Programmazione Orientata agli Oggetti

Risoluzione di un Sistema di Equazioni lineari col metodo di Gauss

Libero Nigro

Una gerarchia di classi per la risoluzione di sistemi di equazioni lineari

- Si considerano sistemi di n equazioni lineari in n incognite. È noto che esistono più algoritmi risolutivi, es. mediante triangolazione di Gauss, regola di Cramer, uso della Matrice Inversa, etc
- Per consentire ad un programma di avvalersi di una qualunque tecnica risolutiva, avendo sempre a disposizione uno schema comune di riferimento, si introduce una classe astratta in cui il metodo fondamentale risolvi() è necessariamente astratto

```
package poo.sistema;
public abstract class Sistema{
    private int n;
    public int getN(){ return n; }
    public Sistema( double [][]a, double []y ){
        if( a.length != y.length )
            throw new RuntimeException("Sistema Inconsistente");
        for( int i=0; i<a.length; i++ )
            if( a[i].length != a.length )
                throw new RuntimeException("Sistema Inconsistente");
        this.n=a.length;
    }
    public abstract double[] risolvi();
}//Sistema
```

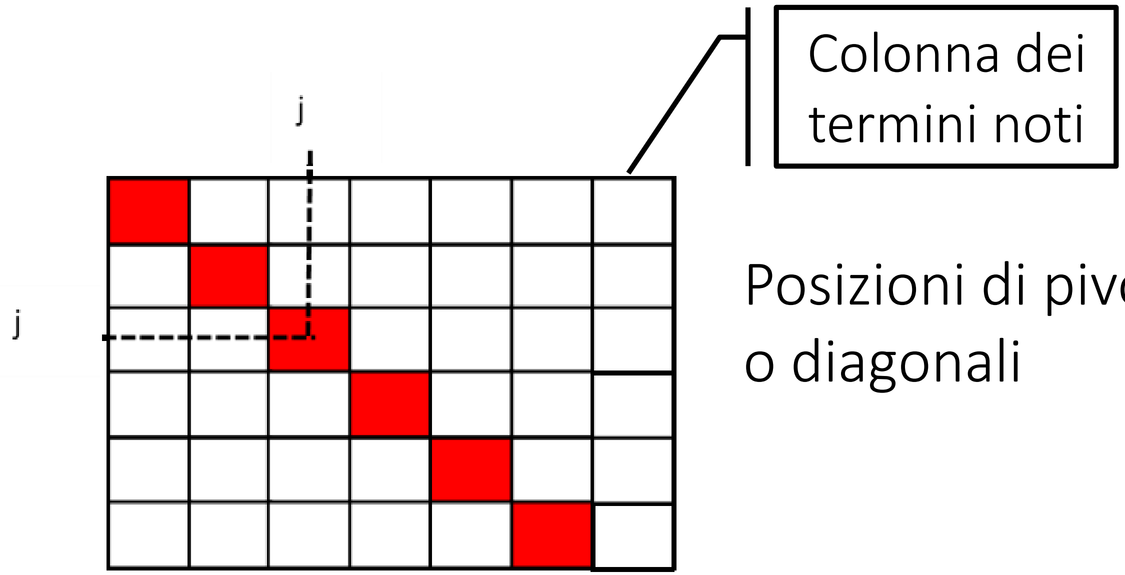
- Il costruttore della classe astratta Sistema si occupa di controllare che il sistema sia ben definito, ossia che la matrice **a** dei coefficienti sia effettivamente quadrata e che la dimensione comune di righe e colonne è uguale alla dimensione del vettore **y** dei termini noti.

- La classe Sistema memorizza solo la dimensione n del sistema, ma non gli array. Ogni particolare metodo realizzativo usa uno schema ad hoc per i dati (es. Gauss usa una matrice n*(n+1) etc.).

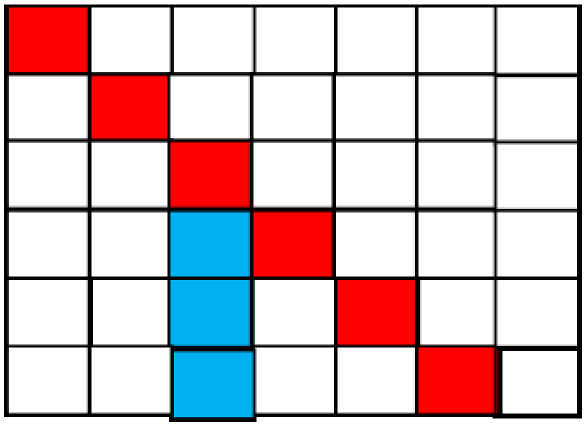
- Il metodo fondamentale risolvi() ritorna il vettore di n incognite x, se il sistema è determinato. Diversamente il metodo si conclude sollevando l'eccezione unchecked “SistemaSingolare”.

Il metodo di Gauss

- È noto dalla matematica che un sistema di n equazioni lineari A*X=Y si trasforma in uno equivalente se:
  - si scambiano due righe (due equazioni)
  - si moltiplica una riga per uno scalare
  - si sostituisce una riga r con il risultato della combinazione lineare tra r ed un'altra riga r' moltiplicata per uno scalare c: r=r±r'*c
- L'algoritmo di Gauss si compone di due fasi: (a) *triangolazione in avanti*, (b) *sostituzione a ritroso*.
- Durante la fase (a) si rende la matrice dei coefficienti A triangolare superiore, ossia si azzerano (attraverso combinazioni lineari) i coefficienti al di sotto della diagonale principale. È importante che le combinazioni lineari coinvolgano *anche* i termini noti
- Durante la fase (b), il sistema triangolare viene risolto partendo dall'ultima equazione: si calcola x[n-1], si sostituisce il valore di x[n-1] nella penultima equazione e si calcola quindi x[n-2] etc. sino a x[0]
- Per facilitare la scrittura del codice, è utile copiare il vettore dei termini noti Y come ultima colonna della matrice dei coefficienti che diviene rettangolare n*(n+1)



Posizioni di pivot o diagonali



Azzeramento elementi sotto una posizione di pivot

Schematizzazione

A*X=Y A matrice nxn dei coefficienti,
X vettore delle incognite, Y vettore dei termini noti
ciascuno di n elementi

$$A_{0,0}x_0 + A_{0,1}x_1 + \cdots + A_{0,n-1}x_{n-1} = y_0$$
$$A_{1,0}x_0 + A_{1,1}x_1 + \cdots + A_{1,n-1}x_{n-1} = y_1$$
$$A_{2,0}x_0 + A_{2,1}x_1 + \cdots + A_{2,n-1}x_{n-1} = y_2$$
$$\ldots$$
$$A_{n-1,0}x_0 + A_{n-1,1}x_1 + \cdots + A_{n-1,n-1}x_{n-1} = y_{n-1}$$

L'obiettivo della triangolazione del sistema, è quello di rendere uguali a 0 tutti coefficienti di A sotto la diagonale principale, realizzando combinazioni lineari tra le equazioni, o comunque operazioni che non alterano matematicamente il sistema

Sistema triangolare:

$$A'_{0,0}x_0 + A'_{0,1}x_1 + \cdots + A'_{0,n-1}x_{n-1} = y'_0$$

$$A'_{1,1}x_1 + \cdots + A'_{1,n-1}x_{n-1} = y'_1$$

$$A'_{2,2}x_2 \ldots + A_{2,n-1}x_{n-1} = y'_2$$

$$\cdots$$

$$A'_{n-1,n-1}x_{n-1} = y'_{n-1}$$

A partire dal sistema triangolare, si procede a ritroso (dall'ultima equazione verso la prima) per calcolare i valori del vettore delle incognite.

$$x_{n-1} = y'_{n-1}/A'_{n-1,n-1}$$

$$x_{n-2} = (y'_{n-2} - A'_{n-2,n-1}x_{n-1})/A'_{n-2,n-2}$$

… etc …

- Fissata una posizione di pivot o diagonale <j,j>, occorre assicurare che a[j][j] sia diverso da 0. Se non lo è si cerca (*pivoting*) una riga p, se esiste, tra j+1 ed n-1, tale che a[p][j] sia diverso da 0 (il calcolo numerico suggerisce che una scelta migliore, che riduce gli errori, consiste nel trovare la riga p tale che a[p][j] sia massimo in valore assoluto nella colonna j tra le righe da j+1 a n-1). Se una tale riga non esiste, allora il sistema è singolare (ammette infinite soluzioni).
- Dopo aver eseguito eventualmente il pivoting, si procede ad azzerare i coefficienti nella parte bassa della colonna j, cioè sulle righe da j+1 sino ad n-1. Detta i una tale riga, perché a[i][j] diventi 0 (nell'ipotesi che già non lo sia!) è sufficiente valutare il coeff=a[i][j]/a[j][j], quindi sottrarre (combinazione lineare) dalla riga i la riga j moltiplicata per coeff. Considerato che a sinistra della colonna j ed al di sotto della diagonale principale già risulta azzerata la matrice, la combinazione lineare può limitarsi ad investire le colonne dalla j-esima alla n-esima (ultima colonna della matrice a, contenente i termini noti)
- Per modularità, la triangolazione è affidata ad un metodo ausiliario triangolarizza()
- Anche il calcolo delle incognite è affidato ad un metodo: calcoloSoluzione() che ritorna il vettore delle incognite. Gli altri dettagli dovrebbero essere auto-esplicativi

La classe Gauss

```
package poo.sistema;
import poo.util.*;
public class Gauss extends Sistema{
    protected double [][]a;
    public Gauss( double [][]a, double []y ){
        super( a, y );
        //genera matrice n*(n+1) dei coeff+termini noti
        double [][] copia=new double[a.length][a.length+1];
        for( int i=0; i<a.length; i++ ){
            System.arraycopy(a[i],0,copia[i],0,a[0].length);
            copia[i][a.length]=y[i];
        }
        this.a=copia;
    }
```

```
protected void triangolazione(){
    //rende a triangolare superiore
    int n=this.getN();
    for( int j=0; j<n; j++ ){
        if( Mat.sufficientementeProssimi(a[j][j],0D) ){//pivoting
            int p=j+1;
            for( ; p<n; p++ )
                if( !Mat.sufficientementeProssimi(a[p][j],0D) ) break;
            if( p==n ) throw new SistemaSingolare();
            //scambia riga p con riga j
            double[] tmp=a[j]; a[j]=a[p]; a[p]=tmp;
        }
        //azzera elementi sulla colonna j, dalla riga (j+1)-esima all'ultima
        for( int i=j+1; i<n; i++ ){
            if( !Mat.sufficientementeProssimi(a[i][j],0D) ){
                double coeff=a[i][j]/a[j][j];
                //sottrai dalla riga i-esima la riga j-esima moltip per coeff
                for( int k=j; k<n+1; k++ ) a[i][k] = a[i][k]-a[j][k]*coeff;
            }
        }
    }//for esterno su j
}//triangolazione
```

```java
protected double[] calcoloSoluzione(){
    //a e' triangolare superiore
    int n=this.getN();
    double []x=new double[n];
    for( int i=n-1; i>=0; i-- ){
        //secondo membro inizializzato al valore del termine noto
        double sm=a[i][n];
        for( int j=i+1; j<n; j++ )
            sm = sm - a[i][j]*x[j];
        x[i]=sm/a[i][i];
    }
    return x;
}//calcoloSoluzione
```

```
  @Override
  public double[] risolvi() {
     triangolazione();
     return calcoloSoluzione();
  }//risolvi
  public String toString(){
     StringBuilder sb=new StringBuilder(500);
     for( int i=0; i<a.length; i++ ){
            for( int j=0; j<=a.length; j++ ){
                        sb.append( String.format("%5.2f", a[i][j]) ); //esempio
                        sb.append('  ');
            }
            sb.append("\n");
     }
     return sb.toString();
   }//toString
}//Gauss
```

```java
package poo.sistema;

public class SistemaSingolare extends RuntimeException{
    public SistemaSingolare(){}
    public SistemaSingolare( String msg ){ super(msg); }
}//SistemaSingolare
```

SistemaSingolare è definita come eccezione unchecked. Tuttavia, verificare a priori che il sistema è non singolare (o equivalentemente che il determinante della matrice dei coefficienti **a** è diverso da 0) non è triviale dal momento che calcolare il determinante è un lavoro paragonabile alla risoluzione del sistema. Tutto ciò spiega la struttura di main proposta di seguito

Un esempio di main

```
import java.util.*;
import poo.sistema.*;

public class SEL{
    public static void main( String []args ) throws SistemaSingolare{
        System.out.println(“Sistema risolto con GAUSS");
        Scanner sc=new Scanner( System.in );
        System.out.print("dimensione del sistema=");
        int n=sc.nextInt();
        double [][]a=new double[n][n];
        double []y=new double[n];
        double []x=null;
```

```
//lettura matrice
System.out.println
     ("Fornisci gli "+n+"x"+n+" elementi della matrice a per righe");
for( int i=0; i<n; i++ )
          for( int j=0; j<n; j++ ) {
                    System.out.print("a["+i+","+j+"]=");
                    a[i][j]=sc.nextDouble();
          }
System.out.println();
System.out.println("Fornisci i(gli) "+n+" termini noti");
for( int i=0; i<n; i++ ){
          System.out.print("y["+i+"]=");
          y[i]=sc.nextDouble();
}
```

```
            Sistema s=new Gauss(a,y);
            System.out.println( s );
     try{
          x=s.risolvi();
     }catch( SistemaSingolare e ){
          System.out.println("Sistema Singolare!");
          System.exit(-1);
     }
     System.out.println( s ); //visualizza sistema triangolare
            //scrivi risultati
            System.out.println("Vettore delle incognite");
            for( int i=0; i<n; i++ ) System.out.printf("x["+i+"]=%1.2f%n",x[i]);
     }//main
}//SEL
```

Sul metodo di Cramer

Supponendo di disporre del metodo determinante di una matrice quadrata, la risoluzione del sistema A*X=Y si può ottenere come segue. Sia:

$double\ dA = \det(A)$; //calcolato una sola volta

Per ogni colonna j della matrice A, sia A^j la matrice che si ottiene da A sostituendo alla colonna j il vettore y dei termini noti. Allora:

$$x[j] = \frac{\det(A^j)}{dA}$$

Considerato che dopo il calcolo di x[j], occorre ripristinare la colonna j-esima di A, si può scambiare il vettore y con la colonna j-esima di A prima e dopo il calcolo di x[j]. Quindi si procede con j+1 etc.