SOLUZIONE DI SISTEMI LINEARI

Metodi di calcolo numerico per la soluzione di ***sistemi di equazioni lineari***

Equazione lineare:

$$a_1X_1 + a_2X_2 + \ldots + a_NX_N = b$$

- incognite: $X_1, \ldots, X_N$
- coefficienti: $a_1, a_2, \ldots, a_N, b$

Soluzione dell'equazione: tupla $(X_1,\ldots X_N)$ che la verifica

Esempio:

- La terna (1,1,5) è una soluzione dell'equazione lineare:

$$X_1 - 2X_2 + X_3 = 4$$

SISTEMI LINEARI

- Si trovano in ***molti campi dell'ingegneria*** (es: circuiti elettrici), nella soluzione di equazioni differenziali, …

- Un ***"sistema lineare di m equazioni in n incognite"*** è un sistema di *m* equazioni nelle n incognite $X_1, X_2,...,X_N$

$$a_{11}X_1 + a_{12}X_2 + ...+a_{1N}X_N = b_1$$
$$a_{21}X_1 + a_{22}X_2 + ...+a_{2N}X_N = b_2$$
$$...$$
$$a_{M1}X_1 + a_{M2}X_2 + ...+a_{MN}X_N = b_M$$

- Risolvere un sistema di questo tipo, ***significa trovare un insieme di valori per le variabili che soddisfi simultaneamente tutte le equazioni***

SISTEMI LINEARI

Ovviamente vi ricordate ☺:

- Siamo interessati ai sistemi in cui il numero di equazioni è uguale al numero di incognite: $m = n$

  In questo caso la ***soluzione è unica***

- Se il numero di equazioni è ***minore delle incognite***, la ***soluzione non è unica***

- Se il numero di equazioni è maggiore delle incognite, è possibile che:
  - alcune equazioni siano ***dipendenti*** (*combinazioni lineari*) da altre (e si possono eliminare)
  - Oppure il ***sistema sia indeterminato***

SISTEMI LINEARI

Rappresentazione compatta (tramite matrici):

$$A * X = B$$

dove **A** (matrice dei coefficienti):

$$A = \begin{bmatrix} a_{11} & a_{12} & . & . & a_{1N} \\ a_{21} & a_{22} & . & . & a_{2N} \\ . & & & & \\ . & & & & \\ a_{N1} & . & . & . & a_{NN} \end{bmatrix}$$

B vettore dei termini noti e **X** quello delle soluzioni:

$$B = \begin{bmatrix} b_1 \\ b_2 \\ . \\ b_N \end{bmatrix} \qquad X = \begin{bmatrix} X_1 \\ X_2 \\ . \\ X_N \end{bmatrix}$$

Queste equazioni hanno ***soluzione unica se det(A) è diverso da zero (matrice non singolare)***

RISOLUZIONE DI SISTEMI LINEARI

Si possono applicare due metodi:

- ***metodi diretti***, basati su trasformazioni in sistemi di equazioni equivalenti; forniscono sempre la soluzione **esatta**
- ***metodi indiretti o iterativi***, basati su successive approssimazioni. Forniscono soluzioni non esatte (approssimate): grado di precisione specificato dall'utente e influenza tempo di esecuzione

Criteri di Scelta:

- metodi ***diretti*** hanno numero di operazioni da eseguire finito e prefissato, che dipende dalle dimensioni della matrice; quelli ***iterativi*** non è detto che convergano
- Se ***matrice sparsa*** convengono in genere quelli ***iterativi*** poiché tempo di calcolo è proporzionale a numero degli elementi diversi da zero

METODI DIRETTI

- L'idea principale è quella ***dell'eliminazione***: si ricava da un'equazione ***una particolare incognita*** e la si ***sostituisce*** nelle rimanenti
  - si diminuisce di 1 dimensione del problema. Quando si arriva a determinare un valore, si procede a ritroso e si trovano gli altri

Equivalenza:

- Due sistemi di equazioni lineari nello stesso numero di incognite ***sono equivalenti se hanno stesse soluzioni***

- Si può ottenere un ***sistema equivalente***:
  - scambiando a due a due le equazioni
  - moltiplicando ogni equazione per una costante diversa da zero
  - sommando ad ogni equazione un'altra equazione moltiplicata per una costante

EQUIVALENZA DI SISTEMI LINEARI

Esempio: $2X + Y - Z = 5$
$3X - 2Y + 2Z = -3$
$X - 3Y - 3Z = -2$

- Dividiamo ogni equazione per il coefficiente di X:
  $X + Y/2 - Z/2 = 5/2$
  $X - 2Y/3 + 2Z/3 = -1$
  $X - 3Y - 3Z = -2$

- Sottraiamo la prima equazione dalla seconda e dalla terza:
  $X + Y/2 - Z/2 = 5/2$
  $-7Y/6 + 7Z/6 = -7/2$
  $-7Y/2 - 5Z/2 = 9/2$

- Dividiamo per il coeff. del primo termine:
  $X + Y/2 - Z/2 = 5/2$
  $Y - Z = 3$
  $Y + 5Z/7 = 9/7$

EQUIVALENZA DI SISTEMI LINEARI

- Sottraendo la seconda equazione dalla terza:

X +Y/2 -Z/2 = 5/2

Y -Z = 3

12Z/7 = -12/7

- Il sistema diventa in forma ***triangolare superiore***

$$\begin{bmatrix} 1 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & -1 \\ 0 & 0 & \frac{12}{7} \end{bmatrix} . X = \begin{bmatrix} \frac{5}{2} \\ 3 \\ -\frac{12}{7} \end{bmatrix}$$

Ora si può calcolare direttamente il valore delle incognite:

Z = -1

Y = 3 -1 =2

X=5/2-1/2-2/2= 1

METODO DI GAUSS

2 Fasi:

- ✶ ***Triangolarizzazione*** della matrice dei coefficienti
- ✸ ***Eliminazione*** all'indietro => Calcolo della soluzione

Triangolarizzazione

- ***Eliminazione dell'incognita X_1***: se a_{11} è diverso da zero si può eliminare X_1 dalle righe 2,3,...n sommando alla generica riga i-ma la prima moltiplicata per

$$m_{i1} = -a_{i1}/a_{11} \quad (i = 2, 3, \ldots, n)$$

- Dopo questa operazione, nella matrice risultante ***mancheranno i coeff. a_{i1} i=2,3,...n*** mentre il generico elemento $a_{ij}^{(2)} = a_{ij} - m_{i1}a_{1j}$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ 0 & a_{22}^{(2)} & \cdots & a_{2N}^{(2)} \\ 0 & \cdots & \cdots & \cdots \\ 0 & a_{N2}^{(2)} & \cdots & a_{NN}^{(2)} \end{bmatrix}$$

METODO DI GAUSS

Ad ogni passo *k* del procedimento (ripetuto *n-1* volte) si elimina X_k con la stessa tecnica:

$m_{ik} = -a_{ik}^{(k)} / a_{kk}^{(k)}$ (i = k+1,..n)

$a_{ij}^{(k+1)} = a_i^{j\,(k)} - m_{ik} a_{kj}^{(k)}$ (i=k+1,...n) (j = k+1,..n+1)

Si ottiene una ***matrice triangolare superiore***:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1N}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2N}^{(2)} \\ 0 & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & a_{NN}^{(n)} \end{bmatrix}$$

METODO DI GAUSS
al passo k-esimo vale:

$$X_N = \frac{b_N^{(k)}}{a_{NN}^{(k)}}$$

$$X_i = \frac{(b_i^{(k)} - \sum_{j=i+1}^{N} a_{ij}^{(k)} . X_j)}{a_{ii}^{(k)}}$$

- Numero di calcoli da eseguire è ***proporzionale a $n^3/3$***

METODO DI GAUSS

- ***Algoritmo di triangolarizzazione***:

```
for (k=1; k<n; k++)
    for (i=k+1; i<=n; i++)
            {
            m_ik = a_ik^(k)/a_kk^(k)
            for (j = k; j<=n; j++)
                    a_ij^(k+1) = a_ij^(k) - m_ik * a_kj^(k)
            }
```

Il sistema triangolare così ottenuto può essere risolto facilmente con la procedura seguente

METODO DI GAUSS: Eliminazione

- ***Algoritmo di eliminazione all'indietro***:
  - Data una matrice **U**(n x n) triangolare superiore non singolare, e un vettore Y di dimensione n la soluzione del sistema

$$UX = Y \ldots$$

$x_n = y_n/u_{nn}$;
for (i = n-1; i>=0; i--) {
  for (j=i+1; j<=n; j++) $x_i=x_i + u_{ij}*x_j$;
  $x_i = (y_i - x_i)/u_{ii}$;
  }

METODO DI JORDAN

- VARIANTE: se al generico passo k-esimo il processo di eliminazione non viene effettuato solo sulle righe successive alla k-esima ma anche sulle precedenti otteniamo dopo n passi un sistema diagonale (**metodo di Jordan**)
  - In particolare, la formula:

$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} * a_{kj}^{(k)}$ $\quad j = k+1, \ldots, n$

si applica per ogni i da 1 a n e diverso da k

- Non richiede la propagazione indietro, ma è computazionalmente più costoso $O(n^3/2)$; ***preferibile il metodo di Gauss***

Metodo di GAUSS : ESEMPIO

Triangolazione:

$$
\begin{array}{llll}
2X_1 & -X_2 & +X_3 & -2X_4 & =0 \\
 & 2X_2 & & -X_4 & =1 \\
X_1 & & -2X_3 & +X_4 & =0 \\
 & 2X_2 & +X_3 & +X_4 & =4
\end{array}
$$

Passo 1:

$$
\begin{array}{llll}
2X_1 & -X_2 & +X_3 & -2X_4 & =0 \\
 & 2X_2 & & -X_4 & =1 \\
 & 1/2X_2 & -5/2X_3 & +2X_4 & =0 \\
 & 2X_2 & +X_3 & +X_4 & =4
\end{array}
$$

Passo 2:

$$
\begin{array}{llll}
2X_1 & -X_2 & +X_3 & -2X_4 & =0 \\
 & 2X_2 & & -X_4 & =1 \\
 & & -5/2X_3 & +9/4X_4 & =-1/4 \\
 & & +X_3 & +2X_4 & =3
\end{array}
$$

Passo 3:

$$
\begin{array}{llll}
2X_1 & -X_2 & +X_3 & -2X_4 & =0 \\
 & 2X_2 & & -X_4 & =1 \\
 & & -5/2X_3 & +9/4X_4 & =-1/4 \\
 & & & +29/10X_4 & =29/10
\end{array}
$$

Metodo di Gauss : ESEMPIO

Partendo dall'ultima equazione (del sistema triangolare finale) otteniamo:

$X_4=1$

$X_3\ (-1/4 - 9/4X_4)(-2/5) = 1$

$X_2=(1+X_4)/2 = 1$

$X_1 = (X_2-X_3+2X_4)/2 = 1$

PIVOTING

Problema:

se $a_{kk}=0$, processo di triangolarizzazione impossibile:

$m_{ik}=a_{ik}/a_{kk}$ => **divisione per zero!**

Soluzione (ovvia):

Scambio di equazioni:

se $a_{kk}^{(k)}=0$ ed esiste $a_{rk}^{(k)} \neq 0$

- si scambia l'equazione *r*-sima con l'equazione *k*-sima: => il nuovo a_{kk} è diverso da zero

- Quindi, per poter eseguire il metodo di costruzione della matrice triangolare, ad ogni passo si procede con un eventuale scambio di equazioni

PIVOTING

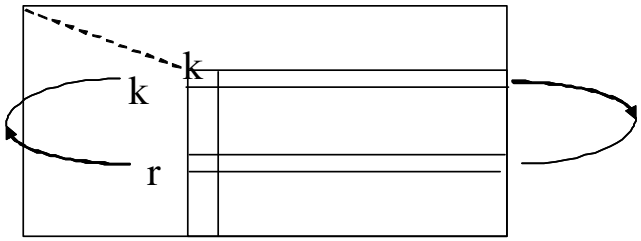
Ulteriore Problema:

Ad ogni passo k, se il ***valore assoluto di a_{kk} è prossimo allo zero***, la propagazione degli errori viene ***amplificata***

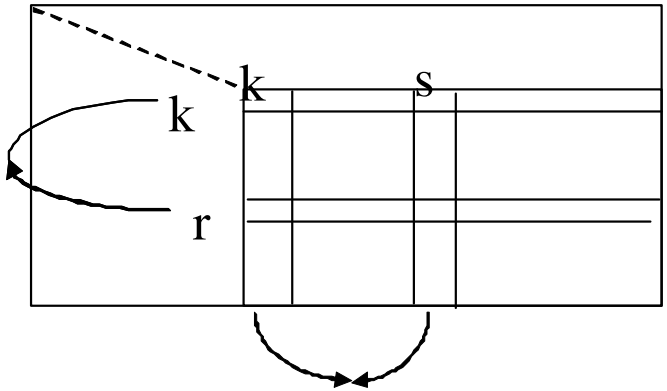
Soluzione: Pivoting

Occorre scambiare l'equazione *k*-sima con una equazione (r-sima) tale che il valore assoluto di a_{rk} risulti:

- il più grande tra tutti gli a_{ik} della sottomatrice (i=k, ..., N) (ricerca sulla *colonna* della sottomatrice) => ***pivoting parziale***
- il più grande tra tutti gli a_{ij} (i=k,..N; j=k,..N) (ricerca sulle righe e sulle colonne della sottomatrice): in questo caso si provvede anche ad un eventuale **scambio di incognite** (se j≠k) => ***pivoting completo***



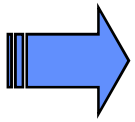
$$r: \ |a_{rk}| = \max_{k \le i \le N} |a_{ik}|$$



$$r,s: \ |a_{rs}| = \max_{\substack{i=k,..N \\ j=k,..N}} |a_{ij}|$$

PIVOTING

- ***Pivoting completo*** è procedimento ***costoso***
- Generalmente, ***pivoting parziale*** dà risultati ***soddisfacenti***
- Può accadere che la sola tecnica di pivoting ***non sia sufficiente a limitare la propagazione di errori***. Ad esempio:
  - se tutti gli elementi della matrice A hanno valore vicino allo zero
  - se gli elementi di A hanno valori molto diversi tra loro (ordini di grandezza)



In questi casi è necessario ***ri-equilibrare*** la matrice con un ***procedimento di Scaling***

SCALING

- Si normalizzano gli elementi di ciascuna riga della matrice assumendo come valore di riferimento, quello del pivot della riga $\boldsymbol{d_i}$ (*dimensione*):

$$d_i = \max_{j=1,\ldots,N} \left| a_{ij} \right|$$

- Quindi, al generico passo *k*-esimo si assume come equazione pivotale (fra le n-k rimanenti) ***la r-sima***, in modo tale che:

$$\frac{\left| a_{rk} \right|}{d_r} = \max_{i=k,\ldots,N} \frac{\left| a_{ik} \right|}{d_i}$$

GAUSS: Pivoting

```
#include <stdio.h>
#include <math.h>
#define N 4
typedef float matrice[N][N+1]; // PERCHE'????

void pivot(matrice A, int k, int dim) {       // pivoting
   int i, max;                                // parziale
   float tmp;
   printf("pivot al passo %d...\n", k);
   max=k;
   for (i=k+1; i<dim; i++)
      if (fabs(A[i][k])>fabs(A[max][k])) max=i;
   if (max!=k)
      for(i=0; i<dim+1;i++) {
            tmp=A[k][i];
            A[k][i]=A[max][i];
            A[k][i]=tmp;
      }
   return;
}
```

GAUSS: Triangolar. + Eliminazione

```
void triangolarizza(matrice A, int dim) {
   int i, j, k; float m;
   for (k=0; k<dim; k++) {
       pivot(A,k,dim);
       for (i=k+1; i<dim; i++) {
             m=-A[i][k]/A[k][k];
             for (j=k; j<dim+1; j++)
                   A[i][j]=A[i][j]+m*A[k][j];
       }
   }
   return;
}

void elim_indietro(matrice A,float *X,int dim) {
   int i,j;
   for (i=dim-1; i>=0; i--) {
       for (X[i]=0, j=i+1; j<dim; j++)
             X[i] -= A[i][j]*X[j];
       X[i] += A[i][dim];
       X[i]/=A[i][i];
    }
  return;
}
```

GAUSS: Invocazione

```
void leggi_m(matrice M,int righe,int colonne);
void stampa_v(float *V, int dim);


main() {
   matrice A;
   float X[N];
   leggi_m(A, N, N);
   triangolarizza(A,N);
   elim_indietro(A,X,N);
   printf("\nRisultato:\n");
   stampa_v(X, N);
   return 0;
}
```

GAUSS: Funzioni Ausiliarie

```
void leggi_m (matrice M,int righe,int colonne){
   int i,j;
   for (i=0; i<righe; i++)
       for (j=0; j<colonne; j++) {
            printf("Coeff. M[%d][%d]?", i, j);
            scanf("%f", &(M[i][j]));
        }
   for (i=0; i<righe; i++) {
      printf("\nTermine noto %d ?",i);
      scanf("%f", &(M[i][colonne]));
      }
}

void stampa_v (float *V, int dim) {
   int i;
   for (i=0; i<dim; i++)
      printf("\t%f\n", *(V+i));
}
```

DECOMPOSIZIONE LU (o Doolittle)

A.X=b

- Hp: supponiamo di poter esprimere la matrice A come prodotto di due matrici

A=L.U

dove

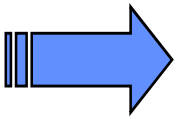
- ***L*** è una matrice ***triangolare inferiore***
- ***U*** è una matrice ***triangolare superiore***

$$L = \begin{bmatrix} \alpha_{11} & 0 & .. & 0 \\ \alpha_{21} & \alpha_{22} & .. & 0 \\ . & . & . & . \\ \alpha_{N1} & \alpha_{N2} & .. & \alpha_{NN} \end{bmatrix}$$

$$U = \begin{bmatrix} \beta_{11} & \beta_{12} & .. & \beta_{1N} \\ 0 & \beta_{22} & .. & \beta_{2N} \\ . & . & . & . \\ 0 & 0 & .. & \beta_{NN} \end{bmatrix}$$

DECOMPOSIZIONE LU

- **Allora A.X=(L.U).X=L.(U.X)=b**



il sistema originario è ***equivalente*** alla coppia di sistemi lineari:

[1] L.Y=b

[2] U.X=Y

- abbiamo raddoppiato il numero di equazioni da risolvere, ma i due sistemi sono già in ***forma triangolare***

[1] viene risolto con sostituzione in avanti (***forward substitution***)

[2] viene risolto con sostituzione indietro (***backward substitution***)

PROBLEMA: come realizzare la decomposizione LU?

DECOMPOSIZIONE LU

$$A = \begin{bmatrix} a_{11} & a_{12} & .. & a_{1N} \\ a_{21} & a_{22} & .. & a_{2N} \\ . & . & .. & . \\ a_{N1} & a_{N2} & .. & a_{NN} \end{bmatrix} =$$

$$= \begin{bmatrix} \alpha_{11} & 0 & .. & 0 \\ \alpha_{21} & \alpha_{22} & .. & 0 \\ . & . & . & . \\ \alpha_{N1} & \alpha_{N2} & .. & \alpha_{NN} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & .. & \beta_{1N} \\ 0 & \beta_{22} & .. & \beta_{2N} \\ . & . & . & . \\ 0 & 0 & .. & \beta_{NN} \end{bmatrix}$$

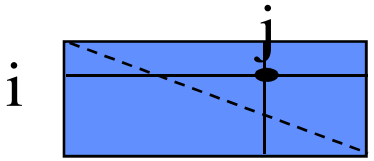
- Ogni elemento della matrice A è ottenuto come combinazione lineare di elementi delle due matrici L, U (prodotto riga per colonna):

$$a_{ij}=\alpha_{i1}.\beta_{1j}+ \alpha_{i2}.\beta_{2j} +\ldots+ \alpha_{iN}.\beta_{Nj}$$

DECOMPOSIZIONE LU

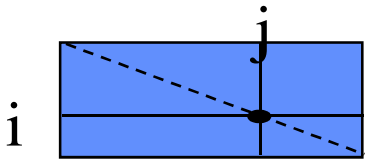
In questo caso particolare, essendo molti elementi delle due matrici L, U ***nulli***, valgono le semplificazioni:

- **i<j**



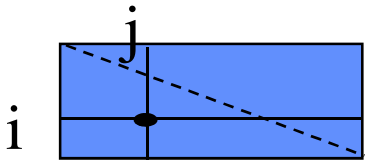
$$a_{ij}=\alpha_{i1}.\beta_{1j}+\alpha_{i2}.\beta_{2j}+\ldots+\alpha_{ii}.\beta_{ij}$$

- **i=j**



$$a_{ij}=\alpha_{i1}.\beta_{1j}+\alpha_{i2}.\beta_{2j}+\ldots+\alpha_{ii}.\beta_{jj}$$

- **i>j**



$$a_{ij}=\alpha_{i1}.\beta_{1j}+\alpha_{i2}.\beta_{2j}+\ldots+\alpha_{ij}.\beta_{jj}$$

DECOMPOSIZIONE LU

- Per calcolare gli elementi di L e U è necessario risolvere un sistema di N^2 equazioni in (N^2+N) incognite
  - Fissiamo arbitrariamente il valore di N incognite, ponendo

  $\alpha_{ii}=1 \qquad (i=1, \ldots, N)$

  e ricaviamo gli altri valori

- Mappiamo gli elementi di L e di U in un'unica matrice LU:

$$LU = \begin{bmatrix} \beta_{11} & \beta_{12} & .. & \beta_{1N} \\ \alpha_{21} & \beta_{22} & .. & \beta_{2N} \\ . & . & . & . \\ \alpha_{N1} & \alpha_{N2} & .. & \beta_{NN} \end{bmatrix}$$

- Il sistema $N^2 \times N^2$ può essere risolto con gli usuali metodi (es. Gauss) ma, ***grazie alla particolare configurazione delle matrici L e U***, è possibile anche ottenere la soluzione ***direttamente in N^2 passaggi***

ALGORITMO DI CROUT

- Stabilisce semplicemente ***l'ordine*** con cui calcolare gli elementi di L e U:

  1) si ponga $\alpha_{ii}=1$ (i=1, …, N)
  2) per ogni j=1, 2, …, N:

$$\forall i = 1,2,..\, j \quad \beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj}$$

$$\forall i = j+1,\, j+2,..,\, N \quad \alpha_{ij} = \frac{1}{\beta_{jj}}(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj})$$

ALGORITMO DI CROUT

In pratica, si considera una colonna per volta della matrice LU (j=1,..N):

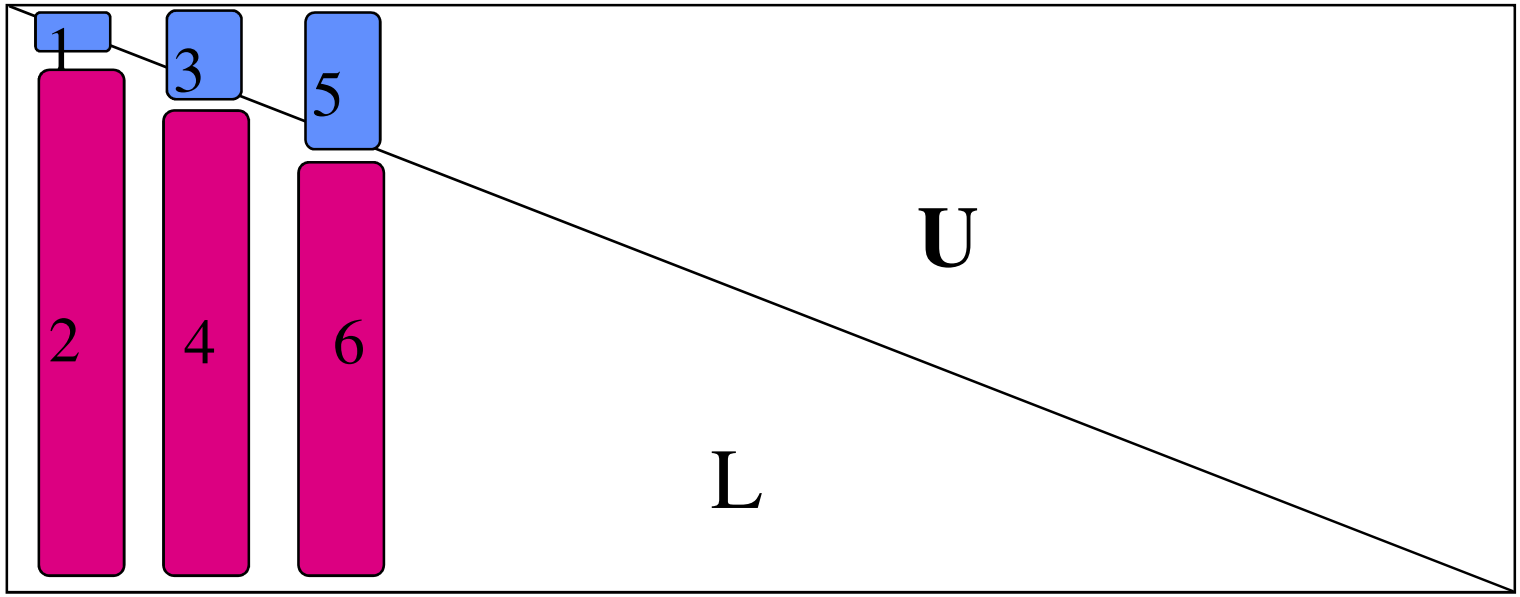
- **j=1** **i<=j**
  - i=1: $a_{11}=\alpha_{11}\beta_{11}$ -> $\beta_{11}=a_{11}$ (i<=j)

  i>j
  - i=2: $a_{21}=\alpha_{21}\beta_{11}$ -> $\alpha_{21}=a_{21}/\beta_{11}$
  - i=3: $a_{31}=\alpha_{31}\beta_{11}$ -> $\alpha_{31}=a_{31}/\beta_{11}$
  - ...
  - i=N: $a_{N1}=\alpha_{N1}\beta_{11}$ -> $\alpha_{N1}=a_{N1}/\beta_{11}$
- **j=2** **i<=j**
  - i=1: $a_{12}=\alpha_{11}\beta_{12}$ -> $\beta_{12}=a_{12}$ (i<=j)
  - i=2: $a_{22}=\alpha_{21}\beta_{12}+\alpha_{22}\beta_{22}$ -> $\beta_{22}=a_{22}-\alpha_{21}\beta_{12}$

  i>j
  - i=3: $a_{32}=\alpha_{31}\beta_{12}+\alpha_{32}\beta_{22}$ -> $\alpha_{32}=(a_{32}-\alpha_{31}\beta_{12})/\beta_{22}$
  - ...
  - i=N: $a_{N2}=\alpha_{N1}\beta_{12}+\alpha_{N2}\beta_{22}$ -> $\alpha_{N2}=(a_{N2}-\alpha_{N1}\beta_{12})/\beta_{22}$
  - ...

ALGORITMO DI CROUT

- Ad ogni passo dell'algoritmo, si utilizzano valori già calcolati nei passi precedenti
- Graficamente, gli elementi della matrice LU vengono calcolati nell'ordine:



- Problema: ***stabilità dell'algoritmo di Crout*** => necessità di pivoting

⇒ Applichiamo l'algoritmo di Crout su una ***permutazione P*** della matrice A, ottenuta con pivoting parziale **P.A=L.U**

SOLUZIONE SISTEMI LINEARI tramite DECOMPOSIZIONE LU

```
#include <stdio.h>
#include <stdlib.h>
#include "matrici.h"

matrice A, LU;
float X[N];

int main() {
   leggi_m(A, N, N);
   ludcmp(A, LU, N);
   lu_sol(LU,X,N);

   stampa_v(X, N);
}
```

CALCOLO DELLA MATRICE LU

```
void ludcmp (matrice A, matrice LU, int dim) {
   int i,j,k;
   for(j=0; j<dim; j++) {
      for(i=0; i<=j; i++) {
            LU[i][j]=A[i][j];
            for(k=0;k<i;k++) LU[i][j] -= LU[i][k]*LU[k][j];
      }
      for (i=j+1; i<dim; i++) {
            LU[i][j]=A[i][j];
            for (k=0; k<j; k++) LU[i][j]-=LU[i][k]*LU[k][j];
            LU[i][j]/=LU[j][j];
      }
   }
 /* copia dei termini noti nell'ultima colonna di LU */
   for (i=0; i<dim; i++) LU[i][dim]=A[i][dim];

return;
}
```

CALCOLO DELLA SOLUZIONE

```
void lu_sol (matrice LU, float *X, int dim) {
   int i, j; float Y[N];

   /*LY=b sostituzione in avanti: ottengo Y */
  for (i=0; i<dim; i++) {
      Y[i]=LU[i][dim]; /* termine b[i]*/
      for (j=0; j<i; j++)
            Y[i] -= LU[i][j]*Y[j];
   }

   /*UX=Y: sostituzione all'indietro */
  for(i=dim-1; i>=0; i--) {
      X[i]=Y[i]; /*termine  noto Y[i] */
      for(j=i+1; j<dim; j++)
            X[i] -= LU[i][j]*X[j];
      X[i]/=LU[i][i];
   }

return;
}
```

INVERSIONE DI MATRICE QUADRATA

- Data una matrice quadrata A di ordine N non singolare, si definisce ***matrice inversa di A*** la matrice quadrata A^{-1} di ordine N: $A^{-1} * A = I$

dove I è la matrice "Identità" di ordine N (tale che gli elementi sulla diagonale principale siano uguali a 1 e tutti gli altri siano uguali a 0)

- Calcolo della matrice inversa comodo per risolvere più sistemi di equazioni lineari con la stessa matrice dei coefficienti:

  AX=b'

  AX=b''

  ...

MATRICE INVERSA

- Infatti il sistema si può scrivere come: **A X = b**

  e moltiplicando per A^{-1} si ottiene:

  $X = A^{-1} * b$

- Se indichiamo con $q_{i,j}$ (i,j=1,...n) gli elementi della matrice inversa:

$$\forall k = 1,..N:$$

$$\sum_{i=1}^{N} a_{ik} \cdot q_{ki} = \begin{cases} 0 & se \quad i \neq k \\ 1 & se \quad i = k \end{cases}$$

  - Risoluzione di N^2 equazioni lineari in N^2 incognite
  - Per risolverle, basta risolvere N sistemi lineari ognuno di essi ottenuto per un valore di j

MATRICE INVERSA

- Ciascuno è un sistema lineare di n equazioni in n incognite che rappresentano la j colonna di A^{-1} $(q_{1,j}, q_{2,j},...,q_{n,j})$
- Per j=1:

  i=1 $\quad \sum_{(k=1,..n)} a_{1,k}.q_{k,1} = 1$

  i=2 $\quad \sum_{(k=1,..n)} a_{2,k}.q_{k,1} = 0$

  .....

  i=n $\quad \sum_{(k=1,..n)} a_{n,k}.q_{k,1} = 0$

- Per ogni j, la colonna dei termini noti coincide con la colonna j-esima della matrice unitaria
- Si devono ***risolvere n sistemi lineari con la stessa matrice dei coefficienti***
- Invece di risolvere separatamente n sistemi, si possono ottenere i valori $q_{k,j}$ applicando ***metodo di eliminazione diagonale o triangolare***

METODI ITERATIVI

- Utilizzati come alternativa ai metodi diretti ***quando la matrice è di ordine elevato o sparsa***
- Non alterano mai la matrice iniziale

Esempio:

$$10X_1 + X_2 + X_3 = 24$$
$$-X_1 + 20X_2 + X_3 = 21$$
$$X_1 - 2X_2 + 100X_3 = 300$$

Si ricava da ogni equazione una diversa incognita in funzione delle rimanenti:

$$X_1 = (24 - X_2 - X_3)/10$$
$$X_2 = (21 + X_1 - X_3)/20$$
$$X_3 = (300 - X_1 + 2X_2)/100$$

Si prende una soluzione di tentativo (ad es: (0, 0, 0)) e poi si itera fino a trovare una soluzione soddisfacente

METODO DI JACOBI
(o delle sostituzioni simultanee)

- Data una matrice A quadrata di dimensione N, con a_{ii} diverso da 0 (per ogni i) ed un vettore b, a partire da un vettore di tentativo $X^{(0)}$ si costruisce la successione $X^{(k)}$ mediante la formula:

$$\forall i = 1,\ldots,N \qquad X_i^{(k+1)} = \frac{1}{a_{ii}} \cdot \left(b_i - \sum_{j=1,\, j \neq i}^{N} a_{ij} X_j^{(k)} \right)$$

- Il metodo richiede due vettori X_{old} (valori delle incognite all'iterazione precedente) e X_{new} (valori delle incognite all'iterazione corrente)
- Alla fine di ogni ciclo si pone X_{new}--> X_{old}. Le componenti sono costruite in modo indipendente (***possibilità di parallelismo***)
- Il metodo ***può anche non convergere*** (fissare bene i valori di primo tentativo)

METODO DI JACOBI: Esempio

Scriviamo le equazioni precedenti come:

$X_1 = 300 + 2X_2 - 100X_3$

$X_2 = 24 - 10X_1 - X_3$

$X_3 = 21 + X_1 - 20X_2$

partendo da (0, 0, 0) non si converge mai a (2, 1, 3)

METODO DI JACOBI

```
void copia(float *V1, float *V2, int dim);

int jacobi (matrice A, float *X, int dim, float acc) {
   int i,j,k, sol=0;
   float Xnew[N], err, errmax;
   for (k=1; k<=MAXITER && !sol; k++) {
      errmax=0;
      for (i=0; i<dim; i++) {
            Xnew[i]=A[i][dim];
            for(j=0; j<dim; j++)
              if (j!=i) Xnew[i] -= A[i][j]*X[j];
            Xnew[i]=Xnew[i]/A[i][i];
            err=fabs(Xnew[i]-X[i]);
            if (err>errmax) errmax=err; }
      if (errmax<acc) sol=1;
      else copia(X, Xnew, dim);
   }
   if (!sol) {
      printf("%d iterazioni !!", k);
      exit(); }
   else return k; }
```

METODO DI GAUSS-SEIDEL

- Diversamente da Jacobi, ***appena si calcola un valore, lo si utilizza***
- ***Non è garantita la convergenza***, ma se converge, lo fa ***più rapidamente del metodo di Jacobi***

Esempio:

$$10X_1 + X_2 + X_3 = 24$$
$$-X_1 + 20X_2 + X_3 = 21$$
$$X_1 - 2X_2 + 100X_3 = 300$$

Trasformiamo le equazioni nella forma:

$$X_1 = (24 - X_2 - X_3)/10$$
$$X_2 = (21 + X_1 - X_3)/20$$
$$X_3 = (300 - X_1 + 2X_2)/100$$

Si parte con un valore di tentativo (es:(0, 0)) per X_2 e X_3
Si calcola X_1 e si utilizza subito per calcolare X_2
Si utilizzano nuovi valori di X_1 e X_2 per calcolare X_3, e così via…

METODO DI GAUSS-SEIDEL

Data una matrice A di dimensione N con a_{ii} diverso da 0 ed un vettore b, a partire da un vettore di tentativo $X^{(0)}$ si costruisce la successione $X^{(k)}$ applicando la formula:

$$\forall i = 1,..N$$

$$X_i^{(k+1)} = \frac{1}{a_{ii}} \cdot \left(b_i - \sum_{j=1}^{i-1} a_{ij} X_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} X_j^{(k)} \right)$$

Il metodo richiede ***un solo vettore X***

METODO DI GAUSS-SEIDEL

```
#define MAXITER 1000;
int Gseidel (float **A, float *b, int N, float xacc, float
   **X) { /*restituisce numero iterazioni, A matrice
   coefficienti, b vettore termini noti, xacc precisione, X
   vettore soluzioni */

int i, j, k, sol=0; float err, errmax;
for (k=1; k<=MAXITER || sol; k++){  /*ciclo approx.success.*/
   errmax=0;
   for (i=0; i<N; i++) { /*i indice di riga */
      y=X[i]; X[i]=b[i];
      for(j=0; j<N; j++)
            if (i!=j) X[i]=X[i]-A[i][j]*X[j];
      X[i]=X[i]/A[i][i]; /*valore i-sima incognita passo k*/
      if (err=fabs(y-X[i])<errmax) errmax=err;
   }
   if (errmax<xacc) sol=1;
}
if (!sol)   printf("troppe iterazioni\n");
else return k; }
```