:

```
octave:5> testmatrix(5)
ans =

   5   1   1   1   1
   1   5   1   1   1
   1   1   5   1   1
   1   1   1   5   1
   1   1   1   1   5
```

Inizialmente potreste pensare di risolverlo così:

```
function M=testmatrix(n)
M=ones(n,n);
for i=1:n
  M(i,i)=n;
endfor
```

Un'idea migliore però è questa:

```
octave:7> M=ones(n,n)+(n-1)*eye(n)
M =

   5   1   1   1   1
   1   5   1   1   1
   1   1   5   1   1
   1   1   1   5   1
   1   1   1   1   5
```

Octave è molto efficiente su calcoli su matrici e vettori "come un tutt'uno", si cerca sempre di rimpiazzare un ciclo **for** con una singola istruzione che lavora su vettori.

1.2 Soluzione di sistemi triangolari

Le seguenti funzioni risolvono un sistema triangolare inferiore/superiore. Sono diverse da quelle della lezione scorsa.

```
function x=inf_solve(L,b)
%risolve un sistema con L triangolare inferiore
  n=size(L,1);
  x=b; %x "vettore di accumulatori"
  for i=1:n
    x(i)=x(i)/L(i,i);
    x(i+1:n)=x(i+1:n) - L(i+1:n,i)*x(i);
  endfor
endfunction
```

```
function x=sup_solve(U,b)
%risolve un sistema con U triangolare superiore
  n=size(U,1);
  x=b; %x "vettore di accumulatori"
  for i=n:-1:1
    x(i)=x(i)/U(i,i);
    x(1:i-1)=x(1:i-1) - U(1:i-1,i)*x(i);
  endfor
endfunction
```

Esercizio 2. Capire come (e perché) funziona questa versione di infsolve /supsolve. Ricordarsi che la formula è

$$x_i = \frac{b_i - \sum_{j=0}^{i-1} L_{ij} x_j}{L_{ii}}, \quad x = 1 \dots n.$$

1.3 Fattorizzazione LU

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ * & 1 & 0 & 0 & 0 \\ * & 0 & 1 & 0 & 0 \\ * & 0 & 0 & 1 & 0 \\ * & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} * & * & * & * & * \\ 0 & U_{22} & * & * & * \\ 0 & U_{32} & * & * & * \\ 0 & U_{42} & * & * & * \\ 0 & U_{52} & * & * & * \end{pmatrix}$$

Step 2: al posto degli elementi rossi di L, sostituiamo $L_{i2} = \frac{U_{i2}}{U_{22}}$. Al posto degli elementi rossi di U, sostituiamo $U_{ij} = U_{ij} - L_{i2}U_{2j}$. Nota che in questo modo gli elementi rossi sulla seconda colonna (U_{i2}) diventano 0.

La seguente funzione calcola la fattorizzazione LU di una matrice quadrata (non possiamo chiamarla **lu** perché c'è già una funzione di Octave che si chiama così).

```
function [L,U]=lu_labo(A)
  n=size(A,1);
  L=eye(n);
  U=A;
  for k=1:n
    pivot=U(k,k);
    L(k+1:n,k)=U(k+1:n,k)/pivot;
    colonnaL=L(k+1:n,k);
    rigaU=U(k,k+1:n);
    U(k+1:n,k+1:n)=U(k+1:n,k+1:n)-colonnaL*rigaU;
    U(k+1:n,k)=0;
  endfor
endfunction
```

```
octave:47> M=testmatrix(5)
M =

   5   1   1   1   1
   1   5   1   1   1
   1   1   5   1   1
   1   1   1   5   1
   1   1   1   1   5

octave:48> [L,U]=lu_labo(M)
octave:49> L*U-M
ans =

 Columns 1 through 3:

    0.00000000000000e+00    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00   -8.88178419700125e-16
    0.00000000000000e+00    0.00000000000000e+00    2.22044604925031e-16
    0.00000000000000e+00    0.00000000000000e+00    2.22044604925031e-16

 Columns 4 and 5:

    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    0.00000000000000e+00
    0.00000000000000e+00    2.22044604925031e-16
    2.22044604925031e-16    0.00000000000000e+00
```

Possiamo usare le funzioni scritte finora per risolvere un generico sistema lineare:

```
function x=sys_solve(A,b)
  [L,U]=lu_labo(A);
  y=inf_solve(L,b);
  x=sup_solve(U,y);
endfunction
```

Se usiamo la matrice testmatrix(5), i risultati sono buoni (infatti la matrice è dominante diagonale):

```
octave:60> M=testmatrix(5)
M =

   5   1   1   1   1
   1   5   1   1   1
   1   1   5   1   1
   1   1   1   5   1
   1   1   1   1   5

octave:61> y=[1:5]'
y =

   1
   2
   3
```

```
  4
  5

octave:62> sys_solve(M,M*b)
ans =

  1.00000000000000
  2.00000000000000
  3.00000000000000
  4.00000000000000
  5.00000000000000
```

Se usiamo matrici peggiori, cominciamo ad ottenere errori significativi. Per esempio, il comando **rand**(n,n) restituisce una matrice $n \times n$ contenente numeri casuali tra 0 e 1.

```
octave:81> M=rand(10,10)
octave:82> b=M*[1:10]';
octave:83> sys_solve(M,b)
ans =

   0.999999999999899
   2.000000000000022
   3.000000000000054
   4.000000000000063
   4.999999999999970
   6.000000000000018
   6.999999999999960
   7.999999999999973
   8.999999999999996
  10.000000000000046
```

Aumentando la dimensione del sistema, gli errori aumentano (verificare!).

1.4 Matrici che non ammettono fattorizzazione LU

Cosa succede numericamente se la matrice non ammette una fattorizzazione LU? Per fare in modo che la sottomatrice 4×4 di testa sia singolare, possiamo inserire una riga di zeri:

$$\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ 0 & 0 & 0 & 0 & * \\ * & * & * & * & * \end{pmatrix}$$

```
octave:85> M=testmatrix(5);
octave:86> M(4,1:4)=0
M =

  5  1  1  1  1
  1  5  1  1  1
  1  1  5  1  1
  0  0  0  0  1
  1  1  1  1  5
```

```
octave:87> b=M*[1:5]';
octave:88> sys_solve(M,b)
warning: in /.automount/homeserver/root/RAID/5/home/p/poloni/prova/lu_labo.m near lin
warning: division by zero
warning: division by zero
ans =

   NaN
   NaN
   NaN
   NaN
   NaN
```

La matrice M non è singolare (testare calcolando **det**(M)), tuttavia non ammette fattorizzazione LU, per cui il nostro algoritmo fallisce. Dove? Verificare facendo scrivere a schermo un po' di matrici parziali.

1.5 Matrici quasi singolari

Nella pratica, il caso precedente non si verifica quasi mai. Se i calcoli sono più complessi di quelli del nostro esempio, gli errori di arrotondamento trasformano gli zeri in numeri dell'ordine di 10^{-16}.

Il codice seguente genera una matrice casuale 10×10, e poi modifica la sottomatrice principale di testa 9×9 in modo che la sua ultima riga sia la somma delle otto precedenti.

```
octave:5> M=rand(10,10);
octave:6> M(9,1:9)=sum(M(1:8,1:9));
```

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

(gli elementi in rosso sono ognuno la somma degli otto elementi che stanno direttamente sopra di esso).

Gli errori generati da sys_solve sono notevoli:

```
octave:9> b=M*[1:10]';
octave:10> sys_solve(M,b)
ans =

   0.91107
   1.99647
   3.20539
   4.04115
   5.13499
   5.93008
   7.07241
   7.82438
   8.94444
  10.00000
```

Il motivo? Guardiamo il fattore U della fattorizzazione LU di M:

```
octave:14> [L,U]=lu_labo(M); U
```

L'elemento $(9, 9)$, che in aritmetica esatta varrebbe zero, vale $\approx 10^{-15}$. Di conseguenza, anche l'elemento $(10, 10)$ è sbilanciato ($\approx 10^{14}$), visto che il prodotto dei pivot dev'essere costante (perché?).

La funzione **lu** di Octave, che calcola una fattorizzazione LU *con pivoting parziale* fornisce invece risultati molto migliori.

```
octave:25> [L,U]=lu(M)
octave:26> inv(U)*inv(L)*b
ans =

   0.999999999999986
   2.000000000000005
   3.000000000000003
   3.999999999999987
   5.000000000000014
   5.999999999999999
   6.999999999999997
   8.000000000000004
   8.999999999999906
  10.000000000000000
```

(Warning: questo è solo per confronto, fare esplicitamente la moltiplicazione **inv**(A)∗b non è il modo migliore per risolvere un sistema lineare!)

Esercizio 3. Scrivere una funzione det_gauss(A) che calcoli il determinante di A attraverso la fattorizzazione LU (eventualmente con pivoting).

Esercizio 4. Provare a scrivere una funzione gepp(A,b) che risolva il sistema $Ax = b$ utilizzando l'eliminazione di Gauss con pivoting parziale. Per fare questo, vi servirà la funzione [val,ind]=**max**(v) che ritorna un indice ind tale che val=v(ind) è l'elemento più grande del vettore v. Occhio agli indici però: vogliamo applicarla in una forma del tipo [pivot,pivotpos]=max(abs(A(i:n,i))); se per esempio il massimo del vettore sta sul primo elemento, **max** ritorna 1, ma in realtà è la riga i-esima della matrice...

Se vi stavate annoiando.... Scrivere una funzione [P,L,U]=plu(A) che calcoli la fattorizzazione PLU di una matrice quadrata. Come si può rappresentare P in modo più compatto?

1.6 Qualche esperimento sul pivoting (con funzioni già fatte)

Copiate e incollate (non riscrivetele a mano!) le seguenti funzioni, che risolvono un sistema con l'eliminazione di Gauss con pivoting parziale (gepp, *Gaussian elimination with partial pivoting*) e totale (gecp, *Gaussian elimination with complete pivoting*).

```
function x=gepp(A,b)
  n=size(A,1);
  x=b;
  for i=1:n
    %trova il pivot
```

```
    [abspivot,pivotpos]=max(abs(A(i:n,i)));
    pivotpos=pivotpos+i-1;
    %scambia le righe
    A([i,pivotpos],:)=A([pivotpos,i],:);
    x([i,pivotpos])=x([pivotpos,i]);
    %passo di eliminazione di Gauss
    pivot=A(i,i);
    A(i+1:n,i)=A(i+1:n,i)/pivot;
    A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
    x(i+1:n)=x(i+1:n)-A(i+1:n,i)*x(i);
  endfor
  %risolve U^{-1}x
  for i=n:-1:1
    x(i)=x(i)/A(i,i);
    x(1:i-1)=x(1:i-1)-A(1:i-1,i)*x(i);
  endfor
endfunction
```

```
function x=gecp(A,b)
  n=size(A,1);
  x=b;
  p=1:n; %per tenere traccia della permutazione sulle colonne
  for i=1:n
    %trova il pivot (fa un po' di casino)
    maxcols=max(abs(A(i:n,i:n))); %vettore riga dei max sulle colonne
    [abspivot,jpivot]=max(maxcols);
    jpivot=jpivot+i-1;
    [abspivot,ipivot]=max(abs(A(i:n,jpivot)));
    ipivot=ipivot+i-1;
    %scambia le righe
    A([i,ipivot],:)=A([ipivot,i],:);
    x([i,ipivot])=x([ipivot,i]);
    %scambia le colonne
    A(:,[i,jpivot])=A(:,[jpivot,i]);
    %tiene traccia della permutazione delle colonne
    p([i,jpivot])=p([jpivot,i]);
    %passo di eliminazione di Gauss
    pivot=A(i,i);
    A(i+1:n,i)=A(i+1:n,i)/pivot;
    A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
    x(i+1:n)=x(i+1:n)-A(i+1:n,i)*x(i);
  endfor
  %risolve U^{-1}x
  for i=n:-1:1
    x(i)=x(i)/A(i,i);
    x(1:i-1)=x(1:i-1)-A(1:i-1,i)*x(i);
  endfor
  %applica la permutazione sulle colonne
  x(p)=x;
endfunction
```

Esercizio 5. Prendete una matrice quasi singolare. Come cambia l'errore sulla soluzione con le varie strategie di pivoting (parziale, totale, nessuno)? E se la matrice è presa con elementi casuali?