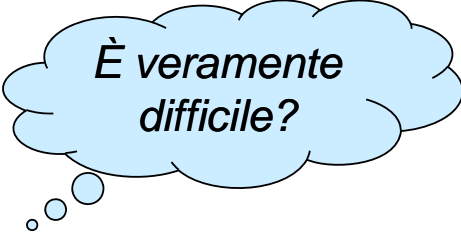


Matrici

- Un po' di esercizi sulle matrici
- Semplici
 - Lettura e scrittura
 - Calcolo della trasposta
- Media difficoltà
 - Calcolo del determinante
- Difficili
 - Soluzione di sistemi lineari



È veramente difficile?

Matrici.h – Definizione dei tipi

```
#define MAXROWS 10
#define MAXCOLS 10
#define ELEMENT int
#define ELEMENTFORMAT "%d"

#ifndef MATRICE
#define MATRICE

typedef ELEMENT Matrice[MAXROWS][MAXCOLS];

#endif
```

Tipo degli elementi della matrice

Formato di stampa degli elementi della matrice

Perché potrebbe esserci la necessità di #ifndef...?
Non è possibile avere definizioni multiple di un tipo!

Lettura di una matrice

- Leggere da console una matrice di r righe e c colonne...
- Parametri:
 - La matrice da leggere (obbligatorio riferimento – va bene così)
 - La dimensione (righe, colonne)
- Pseudocodice:
 - Leggere la matrice riga per riga – elemento per elemento → stampare un opportuno messaggio prima di richiedere i dati
 - Un ciclo per le righe
 - Un ciclo innestato nel precedente per le colonne
 - Per ogni riga della matrice
 - Indicare quale riga si sta leggendo
 - Leggere la riga con tante `scanf()` quante sono le colonne
 - Controllare che i valori siano stati tutti convertiti correttamente, altrimenti uscire dalla funzione con un codice d'errore opportuno

Lettura di una matrice

```
CODICEUSCITA readMatrice(Matrice m, int righe, int colonne)
{
    char formato[100] = "";
    int riga, colonna;
    for (riga = 0; riga < righe; riga++)
    {
        printf("Inserire la riga %d: ", riga);
        for (colonna = 0; colonna < colonne; colonna++)
        {
            if (scanf(ELEMENTFORMAT, &m[riga][colonna]) != 1)
                return VALORINONVALIDI;
        }
        while (getchar() != (char)10);
    }
    return OK;
}
```

- **CODICEUSCITA** è il simbolo utilizzato anche nelle slide relative al calcolo delle radici
- **VALORINONVALIDI** è un simbolo aggiuntivo...

Stampa di una matrice

- Stampare a console una matrice di r righe e c colonne...
- Parametri:
 - La matrice da stampare
 - La dimensione (righe, colonne)
- Pseudocodice:
 - Predisporre la stringa di formato
 - Per ogni riga r
 - Per ogni colonna c
 - Stampare l'elemento di posizione r, c usando la stringa di formato

Stampa di una matrice

```
void printMatrice(Matrice m, int righe, int colonne)
{
    int riga, colonna;
    char formato[10] = "";
    strcat(formato, ELEMENTFORMAT);
    strcat(formato, "\t");
    for (riga = 0; riga < righe; riga++)
    {
        for (colonna = 0; colonna < colonne; colonna++)
            printf(formato, m[riga][colonna]);
        printf("\n");
    }
    printf("\n");
}
```

- Come eliminare l'uso del *tab*?
- Come minimizzare il numero degli spazi?

Trasposta di una matrice quadrata

- La definizione di trasposta è...
- Parametri:
 - Una matrice in ingresso
 - Una matrice in uscita
 - La dimensione della matrice
- Pseudo codice:
 - Per ogni riga r , colonna c
 - Porre l'elemento di posizione r, c nella matrice in ingresso nella posizione c, r nella matrice in uscita

Trasposta di una matrice quadrata

```
void trasposta(Matrice m, Matrice tm, int dim)
{
    int riga, colonna;
    for (riga = 0; riga < dim; riga++)
        for (colonna = 0; colonna < dim; colonna++)
            tm[colonna][riga] = m[riga][colonna];
}
```

- Come fare per calcolare la trasposta di una matrice qualsiasi?
- Come viene cambiato il codice?
- Attenzione all'allocazione dello spazio per le matrici nel chiamante

Determinante di una matrice

- Per una matrice quadrata qualsiasi $n \times n$, il determinante è definito come:

$$\det(A_k) = \sum_{i=1}^n (-1)^{i+k} a_{i,k} \det(A_{i,k})$$

dove $a_{i,k}$ è l'elemento di coordinate i,k e $A_{i,k}$ è il minore ottenuto sopprimendo la i -esima riga e la k -esima colonna

Estrazione del minore

Sotto-problema: **estrazione del minore (i, k)**

- Sopprimere dalla matrice in ingresso la riga i e la colonna k
- Il risultato va messo in una nuova matrice... anch'essa passata in ingresso
- Pseudo-codice:
 - In ingresso:
 - Matrice sorgente, matrice destinazione, dimensione matrice sorgente, riga e colonna da sopprimere
 - Per ogni riga r della matrice sorgente
 - La riga destinazione rm vale r se r è inferiore alla riga da sopprimere, $r-1$ altrimenti
 - Per ogni colonna c della matrice sorgente
 - La colonna destinazione cm vale c se c è inferiore alla colonna da sopprimere, $c-1$ altrimenti
 - Copiare l'elemento (r, c) della matrice sorgente nella posizione (rm, cm) nella matrice destinazione

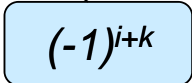
Estrazione del minore

```
void estraiMinore(Matrice minore, Matrice m, int dim,
                 int rigaElemento, int colonnaElemento)
{
    int riga, colonna;
    for (riga = 0; riga < dim; riga++)
    {
        int rigaMinore = (riga < rigaElemento) ? riga : riga - 1;
        for (colonna = 0; colonna < dim; colonna++)
        {
            if (riga != rigaElemento
                && colonna != colonnaElemento)
            {
                int colonnaMinore = colonna < colonnaElemento ?
                    colonna : colonna - 1;
                minore[rigaMinore][colonnaMinore] =
                    m[riga][colonna];
            }
        }
    }
}
```

11

Calcolo del determinante

- Pseudo-codice
 - In ingresso: matrice, dimensione
 - In uscita: valore del determinante
- Se la dimensione è 2, calcolare direttamente il determinante e restituire il valore
- Altrimenti, inizializzare la variabile **determinante** a 0 e per ogni colonna della prima riga (indice 0)
 - Creare una matrice d'appoggio
 - Estrarre il minore nella matrice d'appoggio
 - Calcolare il determinante del minore (ricorsione!)
 - Sommare a **determinante** il valore dell'elemento di posizione `[0, colonnaCorrente]` moltiplicato per il valore del determinante del minore eventualmente cambiato di segno nel caso in cui il resto della divisione per 2 della colonna corrente non sia nullo


$$(-1)^{i+k}$$

Calcolo del determinante

```
ELEMENT determinante(Matrice m, int dim)
{
    int riga = 0, colonna;
    if (dim == 2)
    {
        return m[0][0] * m[1][1] - m[0][1] * m[1][0];
    }
    else
    {
        ELEMENT det = 0;
        for (colonna = 0; colonna < dim; colonna++)
        {
            Matrice minore;
            ELEMENT detMin;
            estraiMinore(minore, m, dim, riga, colonna);
            detMin = determinante(minore, dim - 1);
            det += m[riga][colonna]*(colonna%2 == 0 ? detMin:-detMin);
        }
        return det;
    }
}
```

13

Sistemi lineari

Un “sistema lineare di m equazioni in n incognite” è un sistema di m equazioni nelle n incognite X_1, X_2, \dots, X_N .

$$a_{11}X_1 + a_{12}X_2 + \dots + a_{1N}X_N = b_1$$

$$a_{21}X_1 + a_{22}X_2 + \dots + a_{2N}X_N = b_2$$

...

$$a_{M1}X_1 + a_{M2}X_2 + \dots + a_{MN}X_N = b_M$$

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

14

Sistemi lineari

- Siamo interessati a sistemi in cui numero equazioni uguale al numero di incognite ($m = n$)
 - In questo caso, ***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, può essere che:
 - alcune equazioni sono *dipendenti* (combinazioni lineari) da altre (e si possono eliminare)

oppure

 - il sistema è indeterminato

Sistemi lineari

È possibile rappresentare il sistema come:
 $A * X = B$
dove

- A è la matrice dei coefficienti
- B il vettore dei termini noti
- X il vettore delle incognite (o soluzioni)

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

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

Sistemi lineari

- Rappresentazione come matrice completa

$$\begin{bmatrix} a_{11} & a_{12} & \cdot & a_{1N} & b_1 \\ a_{21} & a_{22} & \cdot & a_{2N} & b_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{N1} & a_{N2} & \cdot & a_{NN} & b_N \end{bmatrix}$$

- Il sistema ha una soluzione unica se **$\det(\mathbf{A})$** è **diverso da zero** (matrice *non singolare*)

Sistemi lineari

- Per risolvere questi sistemi si possono applicare due classi di metodi
 - **metodi *diretti***: basati su trasformazioni in sistemi di equazioni equivalenti
 - **metodi *indiretti* o *iterativi***: basati su successive approssimazioni
- Verrà studiato solo un esempio della prima classe di metodi

Sistemi lineari – 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 uno la dimensione del problema. Quando si arriva a determinare un valore, si procede a ritroso e si trovano tutti gli altri
- Il tutto è basato sul **principio di Equivalenza**:
 - Due sistemi di equazioni lineari nello stesso numero di incognite sono *equivalenti* se hanno le stesse soluzioni
 - Si può ottenere da un sistema un altro 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

Metodo di Gauss

- Avviene in due fasi
 - **Triangolarizzazione** della matrice dei coefficienti
 - **Eliminazione** all'indietro → Calcolo della soluzione

Metodo di Gauss - Triangolarizzazione

- Eliminazione della 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

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

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

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

Metodo di Gauss - Triangolarizzazione

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

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

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} + m_{ik}a_{kj}^{(k)} \quad (i = k + 1, \dots, n)$$

$$(j = k + 1, \dots, n + 1)$$

- Al termine si ottiene una **matrice triangolare superiore**:

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

Metodo di Gauss – Eliminazione

- Calcolo della soluzione

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

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

- Osservazione: il numero totale di calcoli da eseguire per portare a termine il procedimento è proporzionale a $n^3/3$

Metodo di Gauss

- Algoritmo di Triangolarizzazione

```
for (k = 1; k < n - 1; k++)
  for (i = k + 1; i < n; i++)
  {
    mik = aik(k) / akk(k)
    for (j = k; j <= n; j++)
      aij(k+1) = aij(k) - mik * akj(k)
  }
```

il vettore B dei termini noti e'
in a_{i,n+1}

Il sistema triangolare così ottenuto dopo $n-1$ trasformazioni può essere risolto facilmente con la procedura di eliminazione...

Metodo di Gauss

■ Algoritmo di Eliminazione

```
xn = yn/unn;  
for(i = n - 1; i >= 0; i--)  
{  
    for (j = i + 1; j <= n; j++)  
        xi = xi + uij*xj;  
    xi = (yi - xi)/uii;  
}
```

25

Metodo di Gauss – Interfaccia

```
//File Gauss.h
```

```
#include "Matrici.h"
```

```
//Triangolarizzazione
```

```
void triang(Matrice a, int rows,  
            int columns);
```

Rende triangolare la matrice quadrata "sinistra" di una matrice qualsiasi

```
//Eliminazione all'indietro
```

```
void elim_indietro(double *x, Matrice a,  
                   int rows);
```

Ha senso solo su matrici con N righe e N+1 colonne

Metodo di Gauss – Codifica!

```
#include "Gauss.h"

void triang(Matrice a, int rows, int columns)
{
    int masterEq, coeff, currentEq;
    for (masterEq = 0; masterEq < rows - 1; masterEq++)
    {
        for (currentEq = masterEq + 1; currentEq < rows;
            currentEq++)
        {
            double m =
                a[currentEq][masterEq]/a[masterEq][masterEq];
            for (coeff = masterEq; coeff < columns; coeff++)
                a[currentEq][coeff] = a[currentEq][coeff] -
                    m * a[masterEq][coeff];
        }
    }
}
```

Metodo di Gauss – Codifica!

```
void elim_indietro(ELEMENT *x, Matrice a, int rows)
{
    int currentEq, coeff;
    for (currentEq = rows - 1; currentEq >= 0; currentEq--)
    {
        for (x[currentEq] = 0, coeff = currentEq + 1;
            coeff < rows; coeff++)
            x[currentEq] -= a[currentEq][coeff] * x[coeff];
        x[currentEq] += a[currentEq][rows];
        x[currentEq] /= a[currentEq][currentEq];
    }
}
```