Analisi Numerica I
Sistemi lineari: metodi diretti

Ana Alonso

ana.alonso@unitn.it

11 - 18 ottobre 2019

Comandi per matrici...

...quadrate

- `det(A)`
- `inv(A)`
- `eig(A)`

e non

- `norm(A)`, `norm(A,inf)`, `norm(A,1)`.
- `cond(A)` (e per matrici quadrate anche `cond(A,inf)` `cond(A,1)`).

Sistemi lineari $A\mathbf{x} = \mathbf{b}$

Il comando di Matlab per risolvere sistemi lineari è il comando "\".

- Se A è invertibile `A\b` da lo stesso risultato di `inv(A)*b`.
- Non calcola A^{-1}.
- Vedi `help mldivide`.
- **b** potrebbe essere anche una matrice con tante righe come colonne in A.
- Se A è rettangolare `A\b` risolve il sistema lineare nel senso dei minimi quadratti (A e **b** devono avere lo stesso numero di righe) .
- Non confondere \ con /

$$A\backslash B \to inv(A) * B \quad A/B \to A * inv(B)$$

Matrici triangolari: I comandi `tril, triu, diag`

Data una matrice A (non necessariamente quadrata) provare i seguenti comandi:

- `tril(A), tril(A,1), tril(A,-2)`;
- `triu(A), triu(A,1), triu(A,-2)`;
- `diag(A), diag(A,1), diag(A,-2)`.

Dato un vettore **v** provare

- `diag(v), diag(v,1), diag(v,-2)`.

Risoluzione di un sistema triangolare inferiore

$$\begin{bmatrix} a_{1,1} & 0 & 0 & \dots & 0 & 0 \\ a_{2,1} & a_{2,2} & 0 & \dots & 0 & 0 \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & 0 & 0 \\ \vdots & \vdots & & \ddots & & \vdots \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \dots & a_{n-1,n-1} & 0 \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

$a_{i,i} \neq 0,\ i = 1, \dots, n.$

Sostituzione in avanti:

$$x_1 = b_1 / a_{1,1}$$

$$\textit{For} \quad i = 2 : n$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j \right)$$

function forward

La seguente funzione implementa il metodo della sostituzione in avanti:

```
function x=forward(A,b)
% Risolve il sistema lineare tril(A)*x=b usando il
% metodo della sostituzione in avanti.
n=length(b);
x=zeros(n,1);
x(1)=b(1)/A(1,1);
for i=2:n
    x(i)=(b(i)-A(i,1:i-1)*x(1:i-1))/A(i,i);
end
```

function forward

La seguente funzione implementa il metodo della sostituzione in avanti:

```
function x=forward(A,b)
% Risolve il sistema lineare tril(A)*x=b usando il
% metodo della sostituzione in avanti.
n=length(b);
x=zeros(n,1);
x(1)=b(1)/A(1,1);
for i=2:n
    x(i)=(b(i)-A(i,1:i-1)*x(1:i-1))/A(i,i);
end
```

$$x_1 = b_1/a_{1,1}$$

$$For \quad i = 2 : n$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j \right)$$

function forward

Avendo inizializzato x a zero possiamo anche scrivere:

```
function x=forward(A,b)
% Risolve il sistema lineare tril(A)*x=b usando il
% metodo della sostituzione in avanti.
n=length(b);
x=zeros(n,1);
x(1)=b(1)/A(1,1);
for i=2:n
    x(i)=(b(i)-A(i,:)*x)/A(i,i);
end
```

$$x_1 = b_1 / a_{1,1}$$

$$For \quad i = 2 : n$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j \right)$$

Risoluzione di un sistema triangolare superiore

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \dots & a_{2,n-1} & a_{2,n} \\ 0 & 0 & a_{3,3} & \dots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & \dots & 0 & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}$$

$a_{i,i} \neq 0,\ i = 1, \dots, n.$

Sostituzione all'indietro

$$x_n = b_n / a_{n,n}$$

$$\textit{For} \quad i = n-1 : -1 : 1$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=i+1}^{n} a_{i,j} x_j \right)$$

Esercizio

- Scrivere una funzione di Matlab che implemeti il metodo della sostituzione all'indietro.

Esercizio

- Scrivere una funzione di Matlab che implemeti il metodo della sostituzione all'indietro.

```
function x=backward(A,b)
% Risolve il sistema lineare triu(A)*x=b usando il
% metodo della sostituzione all'indietro.
n=length(b);
x=zeros(n,1);
x(n)=b(n)/A(n,n);
for i=n-1:-1:1
    x(i)=(b(i)-A(i,i+1:end)*x(i+1:end))/A(i,i);
end
```

Esercizio

- Scrivere una funzione di Matlab che implemeti il metodo della sostituzione all'indietro.

```
function x=backward(A,b)
% Risolve il sistema lineare triu(A)*x=b usando il
% metodo della sostituzione all'indietro.
n=length(b);
x=zeros(n,1);
x(n)=b(n)/A(n,n);
for i=n-1:-1:1
    x(i)=(b(i)-A(i,i+1:end)*x(i+1:end))/A(i,i);
end
```

$$x_n = b_n / a_{n,n}$$

$$For i = n-1 : -1 : 1$$

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=i+1}^{n} a_{i,j} x_j \right)$$

Il metodo di eliminazione di Gauss

$$A\mathbf{x} = \mathbf{b} \quad \rightsquigarrow \quad U\mathbf{x} = \mathbf{c}$$

```
for k=1:N-1
  for i=k+1:N
    m(i,k)=a(i,k)/a(k,k);
    for j=k+1:n
      a(i,j)=a(i,j)-m(i,k)*a(k,j);
    end
    b(i)=b(i)-m(i,k)*b(k)
  end
end
```

MEG senza pivoting

```
function x=gaussNPO(A,b)
n=length(b);
M=zeros(n);
for k=1:n-1
    for i=k+1:n
      M(i,k)=A(i,k)/A(k,k);
      for j=k+1:n
        A(i,j)=A(i,j)-M(i,k)*A(k,j);
      end
      b(i)=b(i)-M(i,k)*b(k);
    end
end
x=backward(A,b);
```

MEG senza pivoting

Non è necessario costruire la matrice M: possiamo usare la parte triangolare inferiore di A per salvare i moltiplicatori.

```
function [x,A]=gaussNP1(A,b)
n=length(b);

for k=1:n-1
    for i=k+1:n
      A(i,k)=A(i,k)/A(k,k);
      for j=k+1:n
        A(i,j)=A(i,j)-A(i,k)*A(k,j);
      end
      b(i)=b(i)-A(i,k)*b(k);
    end
end
x=backward(A,b);
```

MEG senza pivoting

Senza i cicli in i e in j:

```
function [x,A]=gaussNP(A,b)
n=length(b);
for k=1:n-1
    if A(k,k)==0, error('Elemento pivotale nullo'), 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);
    b(k+1:n)=b(k+1:n)-A(k+1:n,k)*b(k);
end
x=backward(A,b);
```

MEG con pivoting

```
function [x,A,aux]=gaussP(A,b)
n=length(b);
aux=1:n;
for k=1:n-1
    % Ricerca del pivot
    npivot=abs(A(k,k));
    r=k;
    for i=k+1:n
       if abs(A(i,k))>npivot
          npivot=abs(A(i,k));
          r=i;
       end
    end
    if npivot==0, error('Matrice singolare'); end
```

MEG con pivoting

```
    % Scambio delle righe
    if r~=k,
      A([k,r],:)=A([r,k],:);
      b([k,r])=b([r,k]);
      aux([k,r])=aux([r,k]);
    end
    % Eliminazione di Gauss
    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);
    b(k+1:n)=b(k+1:n)-A(k+1:n,k)*b(k);
end
if A(n,n)==0,  error('Matrice singolare'); end
x=backward(A,b);
```

Fattorizzazione LU di una matrice A

$$PA = LU$$

Per calcolare la fattorizzazione LU di una matrice A si usa il comando `lu`.

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

L è triangolare inferiore e i suoi elementi diagonali sono uguali a 1;
U è triangolare superiore,
P è una matrice di permutazione e

$$LU = PA\,.$$

Fattorizzazione LU di una matrice A

Il comando `lu` risponde in modo diverso a seconda di "quanti risultati chiediamo". Sia $A \in \mathbb{R}^{N \times N}$

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

L è triangolare inferiore e i suoi elementi diagonali sono uguali a 1; U è triangolare superiore, P è una matrice di permutazione e $LU = PA$.

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

Una permutazione delle righe di L è triangolare inferiore con elementi diagonali uguali a 1; U è triangolare superiore e $LU = A$.

```
>> M=lu(A)
```

Se `U= triu(M)` e `L=tril(M,-1)+eye(N)` allora LU è una permutazione delle righe di A.

Fattorizzazione LU di una matrice A

Ma anche la funzione `gaussP` calcola la fattorizzazione LU di A.

```
>> A=rand(7);
>> b=A*ones(7,1);
>> [x,M,p]=gaussP(A,b);
>> U=triu(M);
>> L=tril(M,-1)+eye(7);
>> L*U-A(p,:)
```