

---

# **Metodi Numerici per l'ingegneria LS**

## **a.a. 2008-2009**

Fabiana Zama

<http://www.dm.unibo.it/~zama>

[zama@dm.unibo.it](mailto:zama@dm.unibo.it)

# Numero di Condizione

---

Dati  $\mathbf{Ax} = \mathbf{b}$  e  $(\mathbf{A} + \Delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$  si vuole stimare:

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \dots$$

Si può dimostrare che se  $\Delta\mathbf{A}$  è tale che  $\|\Delta\mathbf{A}\|\|\mathbf{A}^{-1}\| = r < 1$  allora il sistema perturbato  $(\mathbf{A} + \Delta\mathbf{A})\mathbf{y} = \mathbf{b} + \Delta\mathbf{b}$  è non singolare. Inoltre, se per un  $\delta \geq 0$  si ha:  $\|\Delta\mathbf{A}\| \leq \delta\|\mathbf{A}\|$ ,  $\|\Delta\mathbf{b}\| \leq \delta\|\mathbf{b}\|$  allora

$$\frac{\|x - y\|}{\|x\|} \leq 2\delta\|\mathbf{A}^{-1}\|\|\mathbf{A}\| \frac{1}{(1 - r)}, \quad x \neq 0$$

**NUMERO di CONDIZIONE**  $K(\mathbf{A}) = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|$

# Calcolo del Numero di condizione

- La function `cond` calcola il numero di condizione secondo la definizione data.
  - `cond(A)` usa  $\|\cdot\|_2$ : Siano  $\lambda_i, u_i$  autovalori ed autovettori di  $A$ :  $Au_i = \lambda_i u_i$  tali che  $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$  allora  $\|A\|_2 = \lambda_1$  e  
 $\|A^{-1}\|_2 = \max_{1 \leq i \leq n} \sqrt{\rho(\mathbf{A}^{-T} \mathbf{A}^{-1})} = 1 / \min_{1 \leq i \leq n} \sqrt{\rho(\mathbf{A}^{-T} \mathbf{A}^{-1})} = 1 / \lambda_n$   
quindi  $\mu_2(\mathbf{A}) = |\lambda_1| / |\lambda_n|$
  - `cond(A,p)`  $p=1, 'inf', 'fro'$ , usa rispettivamente  $\|\cdot\|_1, \|\cdot\|_\infty, \|\cdot\|_F$   
 $\text{cond}(A,p) = \text{norm}(A,p) * \text{norm}(\text{inv}(A),p)$   
In questo caso viene calcolata la matrice inversa  $A^{-1}$  mediante la funzione `inv`.

# Calcolo del Numero di condizione

---

- La function `condest` calcola una stima del numero di condizione in norma  $\| \cdot \|_1$ .
- `rcond` calcola una stima di  $1/K(A)$ 
  - Se  $A$  è ben condizionata,  $\text{rcond}(A) \approx 1.0$ .
  - Se  $A$  è mal condizionata,  $\text{rcond}(A) \approx 0.0$ .

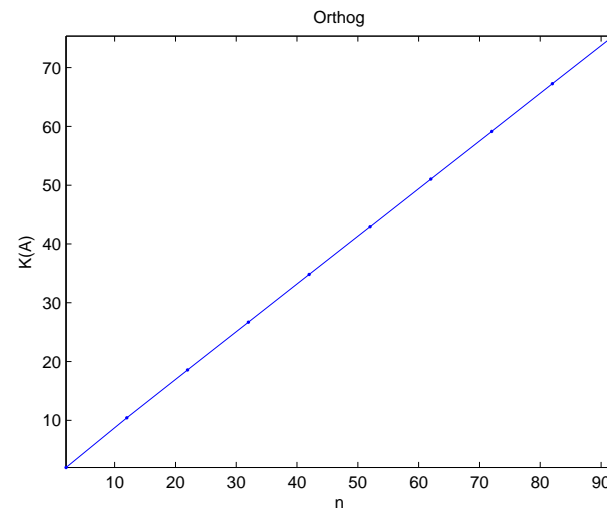
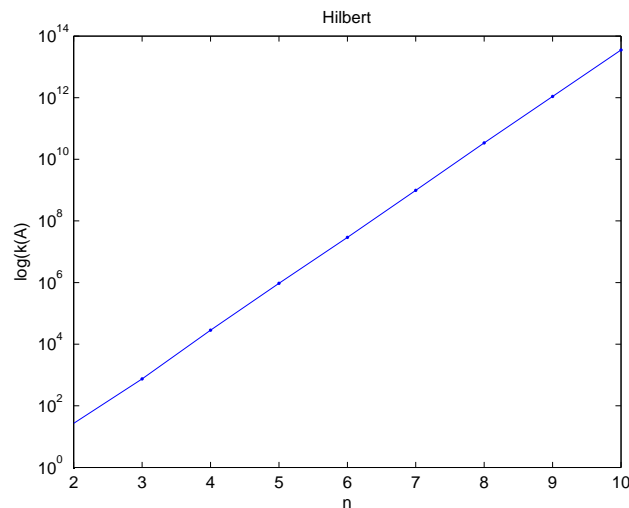
Rispetto a `cond` sono meno precise ma più efficienti in quanto evitano il calcolo della matrice inversa.

```
>>fprintf('Numero di Condizione  cond=%e...  
          condest=%e\n',cond(V),condest(V));
```

```
Numero di Condizione  cond=1.242871e+029  
          condest=2.219221e+029
```

# Studio di $K(A)$ per $n$ variabile

- Riportare in un grafico il numero di condizione  $K(A)$  per valori di  $n = 2, 3, \dots$  e trarre conclusioni sul condizionamento di  $A$ .



```
vec=2:10:100;  
for N=vec  
    A=gallery('orthog',N);  
    ke(N)=condest(A);  
end  
figure;plot(vec,ke(vec),'.-');  
title('Orthog');xlabel('n');ylabel('K(A)');  
axis tight;
```

Metodi per risolvere numericamente il sistema lineare:

$$\mathbf{Ax} = \mathbf{b}, \quad \mathbf{A} \in \mathbb{R}^{n \times n}$$

- Metodi Diretti: la soluzione viene calcolata in un numero finito di passi modificando la matrice del problema in modo da rendere più agevole il calcolo della soluzione.
  - Matrici triangolari: Metodi di Sostituzione;
  - Matrici tridiagonali;
  - Metodo di Eliminazione di Gauss;
  - Matrici simmetriche: Metodo di Cholesky;
- Metodi Iterativi: Calcolo di una soluzione come limite di una successione di approssimazioni  $\mathbf{x}_k$ , senza modificare la struttura della matrice  $\mathbf{A}$ . Adatti per sistemi di grandi dimensioni con matrici **sparse** (pochi elementi non nulli).

## Soluzione $Ax = b$

---

- Calcolo  $P, L, R$  tale che  $PA = LR$ .
- $PA = Pb$  quindi  $z = Pb$ .
- Sistema Triangolare Inferiore:

$$Ly = z$$

- Sistema triangolare superiore:

$$Rx = y$$

La soluzione del sistema lineare con algoritmo di fattorizzazione e pivoting parziale viene realizzato in matlab mediante la function `mldivide`  
`mldivide(A,b)  $\Leftrightarrow A \setminus b$`

# Operatore \

---

Esempio: soluzione di un sistema lineare di ordine  $n = 10$  mediante \:

```
n=10;
```

```
A=rand(n);b=rand(n,1);
```

```
x=A\b;
```

Il medesimo risultato può essere calcolato utilizzando la function `lu`

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

```
z=P*b;
```

```
y=LTriSol(L,z);
```

```
xx=UtriSol(U,y);
```



# Function per sistema triangolare superiore

---

```
function x = UTriSol(U, b)
% Solves the nonsingular upper triangular system  $Ux = b$ .
% where U is n-by-n, b is n-by-1, and X is n-by-1.
n = length(b); x = zeros(n, 1);
for j = n : -1 : 2
    x(j) = b(j)/U(j, j);
    b(1 : j - 1) = b(1 : j - 1) - x(j) * U(1 : j - 1, j);
end
x(1) = b(1)/U(1, 1);
```

## Function per sistema triangolare inferiore

---

```
function x = LTriSol(L, b)
% Solves the nonsingular lower triangular system
%  $Lx = b$  where  $L$  is  $n$ -by- $n$ ,  $b$  is
%  $n$ -by-1, and  $x$  is  $n$ -by-1.
n = length(b); x = zeros(n, 1);
for j = 1 : n - 1
    x(j) = b(j)/L(j, j);
    b(j + 1 : n) = b(j + 1 : n) - L(j + 1 : n, j) * x(j);
end
x(n) = b(n)/L(n, n);
```

# Effetti del condizionamento

- Si risolvono i seguenti sistemi lineari  $A_i x_i = b_i$  con  $A_i$  matrici di Hilbert di ordine  $n = 2 : 2 : 20$  e  $x_i = \sin(\text{linspace}(0, \pi, n))'$  e si riportano in una tabella i valori  $\|b - A\tilde{x}_i\|_\infty$  e  $\|x_i - \tilde{x}_i\|_\infty$
- Utilizzare sia `\` che `lu + LTrisol + UTriSol`.
- Risultati `lu + LTrisol + UTriSol`.

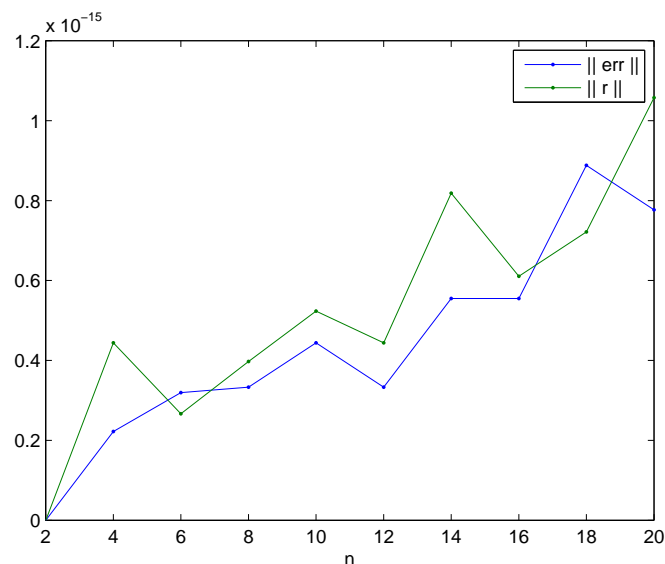
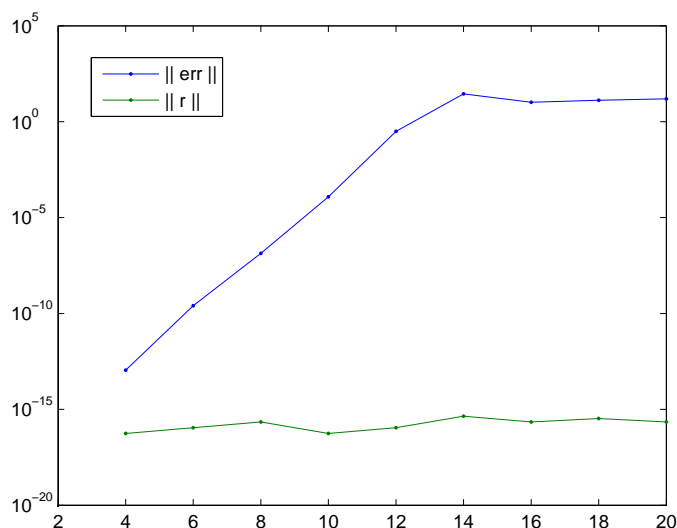
2	$\ r\ $	$= 0.000000e+000$	$\ err\ $	$= 0.000000e+000$
4	$\ r\ $	$= 5.551115e-017$	$\ err\ $	$= 1.119105e-013$
6	$\ r\ $	$= 1.110223e-016$	$\ err\ $	$= 2.528808e-010$
8	$\ r\ $	$= 2.220446e-016$	$\ err\ $	$= 1.377238e-007$
10	$\ r\ $	$= 5.551115e-017$	$\ err\ $	$= 1.196802e-004$
12	$\ r\ $	$= 1.110223e-016$	$\ err\ $	$= 3.132954e-001$
14	$\ r\ $	$= 4.440892e-016$	$\ err\ $	$= 2.848335e+001$
16	$\ r\ $	$= 2.220446e-016$	$\ err\ $	$= 1.032531e+001$
18	$\ r\ $	$= 3.330669e-016$	$\ err\ $	$= 1.319117e+001$
20	$\ r\ $	$= 2.220446e-016$	$\ err\ $	$= 1.551251e+001$

- Con valori di  $n > 10$  si hanno grandi errori nelle soluzioni calcolate a parità di piccoli valori nel residuo.

## Script

```
vect=2:2:20 %vect=10:25:500
i=0;
for n=vect
    A=gallery('orthog',n);%A=hilb(n);%
    x=sin(linspace(0,pi,n))';
    y=A*x; %xx=A\y;
    [L,U,P] = lu(A);z=P*y;
    b=LTriSol(L,z);xx=UtriSol(U,b);
    r=norm(y-A*xx,'inf');
    err=norm(x-xx,'inf');
    i=i+1;R(i)=r;E(i)=err;
    fprintf('%d ||r|| = %e...
           || err || = %e \n',n,r,err);
end
%semilogy(vect,[E',R'],'.-');
plot(vect,[E',R'],'.-');
legend('|| err ||','|| r ||');xlabel('n')
```

● Confronto con la matrice test  $A = \text{gallery}('orthog', n);$



● Quando la matrice è ben condizionata residui ed errori hanno lo stesso ordine di grandezza.

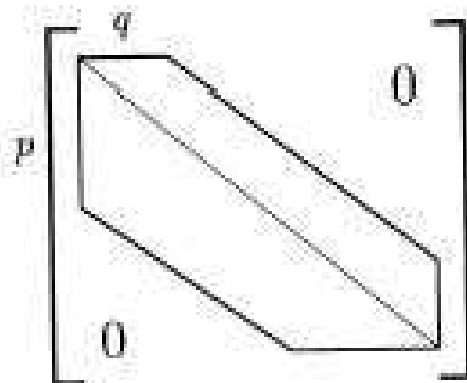
# Matrici a banda

- Si dice che una matrice  $A$  ha una banda inferiore  $q$  se

$$a_{i,j} = 0 \quad \text{per } j > i + q$$

- Si dice che  $A$  ha una banda superiore  $p$  se

$$a_{i,j} = 0 \quad \text{per } i > j + p$$



## Esempio

Si definisce la seguente matrice a banda di ordine 5 con  $p = 2$  e  $q = 1$ .

```
A = [ 2      1      0      0      0
      4      4      1      0      0
      6      5      3      1      0
      0      6      5      3      1
      0      0      6      5      3 ] ;
```

Il comando `sparse` consente di memorizzare solo gli elementi non nulli della matrice.

```
>> Asp=sparse(A) ;
```

Osservare la nuova struttura di A e l'occupazione di memoria.

```
>> whos
```

Name	Size	Bytes	Class	Attributes
A	5x5	200	double	
Asp	5x5	216	double	sparse

Il comando `full` può essere usato per trasformare una memorizzazione sparsa in una memorizzazione piena.

```
>> B= full(Asp)
>> max(max(B-A))
ans =      0
```

## sparse Create sparse matrix

---

- $S = \text{sparse}(X)$  converts a sparse or full matrix to sparse form by squeezing out any zero elements.



$$S = \text{sparse}(i, j, s, m, n, \text{nzmax})$$

uses the rows of  $[i, j, s]$  to generate an  $m$ -by- $n$  sparse matrix with space allocated for  $\text{nzmax}$  nonzeros.

- $i$  and  $j$  are two integer index vectors

- $s$  vector with real or complex entries

all have the same length,  $\text{nnz}$ , which is the number of nonzeros in the resulting sparse matrix  $S$ .

- Any elements of  $s$  which have duplicate values of  $i$  and  $j$  are added together.

Memorizzazione di  $A$  in forma sparsa.

Scegliamo di costruire la matrice a partire dalle diagonal.



## Esempio

```
A0=sparse([1:5],[1:5],[2 4 3 3 3],5,5) % Diag Principale
A1=sparse(1:4,2:5,[1 1 1 1],5,5);% Diag superiore
AL1=sparse(2:5,1:4,[4 5 5 5],5,5);% Diag inferiore 1
AL2=sparse(3:5,1:3,[6 6 6],5,5);% Diag inferiore 2
Ab=A0+A1+AL1+AL2;
max(max(Ab-A))
```

Fattorizzazione LU senza Pivot:

$$[L, U, P] = \text{lu}(Ab, \text{thresh})$$

la variabile `thresh` controlla il pivoting nelle matrici sparse.

Si effettua lo scambio quando l'elemento diagonale ha valore in modulo  $< \text{thresh}$ : `thresh = 0` elimina di fatto lo scambio delle righe.

```
[L,U,P]=lu(Ab,0);
spy(L) %solo per visualizzazione della struttura
spy(U)
```

La struttura a banda si mantiene anche in  $L$  ed  $U$ .

# Matrici tridiagonali

Si considera il sistema tridiagonale:  $\mathbf{A}\mathbf{x} = \mathbf{z}$  con:

$$\mathbf{A} = \begin{bmatrix} a_1 & c_1 & & & 0 \\ b_2 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & & & b_n & c_n \end{bmatrix}$$

Si ha la seguente decomposizione:  $\mathbf{A} = \mathbf{L}\mathbf{U}$

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & & & 0 \\ \beta_2 & 1 & 0 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & 1 & 0 \\ 0 & & & \beta_n & 1 \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \alpha_1 & c_1 & & & 0 \\ 0 & \alpha_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & 0 & \alpha_{n-1} & c_{n-1} \\ 0 & & & 0 & \alpha_n \end{bmatrix}$$

# Algoritmo di Thomas

---

Dove  $\alpha_i, \beta_i$  sono determinati come segue:

$$\alpha_1 = a_1$$

$$\beta_k = \frac{b_k}{\alpha_{k-1}}, \quad \alpha_k = a_k - \beta_k c_{k-1}, \quad k = 2, 3, \dots, n$$

Il sistema diventa allora:

$$\mathbf{LUx} = \mathbf{z}, \quad \begin{cases} \mathbf{Ly} = \mathbf{z} \\ \mathbf{Ux} = \mathbf{y} \end{cases}$$

da cui

$$y_1 = z_1, \quad y_i = z_i - \beta_i y_{i-1}, \quad i = 2, 3, \dots, n$$

$$x_n = \frac{y_n}{\alpha_n}, \quad x_i = \frac{y_i - c_i x_{i+1}}{\alpha_i}, \quad i = n-1, \dots, 2, 1$$

## function thomas

---

```
function x= thomas(A,b)
alpha(1)=A(1,1);
n=length(b);
for k=2:n
    beta(k)=A(k,k-1)/alpha(k-1);
    alpha(k)=A(k,k)-beta(k)*A(k-1,k);
end
y(1)=b(1);
for i=2:n
    y(i)=b(i)-beta(i)*y(i-1);
end
x(n)=y(n)/alpha(n);
for i= n-1:-1:1
    x(i)=(y(i)-A(i,i+1)*x(i+1))/alpha(i);
end
x=x';
```

# Esempio

Studio degli errori e dei residui con la matrice tridiagonale simmetrica:

```
e=ones(n,1);  
A=sparse(1:n,1:n,2*e,n,n)+...  
  sparse(2:n,1:n-1,-1*e(1:n-1),n,n)+...  
  sparse(1:n-1,2:n,-1*e(1:n-1),n,n);
```

