

Metodi Computazionali della Fisica



Matrici e sistemi di equazioni lineari

Problema da risolvere

I **sistemi di equazioni lineari** si trovano spesso in Fisica poichè la “**linearizzazione**” è un’assunzione o approssimazione molto comune nella descrizione dei processi fisici.

Supponiamo di studiare un sistema fisico, descrivibile con N equazioni **lineari**, accoppiate, in N **incognite**:

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1N}x_N = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2N}x_N = b_2 \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ a_{N1}x_1 + a_{N2}x_2 + \dots + a_{NN}x_N = b_N \end{array} \right.$$

Problema da risolvere

dove $\{a_{ij}\}$ e $\{b_i\}$ sono parametri **noti** e $\{x_i\}$ sono le **incognite**.
E' conveniente scrivere le nostre equazioni lineari nella **forma matriciale**:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ \cdot \\ b_N \end{pmatrix}$$

Problema da risolvere

o, in termini compatti,

$$AX = B \quad (1)$$

essendo A una **matrice (nota)** ($N \times N$), B un **vettore (noto)** di lunghezza N e X un **vettore (incognito)** di lunghezza N .

N.B. qui ci limitiamo a descrivere metodi “**diretti**”, appropriati per trattare matrici “**dense**” (in cui la maggior parte degli elementi sono $\neq 0$), di dimensioni relativamente **piccole** ($N \leq 100$); spesso, nelle applicazioni pratiche con matrici molto **grandi**, si sfrutta il fatto che quasi sempre sono “**sparse**” (molti degli elementi sono 0) e si utilizzano opportune routines di **librerie** matematiche.

Chiaramente la (1) sarebbe **risolta** conoscendo la **matrice inversa** A^{-1} , infatti:

$$X = A^{-1} B$$

Librerie matematiche

Alcune importanti librerie matematiche per uso **scientifico** sono le seguenti (vedi, ad esempio, <http://www.gnu.org/software/gsl/>) :

- **NETLIB** a WWW metalibrary of free math libraries
- **IMSL** International Mathematical and Statistical libraries
- **ESSL** Engineering and Scientific Subroutine Library (IBM)
- **DXML** Advanced Mathematical Library (DEC)
- **NAG** Numerical Algorithms Group (UK Labs)
- **SLATEC** Comprehensive Mathematical and Statistical package
- **LAPACK** Linear Algebra Package
- **CERN** European Center for Nuclear Research
- **BLAS** Basic Linear Algebra Subprograms

Problema da risolvere

D'altra parte sappiamo che : $A^{-1} = \frac{adj(A)}{\det(A)}$

dove la matrice “**aggiunta**” $adj(A) \equiv C^T$

C^T essendo la “**trasposta**” di C (scambio di righe con colonne) e con:

$C_{ij} = (-1)^{i+j} (cofA)_{ij}$ dove il “**cofattore**” $(cofA)_{ij}$ è dato

calcolando il **determinante** della matrice $(N-1) \times (N-1)$, ottenuta dalla A eliminando l' i -esima riga e la j -esima colonna;

ad **esempio**, se:

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & 2 & 2 \\ 2 & 4 & 1 \end{pmatrix}$$

Problema da risolvere

allora :

$$\begin{aligned}(cofA)_{11} &= -6 & (cofA)_{12} &= 0 & (cofA)_{13} &= 12 \\(cofA)_{21} &= -2 & (cofA)_{22} &= -1 & (cofA)_{23} &= 0 \\(cofA)_{31} &= 2 & (cofA)_{32} &= -2 & (cofA)_{33} &= -6\end{aligned}$$

e $\det(A) = 1 \cdot (-6) - 2 \cdot 0 + 1 \cdot 12 = 6$, allora:

$$C = \begin{pmatrix} -6 & 0 & 12 \\ 2 & -1 & 0 \\ 2 & 2 & -6 \end{pmatrix} \Rightarrow C^T = \begin{pmatrix} -6 & 2 & 2 \\ 0 & -1 & 2 \\ 12 & 0 & -6 \end{pmatrix} \Rightarrow$$

$$A^{-1} = \frac{C^T}{\det(A)} = \begin{pmatrix} -1 & 1/3 & 1/3 \\ 0 & -1/6 & 1/3 \\ 2 & 0 & -1 \end{pmatrix}$$

Problema da risolvere

Un tale calcolo richiede di valutare N^2 **determinanti** di matrici $(N-1) \times (N-1)$; se, per la generica matrice A $N \times N$, il determinante è calcolato usando la formula standard:

$$\det A = \sum_P (-1)^P A_{1P_1} A_{2P_2} \dots A_{NP_N} \quad (2)$$

dove P indica una delle $N!$ permutazioni delle N colonne, allora sono necessarie $N!$ operazioni per calcolare (2), quindi, per $N=20$, servono $2 \cdot 10^{18}$ moltiplicazioni, allora se un computer fa 10^8 moltiplicazioni al secondo sarebbero necessari circa **600 anni** !

Non è il metodo adatto a meno che N non sia molto *piccolo*.

Uno dei più semplici metodi **pratici** per valutare A^{-1} è costituito dal “**Gauss-Jordan elimination method**”.

“Gauss-Jordan elimination”

L'idea fondamentale è quella di considerare una classe di **operazioni elementari** sulle **righe** della matrice A ; queste includono:

- **moltiplicare** una particolare riga per una **costante**;
- **scambiare** due righe;
- **aggiungere** un **multiplo** di una riga ad un'altra.

ognuna di queste 3 operazioni può essere realizzata moltiplicando “**da sinistra**” la matrice A per una matrice T ; ad esempio, se $N=3$, allora le matrici:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \quad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}$$

“Gauss-Jordan elimination”

realizzano, rispettivamente, le seguenti operazioni:

- **moltiplicano** la terza riga per 2;
- **scambiano** la prima con la seconda riga;
- **sottraggono** $1/2$ della prima riga dalla terza.

La **strategia** del metodo di “**eliminazione Gauss-Jordan**” è quella di trovare una **sequenza di operazioni** $T = \dots T_3 T_2 T_1$, tale che, applicata ad A , la “**riduce**” alla matrice **unitaria**:

$$TA = (\dots T_3 T_2 T_1)A = I$$

allora, evidentemente $T = A^{-1}$ è la matrice inversa **richiesta**.

“Gauss-Jordan elimination”

Consideriamo ancora la matrice A dell'esempio precedente e la matrice unitaria I (con $N=3$):

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & 2 & 2 \\ 2 & 4 & 1 \end{pmatrix}, \quad I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

e applichiamo la seguente procedura in **3 passi**:

“Gauss-Jordan elimination”

I) cerchiamo di **azzerare** tutti gli elementi, **tranne il primo**, nella **prima colonna** di A:

- **sottraiamo** 4 volte la prima riga dalla seconda:

$$TA = \begin{pmatrix} 1 & 2 & 1 \\ 0 & -6 & -2 \\ 2 & 4 & 1 \end{pmatrix}, \quad TI = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **sottraiamo** 2 volte la prima riga dalla terza:

$$TA = \begin{pmatrix} 1 & 2 & 1 \\ 0 & -6 & -2 \\ 0 & 0 & -1 \end{pmatrix}, \quad TI = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

“Gauss-Jordan elimination”

II) cerchiamo di **azzerare** tutti gli elementi, **tranne il secondo**, nella **seconda colonna** di A :

- **sommiamo** $1/3$ della seconda riga alla prima:

$$TA = \begin{pmatrix} 1 & 0 & 1/3 \\ 0 & -6 & -2 \\ 0 & 0 & -1 \end{pmatrix}, \quad TI = \begin{pmatrix} -1/3 & 1/3 & 0 \\ -4 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

- **moltiplichiamo** la seconda riga per $(-1/6)$:

$$TA = \begin{pmatrix} 1 & 0 & 1/3 \\ 0 & 1 & 1/3 \\ 0 & 0 & -1 \end{pmatrix}, \quad TI = \begin{pmatrix} -1/3 & 1/3 & 0 \\ 2/3 & -1/6 & 0 \\ -2 & 0 & 1 \end{pmatrix}$$

“Gauss-Jordan elimination”

III) cerchiamo di **azzerare** tutti gli elementi, **tranne il terzo**, nella **terza colonna** di A :

- **sommiamo** $1/3$ della terza riga alla prima ed alla seconda:

$$TA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \quad TI = \begin{pmatrix} -1 & 1/3 & 1/3 \\ 0 & -1/6 & 1/3 \\ -2 & 0 & 1 \end{pmatrix}$$

- **moltiplichiamo** la terza riga per (-1) :

$$TA = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad TI = \begin{pmatrix} -1 & 1/3 & 1/3 \\ 0 & -1/6 & 1/3 \\ 2 & 0 & -1 \end{pmatrix}$$

“Gauss-Jordan elimination”

allora $TI=T$ è la matrice **inversa** richiesta.

Questo algoritmo può essere facilmente **generalizzato** ad una matrice ($N \times N$) e si può dimostrare che, per N grande, richiede dell'ordine di N^3 operazioni (moltiplicazioni ed addizioni), quindi, a meno che N non sia troppo grande, è un algoritmo **utilizzabile** in pratica.

Osservazioni sull'uso pratico:

- in qualche punto della procedura può darsi che il termine **diagonale** nella colonna su cui si sta lavorando diventi 0 , allora, per evitare che la matrice diventi **singolare** (determinante= 0), è necessario lo **scambio** di 2 righe o colonne per far sì che questo elemento “**pivot**” sia $\neq 0$;

“Gauss-Jordan elimination”

- possono anche sorgere problemi associati all'**arrotondamento** numerico se ci sono elementi della matrice che **differiscono molto** in grandezza; allora è spesso utile “**riscalare**” le file o le colonne in modo che tutti gli elementi siano **dello stesso ordine** di grandezza (“**equilibratura**”);
- vari casi **speciali** (ad esempio quando A è **simmetrica**) possono portare ad una **riduzione** dello sforzo numerico;
- se interessa calcolare **solo il determinante** di A è sufficiente effettuare **solo** le trasformazioni delle righe (le quali hanno un effetto **semplice e calcolabile** sul determinante*) che riducono TA ad una forma “**lower diagonal**” o “**upper diagonal**” (tutti gli elementi sono **nulli sopra** o **sotto** la diagonale, rispettivamente), come all'inizio della fase **II**) descritta precedentemente;

“Gauss-Jordan elimination”

allora il determinante si calcola banalmente facendo il **prodotto** degli elementi diagonali ($\det(A)=6$ nel nostro caso):

$$TA = \begin{pmatrix} 1 & 0 & 1/3 \\ 0 & -6 & -2 \\ 0 & 0 & -1 \end{pmatrix}$$

*:dalla (2) si può facilmente dimostrare che:

- **scambiare** due righe di una matrice **cambia il segno** del determinante;
- **sommare** un multiplo di una riga ad un'altra lascia il determinante **inalterato**;
- **moltiplicare per una costante** una riga **moltiplica** il determinante per la **stessa** costante.

Programma in C++ per risolvere $AX=B$

Gauss.cpp

```
#include <iostream>
#include <cmath>
#define N 100
int main() {
    using namespace std;
    /* soluzione dei sistemi di equazioni con il metodo di eliminazione delle incognite di Gauss */
    double A[N][N], b[N], x[N];
    double C, S;
    int n, i, j, k;
    /* inserisce i dati iniziali */
    cout << " Inserisci il numero di equazioni (<100): " << endl;
    cin >> n ;
    cout << " Inserisci ora i coefficienti del sistema
    e i termini noti" << endl;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {

            cout << "A["<<i<<","<<j<<"] = " << endl;
            cin >> A[i][j];
        }
        cout << "b["<<i<<"] = " << endl;
        cin >> b[i];
    }
```

Programma in C++ per risolvere $AX=B$

```
/* triangolarizza la matrice */
for (i = 0; i < n; i++) {
/* divide l'i-esima equazione per l'elemento diagonale C */
    C = A[i][i];
    for (j = i; j < n; j++) {
        A[i][j] /= C;
    }
    b[i] /= C;
/* sottrae l'equazione normalizzata dalle altre */
    for (k = i + 1; k < n; k++) {
        C = A[k][i];
        for (j = i; j < n; j++) {
            A[k][j] -= A[i][j] * C;
        }
        b[k] -= C * b[i];
    }
/* risolve */
for (k = n - 1; k >= 0; k--) {
    S = 0.;
    for (i = k + 1; i < n; i++) {
        S += A[k][i] * x[i];
    }
    x[k] = b[k] - S;
}
```

Programma in C++ per risolvere $AX=B$



```
/* stampa il risultato */  
for (i = 0; i < n; i++) {  
    cout << "x[" << i << "] = " << x[i] << endl;  
}  
}
```

Applicazioni:

- col programma precedente si può **invertire** la matrice dell'esempio :

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 4 & 2 & 2 \\ 2 & 4 & 1 \end{pmatrix}$$

- si può risolvere $AX=B$ considerando una matrice relativamente **grande**, ad esempio la matrice di **Hilbert** (100×100), con:

$$a_{ij} = \frac{1}{i+j-1}, \quad b_i = \frac{1}{i}$$

la cui **soluzione** è :

$$x_i = \delta_{i1}$$

Applicazioni:

matrice di **Hilbert**:

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & . & . & 1/100 \\ 1/2 & 1/3 & 1/4 & 1/5 & . & . & 1/101 \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ 1/100 & 1/101 & . & . & . & . & 1/199 \end{pmatrix}$$

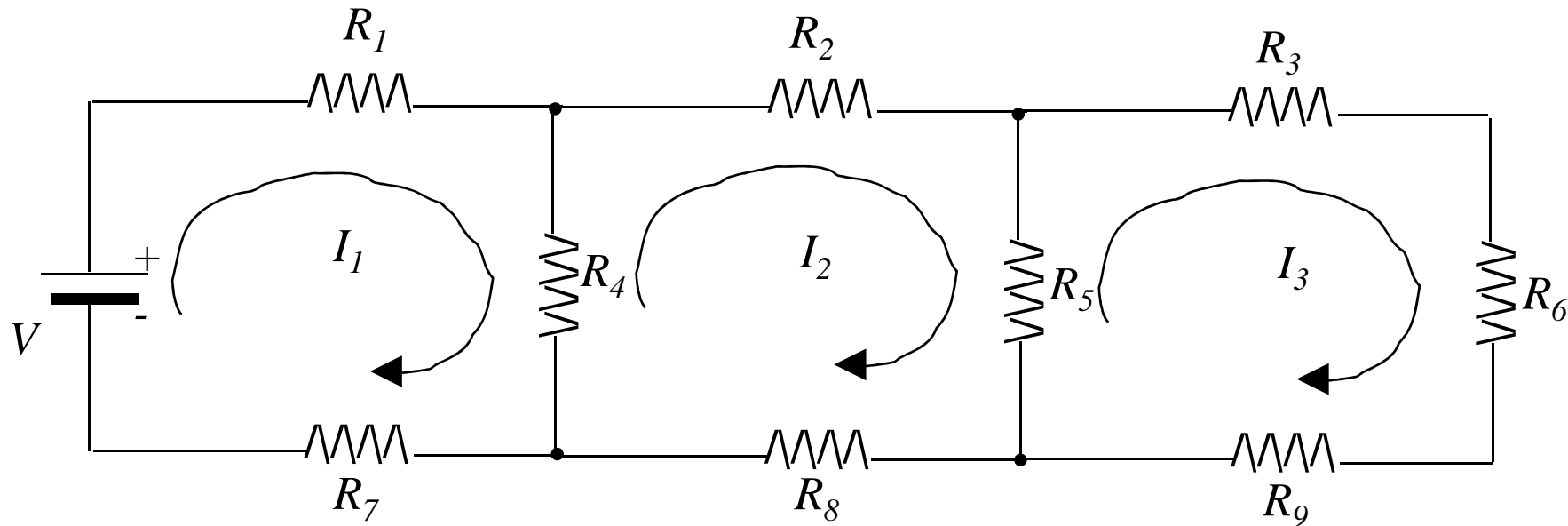
con :

$$B = \begin{pmatrix} 1 \\ 1/2 \\ 1/3 \\ . \\ . \\ 1/100 \end{pmatrix} \Rightarrow X = \begin{pmatrix} 1 \\ 0 \\ 0 \\ . \\ . \\ 0 \end{pmatrix}$$

Esempio fisico: circuiti elettrici e legge di Kirchhoff

La (**seconda**) legge di **Kirchoff** (o legge “**delle maglie**”) stabilisce che la somma algebrica delle differenze di potenziale rilevate su un circuito **chiuso**, in un giro completo è **nulla**.

Se consideriamo il circuito seguente, con una **batteria** di d.d.p. V e 9 **resistenze**, R_1, R_2, \dots, R_9 :



Esempio fisico: circuiti elettrici e legge di Kirchhoff

allora otteniamo (considerando le **3 maglie** evidenziate in figura e la **prima** legge di Kirchhoff per ridurre il numero di correnti incognite) il seguente sistema di equazioni lineari:

$$\begin{cases} V - I_1 R_1 - (I_1 - I_2) R_4 - I_1 R_7 = 0 \\ (I_1 - I_2) R_4 - I_2 R_2 - (I_2 - I_3) R_5 - I_2 R_8 = 0 \\ (I_2 - I_3) R_5 - I_3 R_3 - I_3 R_6 - I_3 R_9 = 0 \end{cases}$$

che si può riscrivere come:

$$\begin{cases} (R_1 + R_4 + R_7) I_1 - R_4 I_2 + 0 \cdot I_3 = V \\ -R_4 I_1 + (R_2 + R_4 + R_5 + R_8) I_2 - R_5 I_3 = 0 \\ 0 \cdot I_1 - R_5 I_2 + (R_3 + R_5 + R_6 + R_9) I_3 = 0 \end{cases}$$

Esempio fisico: circuiti elettrici e legge di Kirchhoff

Allora, trasformando in forma **matriciale**, il vettore $B=(V,0,0)$, il vettore $X=(I_1, I_2, I_3)$ e la matrice A è:

$$A = \begin{pmatrix} (R_1 + R_4 + R_7) & -R_4 & 0 \\ -R_4 & (R_2 + R_4 + R_5 + R_8) & -R_5 \\ 0 & -R_5 & (R_3 + R_5 + R_6 + R_9) \end{pmatrix}$$

e se supponiamo che $V=36 \text{ volt}$,

$$R_1=R_2=R_3=R_4=R_7=R_8=R_9= 10 \, \Omega$$

$$R_5=R_6= 20 \, \Omega$$

$$\Rightarrow A = \begin{pmatrix} 30 & -10 & 0 \\ -10 & 50 & -20 \\ 0 & -20 & 60 \end{pmatrix} \rightarrow \text{matrice } \mathbf{simmetrica} \quad (a_{ij}=a_{ji})$$

Esempio fisico: circuiti elettrici e legge di Kirchhoff

Allora, applicando il **programma** precedente alla matrice:

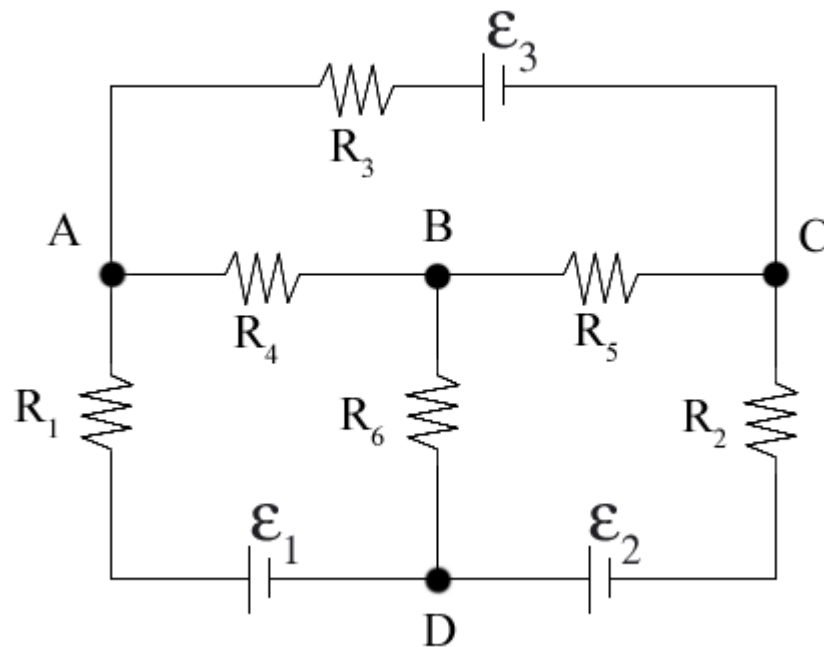
$$A = \begin{pmatrix} 30 & -10 & 0 \\ -10 & 50 & -20 \\ 0 & -20 & 60 \end{pmatrix}$$

si trova facilmente che la **soluzione** (in Ampere) è:

$$\begin{cases} I_1 = 1.3 \\ I_2 = 0.3 \\ I_3 = 0.1 \end{cases}$$

Esercizio 4a: circuiti elettrici e legge di Kirchhoff

Usare il **programma precedente** per **calcolare** le correnti $i_1, i_2, i_3, i_4, i_5, i_6$ passanti attraverso le **6** resistenze ($R_1=R_2=R_3=R_4=R_5=R_6=1\Omega$) del seguente circuito elettrico (relativamente complicato) , con le d.d.p. $\varepsilon_1=\varepsilon_2=1.5\text{ V}$, $\varepsilon_3=3\text{ V}$; il problema si può risolvere anche “*a mano*” (analiticamente) ma la procedura è piuttosto **noiosa** (... e la possibilità di commettere **errori** elevata !) :



Decomposizione LU

è un metodo **più efficiente** per risolvere il sistema di equazioni lineari $AX=B$.

Supponiamo di riuscire a “**fattorizzare**” la matrice A nel prodotto di due matrici **triangolari**:

$$A = L \cdot U, \text{ con:}$$

$$L = \begin{pmatrix} 1 & 0 & \cdot & \cdot & 0 \\ & 1 & 0 & \cdot & 0 \\ & & \cdot & \cdot & \\ & & & \cdot & \\ & & & & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & u_{12} & \cdot & \cdot & u_{1N} \\ 0 & u_{22} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & u_{NN} \end{pmatrix}$$

- L (**lower**) ha elementi $\neq 0$ sulla e **sotto** la diagonale;
- U (**upper**) ha elementi $\neq 0$ sulla e **sopra** la diagonale.

Decomposizione LU

allora possiamo scrivere $AX=B$ come:

$$L \cdot (U \cdot X) = B$$

che è equivalente a risolvere la **coppia** del (**più semplice**) sistema di equazioni:

$$\begin{cases} LY = B \\ UX = Y \end{cases} \quad (1)$$

il punto **chiave** è che le equazioni (1) sono **facili** da risolvere poichè le matrici L ed U sono **triangolari**.

Decomposizione LU

infatti, se consideriamo $LY=B$, cioè:

$$\begin{pmatrix} l_{11} & 0 & . & . & 0 \\ l_{21} & l_{22} & 0 & . & 0 \\ . & . & . & . & . \\ . & . & . & . & . \\ l_{N1} & . & . & . & l_{NN} \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ y_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ . \\ . \\ b_N \end{pmatrix}$$

allora ovviamente y_1 si trova subito: $l_{11}y_1 = b_1 \Rightarrow y_1 = b_1 / l_{11}$

ma allora si trova anche y_2 :

$$l_{21}y_1 + l_{22}y_2 = b_2 \Rightarrow y_2 = \frac{1}{l_{22}}(b_2 - l_{21}y_1)$$

e, in generale, con la “**forward** substitution” (sostituzione “in avanti”):

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right) \quad i = 1, \dots, N$$

Decomposizione LU

analogamente con $UX=Y$:

$$\begin{pmatrix} u_{11} & u_{12} & \cdot & \cdot & u_{1N} \\ 0 & u_{22} & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & u_{NN} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_N \end{pmatrix}$$

x_N si trova subito: $u_{NN}x_N = y_N \Rightarrow x_N = y_N / u_{NN}$

ma allora si trova anche x_{N-1} :

$$u_{N-1,N-1}x_{N-1} + u_{N-1,N}x_N = y_{N-1} \Rightarrow x_{N-1} = \frac{1}{u_{N-1,N-1}}(y_{N-1} - u_{N-1,N}x_N)$$

e, in generale, con la “**backward substitution**” (sostituzione “all’indietro”):

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{j=i+1}^N u_{ij}x_j \right) \quad i = N, \dots, 1$$

Decomposizione LU

resta allora il **problema** fondamentale di determinare le matrici L e U .
Se consideriamo esplicitamente $L \cdot U = A$:

$$\sum_{k=1}^N l_{ik} u_{kj} = a_{ij} \quad i = 1, \dots, N \quad j = 1, \dots, N \quad (2)$$

le (2) sono N^2 equazioni per le $(N^2 + \textcolor{red}{N})$ incognite $\{l_{ij}, u_{ij}\}$ (**N.B.** non sono $2N^2$, infatti è come se avessi un'unica matrice $(N \times N)$, più la diagonale che va presa **2 volte**).

Poichè ho **più** incognite che equazioni, l'idea è quella di assegnare un valore **arbitrario** ad $\textcolor{red}{N}$ incognite e poi calcolare le altre;
in particolare scegliamo (si dimostra che **funziona** !) :

$$l_{ii} = 1 \quad i = 1, \dots, N$$

Decomposizione LU

data la struttura **triangolare** di L e U è facile anche verificare che la

(2) si può scrivere come:
$$\sum_{k=1}^i l_{ik} u_{kj} = a_{ij} \quad i \leq j$$

$$\sum_{k=1}^j l_{ik} u_{kj} = a_{ij} \quad i > j$$

allora si può applicare la **procedura di Crout**, per $j=1,2,\dots,N$:

$$u_{1j} = a_{1j}$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \quad i = 2, \dots, j$$

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) \quad i = j+1, \dots, N$$

Decomposizione LU

e come **sottoprodotto** otteniamo anche il **determinante** di A (è il prodotto degli elementi diagonali di U):

$$\det(A) = \prod_{i=1}^N u_{ii}$$

Illustriamo la procedura di Crout con un **esempio** :

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

prima poniamo : $l_{11}=l_{22}=1$ e poi:

Decomposizione LU

$$j = 1, \quad i = 1 \quad u_{11} = a_{11} = 1$$

$$j = 1, \quad i = 2 \quad l_{21} = \frac{a_{21}}{u_{11}} = 3$$

$$j = 2, \quad i = 1 \quad u_{12} = a_{12} = 2$$

$$j = 2, \quad i = 2 \quad u_{22} = a_{22} - l_{21}u_{12} = -2$$

ovviamente con : $u_{21}=l_{12}=0$, allora:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & -2 \end{pmatrix} = L \cdot U$$

Decomposizione LU

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 0 & -2 \end{pmatrix} = L \cdot U$$

$$\text{e : } \det(A) = \prod_{i=1}^2 u_{ii} = -2$$

per trovare la matrice **inversa** basta risolvere :

$$AX_I = E_I \quad (I = 1, \dots, N)$$

$$E_1 = \begin{pmatrix} 1 \\ 0 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}, E_2 = \begin{pmatrix} 0 \\ 1 \\ \cdot \\ \cdot \\ 0 \end{pmatrix}, \dots, E_N = \begin{pmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ 1 \end{pmatrix}$$

allora i vettori X_I sono le
colonne di A^{-1} ;
nel nostro esempio:

Decomposizione LU

$$AX_1 = L(UX_1) = LY_1 = E_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$LY_1 = E_1$ si risolve con le formule precedenti:

$$y_1^1 = \frac{e_1^1}{l_{11}} = 1, \quad y_2^1 = \frac{1}{l_{22}} (e_2^1 - l_{21}y_1^1) = -3 \quad \Rightarrow \quad Y_1 = \begin{pmatrix} +1 \\ -3 \end{pmatrix}$$

e analogamente $UX_1 = Y_1$:

$$x_2^1 = \frac{y_2^1}{u_{22}} = \frac{3}{2}, \quad x_1^1 = \frac{1}{u_{11}} (y_1^1 - u_{12}x_2^1) = -2 \quad \Rightarrow \quad X_1 = \begin{pmatrix} -2 \\ 3/2 \end{pmatrix}$$

 **prima** colonna di A^{-1}

Decomposizione LU

analogamente : $AX_2 = L(UX_2) = LY_2 = E_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$LY_2=E_2$ si risolve con le formule precedenti:

$$y_1^2 = \frac{e_1^2}{l_{11}} = 0, \quad y_2^2 = \frac{1}{l_{22}} (e_2^2 - l_{21}y_1^2) = 1 \quad \Rightarrow \quad Y_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

e analogamente $UX_2=Y_2$:

$$x_2^2 = \frac{y_2^2}{u_{22}} = -\frac{1}{2}, \quad x_1^2 = \frac{1}{u_{11}} (y_1^2 - u_{12}x_2^2) = 1 \quad \Rightarrow \quad X_2 = \begin{pmatrix} +1 \\ -1/2 \end{pmatrix}$$

 seconda colonna di A^{-1}

Decomposizione LU

allora :

$$A^{-1} = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix}$$

è proprio l'**inversa**, infatti:

$$A^{-1} \cdot A = \begin{pmatrix} -2 & 1 \\ 3/2 & -1/2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

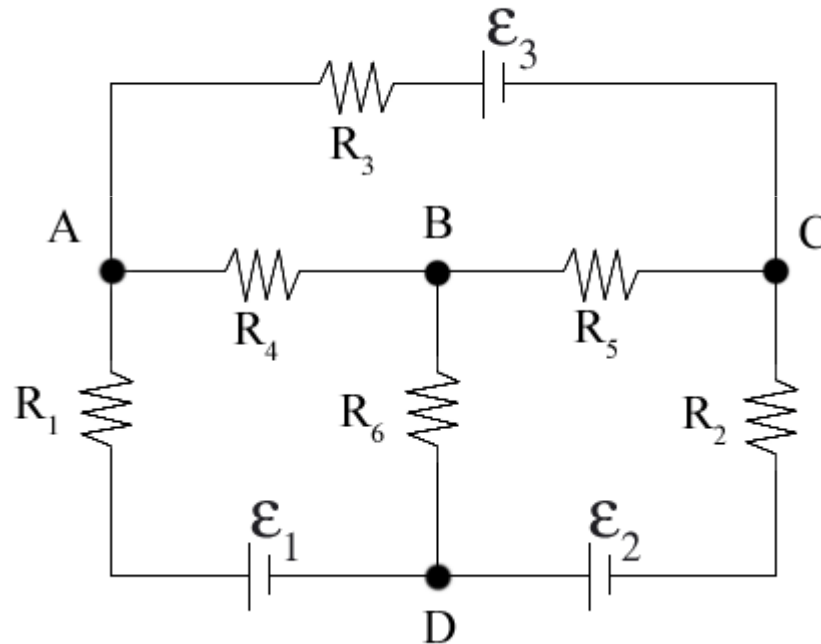
N.B. : con il metodo *LU*, che scala comunque con N^3 ,

poichè il vettore B non è “toccato”, possiamo effettuare una volta per tutte la **fattorizzazione (decomposizione)** $A=LU$ di una data matrice A , che rappresenta il **maggior costo** computazionale, e poi usarla per **diversi vettori** B ;

in pratica è la **scelta migliore** per risolvere il set di equazioni $AX=B$.

Esercizio 4a: circuito elettrico

- **modificare** il programma in modo da ripetere il calcolo delle correnti del circuito precedente col metodo di **decomposizione LU**:



- **calcolare** anche il **determinante** di A e la matrice inversa A^{-1} .

Matrici

I metodi numerici considerati finora, compreso il metodo di decomposizione **LU**, scalano comunque come N^3 ; fortunatamente, nelle applicazioni **reali**, spesso si può sfruttare **proprietà particolari** delle matrici, in modo da avere uno “*scaling*” più **favorevole**.

Ad esempio, consideriamo alcune equazioni importanti per la Fisica:

- equazione di Schrödinger indipendente dal tempo:

$$\left(-\frac{\hbar^2}{2m} \nabla^2 + V(\vec{r}) \right) \psi_E(\vec{r}) = E \psi_E(\vec{r})$$

autovalore
autostato

in **Meccanica Quantistica** descrive una **particella singola** (esempio: stati elettronici dell'atomo di Idrogeno);

Matrici

- equazione di Poisson:

$$\nabla^2 V(\vec{r}) = -\frac{\rho(\vec{r})}{\varepsilon}$$

carica elettrica stazionaria

potenziale elettrico

in **Elettrostatica** mette in relazione il potenziale elettrico con la distribuzione di carica elettrica;

- equazione di Laplace:

$$\nabla^2 V(\vec{r}) = 0$$

è una delle equazioni più **importanti** in Fisica Matematica e compare in molti problemi anche al di fuori della teoria del campo elettromagnetico, come nel moto dei *fluidi* e nella teoria dell'*elasticità*.

Equazioni ellittiche

Le equazioni precedenti appartengono alle equazioni differenziali “*ellittiche*” :

$$\nabla^2 f(\vec{r}) = g(\vec{r})$$
, dove :

- $f(\vec{r}) = -\frac{\hbar^2}{2m}\psi_E(\vec{r})$, $g(\vec{r}) = (E - V(\vec{r}))\psi_E(\vec{r})$ per l’eq. di *Schrödinger*;
- $f(\vec{r}) = V(\vec{r})$, $g(\vec{r}) = -\frac{\rho(\vec{r})}{\varepsilon}$ per l’eq. di *Poisson*;
- $f(\vec{r}) = V(\vec{r})$, $g(\vec{r}) = 0$ per l’eq. di *Laplace*.

N.B. in generale queste equazioni sono soggette a condizioni al *contorno* all’estremo superiore dell’intervallo di validità e **non** possono essere risolte con i metodi illustrati per le ODE.

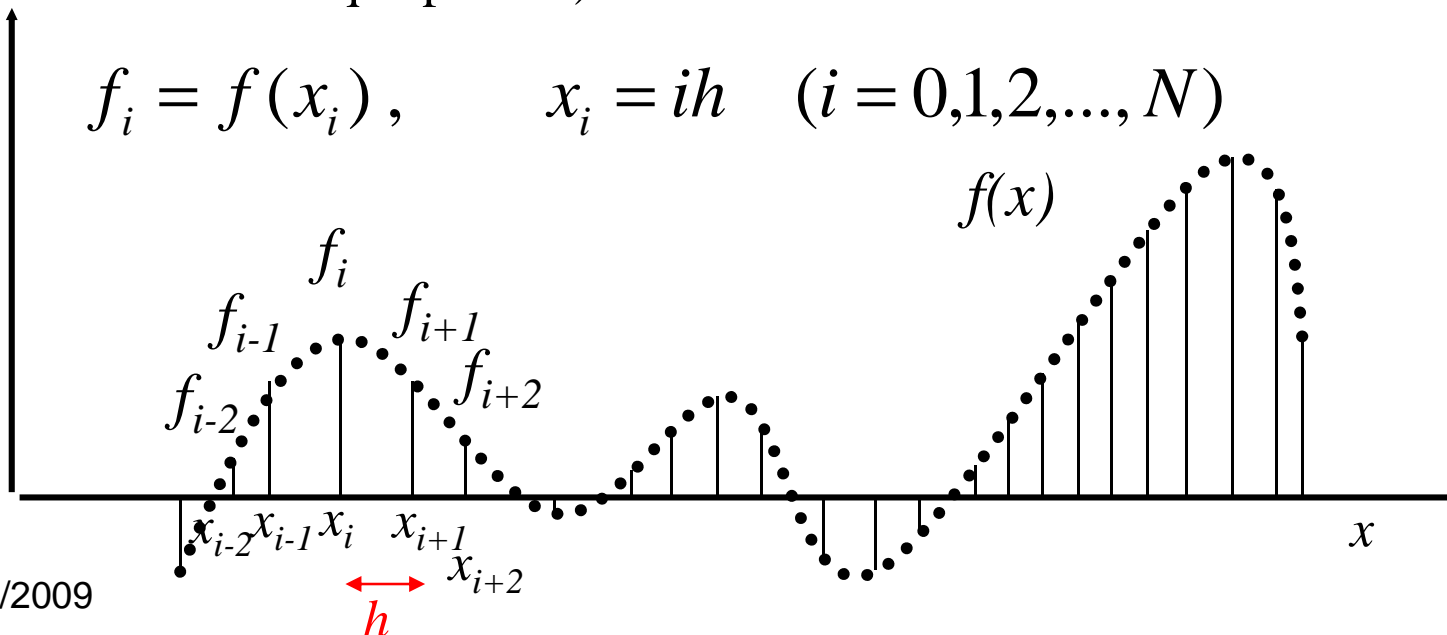
Equazione ellittiche

Il *Laplaciano* è definito come : $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$, $\vec{r} = (x, y, z)$

Allora, ad esempio, in **1D** bisogna risolvere equazioni del tipo:

$$\frac{d^2 f(x)}{dx^2} = g(x)$$

Per usare un approccio numerico, **discretizziamo** le funzioni su di una griglia (non necessariamente equispaziata):



Equazione di Poisson in 1D

Ricordando l'espressione già ricavata per stimare numericamente la derivata seconda:

$$\left. \frac{d^2 f(x)}{dx^2} \right|_{x=x_i} = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} + \mathcal{O}(h^2)$$

l'equazione si può scrivere come:

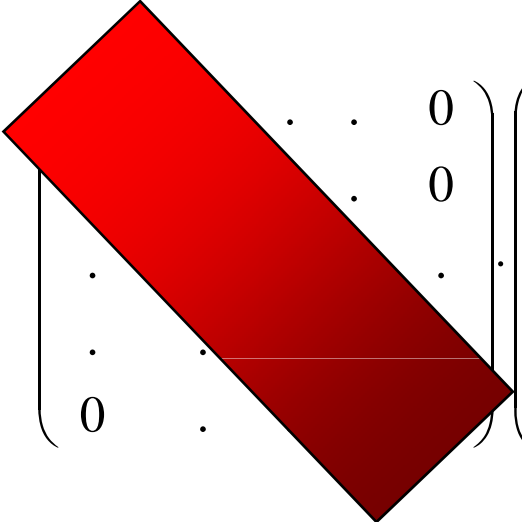
$$\frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} \approx g_i$$

Allora l'eq. di **Poisson** in **1D** diventa (con opportune condizioni al contorno) :

$$\frac{V_{i+1} + V_{i-1} - 2V_i}{h^2} \approx -\frac{\rho_i}{\varepsilon}$$

Matrici Tridiagonali

e quindi possiamo scrivere l'equazione nella forma *matriciale* $AX=B$:


$$\begin{pmatrix} \cdot & \cdot & \cdot & 0 \\ & \cdot & \cdot & 0 \\ \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & & & \cdot \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ \cdot \\ \cdot \\ V_N \end{pmatrix} = -\frac{h^2}{\varepsilon} \begin{pmatrix} \rho_1 + \frac{\varepsilon}{h^2} V_0 \\ \rho_2 \\ \cdot \\ \cdot \\ \rho_N + \frac{\varepsilon}{h^2} V_{N+1} \end{pmatrix}$$

dove A è una matrice **tridiagonale**, cioè ha elementi **non nulli** solo sulla **diagonale** e sulla diagonale ± 1 **colonna**;

N.B. se A è **tridiagonale** l'equazione $AX=B$ si può risolvere, ad esempio con uno dei metodi descritti precedentemente, in maniera molto efficiente, ad un **costo** computazionale che scala come **N** .

Matrici Tridiagonali

Se A è **tridiagonale**, $AX=B$ si può scrivere come:

$$\begin{pmatrix} \beta_1 & \gamma_1 & \cdot & \cdot & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \alpha_N & \beta_N \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \cdot \\ \cdot \\ b_N \end{pmatrix}$$

N.B. se A è **tridiagonale** non serve immagazzinare l'intera matrice $N \times N$,
ma solamente gli elementi $\neq 0$, cioè solo **3 vettori** di ordine N , con
 α_1 e γ_N non definiti e non usati :

$$\vec{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_N)$$

$$\vec{\beta} = (\beta_1, \beta_2, \dots, \beta_N)$$

$$\vec{\gamma} = (\gamma_1, \gamma_2, \dots, \gamma_N)$$

Matrici

Si può generalizzare la *discretizzazione* delle equazioni precedenti a **2D, 3D, ...,** anche se, in generale, la matrice risultante **non sarà tridiagonale**, ad esempio, l'eq. di **Poisson**, in 2D, diventa :

$$\frac{V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{ij}}{h^2} \approx -\frac{\rho_{ij}}{\varepsilon}$$

e la **coppia** di indici $\{i,j\}$, con $i,j=1,2,\dots,N$ si può far corrispondere ad un **unico** indice l , con $l=1,2,\dots,N^2$, allo scopo di definire una matrice $N^2 \times N^2$, ad esempio usando lo schema seguente:

Matrici

(i, j)	\Rightarrow	l
(1,1)	\Rightarrow	1
(1,2)	\Rightarrow	2
...	\Rightarrow	...
(1,N)	\Rightarrow	N
(2,1)	\Rightarrow	$N+1$
(2,2)	\Rightarrow	$N+2$
...	\Rightarrow	...
(2,N)	\Rightarrow	$2N$
...	\Rightarrow	...
(N,1)	\Rightarrow	$N^2 - N + 1$
(N,2)	\Rightarrow	$N^2 - N + 2$
...	\Rightarrow	...
(N,N)	\Rightarrow	N^2

e anche se la matrice $N^2 \times N^2$ risultante **non è** in generale **tridiagonale** (*verificare*), sarà comunque “**sparsa**”, il che consente sempre di ridurre il costo computazionale rispetto al caso generale.

Metodi Computazionali della Fisica

Il problema agli autovalori

Problema da risolvere

se, al posto dell'equazione matriciale $AX = B$ (1)
dobbiamo risolvere l'equazione :

$$AX = \lambda X \quad (2)$$

con X **vettore incognito** e λ **costante** (scalare) **incognita**, allora **invertire** la matrice A , che è la ricetta seguita per risolvere la (1), adesso **non** è di molto aiuto, in quanto:

$$A^{-1} \cdot AX = A^{-1}(\lambda X) \Rightarrow X = A^{-1}(\lambda X)$$

ma adesso (λX) contiene l'**incognita** !

L'equazione (2) rappresenta il **problema agli autovalori**, che è **più difficile** da risolvere della (1) perchè esistono **soluzioni solo** per certi valori della costante λ (o anche **nessuna** soluzione, a seconda di A).

Problema da risolvere

N.B. : la (2) è del tipo (1): $A'X = B'$

con : $A' = (A - \lambda I)$, $B' = 0$

e corrisponde al sistema di equazioni, **lineari, omogenee** (che ammette soluzioni solo se il determinante dei coefficienti è nullo):

$$\left\{ \begin{array}{l} a'_{11}x_1 + a'_{12}x_2 + \dots + a'_{1N}x_N = 0 \\ a'_{21}x_1 + a'_{22}x_2 + \dots + a'_{2N}x_N = 0 \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ \quad \cdot \quad \quad \cdot \quad \quad \quad \cdot \quad \quad \cdot \\ a'_{N1}x_1 + a'_{N2}x_2 + \dots + a'_{NN}x_N = 0 \end{array} \right.$$

Problema da risolvere

se riscriviamo la (2) come : $(A - \lambda I)X = 0$

allora, moltiplicando a sinistra per : $(A - \lambda I)^{-1}$ otteniamo:

$$(A - \lambda I)^{-1} \cdot (A - \lambda I)X = IX = 0 \Rightarrow X = 0$$

ma $X=0$ è una **soluzione banale** !

Ciò significa che, se esiste una soluzione (più interessante) **non banale** ($X \neq 0$), ci deve essere “qualcosa” che impedisce di moltiplicare per $(A - \lambda I)^{-1}$; questo “qualcosa” è la **non-esistenza** della matrice **inversa** di $(A - \lambda I)$.

Poiché il calcolo dell'inversa di una matrice implica dividere per il suo **determinante**, allora $(A - \lambda I)^{-1}$ **non esiste** (e quindi può esistere una soluzione **non banale** del problema agli autovalori) quando:

Problema da risolvere

$$\det(A - \lambda I) = 0 \quad (3)$$

e i valori di λ che soddisfano la (3) sono detti **autovalori** dell'equazione originaria (2).

Se siamo interessati **solo agli autovalori**, in linea di principio basta:

- calcolare il **determinante** della matrice $(A - \lambda I)$;
- trovare gli **zeri** della funzione (di λ) $\det(A - \lambda I)$.

Però la maniera **standard** (consente di calcolare anche gli **autovettori**) di risolvere l'equazione (2) è mediante la **diagonalizzazione**; l'**idea fondamentale** è quella di effettuare successivi cambiamenti dei **vettori di base** in maniera tale che ogni cambiamento lasci gli autovalori inalterati, continuando però a **diminuire** il valore assoluto degli elementi di A **fuori** dalla diagonale.

Problema da risolvere

Tale sequenza di trasformazioni è equivalente ad operare in continuazione sull'equazione originaria (2) con una matrice U tale che: $UA(U^{-1}U)X = \lambda UX \Rightarrow (UAU^{-1})(UX) = \lambda(UX)$

finchè UAU^{-1} è diagonale:

$$UAU^{-1} = A' = \begin{pmatrix} a'_{11} & 0 & . & . & 0 \\ 0 & a'_{22} & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & . & . & . & a'_{NN} \end{pmatrix}$$

Allora i valori di (UAU^{-1}) lungo la diagonale, $\{a'_{ii}\}$ sono gli **autovalori** della (2): λ_i , $i=1,\dots,N$; inoltre si possono scegliere i vettori UX in modo che siano i vettori di base, normalizzati:

Problema da risolvere

$UX_i = E_i \Rightarrow X_i = U^{-1}E_i$ sono gli **autovettori**, cioè sono le **colonne** della matrice U^{-1} ; gli autovettori sono anche tali da soddisfare:

$$(A - \lambda_i I)X_i = 0, \quad i = 1, \dots, N$$

N.B. in generale gli autovalori λ_i e gli autovettori X_i possono coinvolgere numeri **complessi**; in particolare, però, gli autovalori di una matrice **reale e simmetrica** ($a_{ij}=a_{ji}$) sono **tutti reali** e gli autovettori sono tutti **reali ed ortonormali**;

inoltre gli autovalori λ_i non sono necessariamente tutti **distinti**: autovalori **uguali** si dicono **degeneri**.

Metodo iterativo di Jacobi

Il metodo **iterativo** di Jacobi si applica a matrici **reali** e **simmetriche** ed è un metodo **affidabile**, ed efficiente per matrici fino alle dimensioni (10×10) e utilizzabile in pratica fino a matrici dell'ordine (20×20) ; esso consiste di una **sequenza** di trasformazioni del tipo:

$$\begin{aligned} A \rightarrow U_1^{-1} A U_1 \rightarrow U_2^{-1} U_1^{-1} A U_1 U_2 \rightarrow \\ \rightarrow U_3^{-1} U_2^{-1} U_1^{-1} A U_1 U_2 U_3 \rightarrow \dots \end{aligned}$$

dove ogni trasformazione (“**rotazione di Jacobi**”) serve ad “**azzerare**” uno degli elementi **fuori dalla diagonale** della matrice A : in pratica, con successive trasformazioni, gli elementi fuori dalla diagonale diventano sempre più piccoli finchè la matrice si riduce in forma **diagonale**, entro la precisione della macchina (computer).

Metodo iterativo di Jacobi

Allora, alla fine della procedura, gli elementi della matrice diagonale finale sono gli **autovalori**, mentre la matrice ottenuta come **prodotto** delle trasformazioni $U_1 \cdot U_2 \cdot U_3 \cdots$, non è nient'altro che la matrice degli **autovettori** (le **colonne** della matrice sono gli autovettori).

La “rotazione di Jacobi” **basica** è una matrice della forma:

$$U_{pq} = \begin{pmatrix} 1 & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & c & . & s & . \\ . & . & . & 1 & . & . \\ . & . & -s & . & c & . \\ . & . & . & . & . & 1 \end{pmatrix}$$

Diagrammatic annotations: A dashed pink box highlights the 2x2 submatrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ in the middle of the matrix. A horizontal dashed pink arrow labeled p points to the third row, and another labeled q points to the fifth column. Vertical dashed pink arrows labeled p and q point to the third and fifth rows respectively.

- tutti gli elementi **diagonali** sono 1 tranne i 2 elementi c nelle righe e colonne p e q ;
- tutti gli elementi **fuori** dalla diagonale sono 0 tranne i 2 elementi s e $-s$
- **N.B.** qui gli indici p e q non denotano gli elementi di U ma identificano la matrice (cioè il **tipo** di “rotazione”).

Metodo iterativo di Jacobi

I numeri c ed s sono, rispettivamente, il COSENO e il SENNO di un **angolo di rotazione** φ , cosicchè $c^2 + s^2 = 1$; allora, applicando la “rotazione” U_{pq} , otteniamo: $A' = U_{pq}^{-1} A U_{pq} = U_{pq}^T A U_{pq}$ (4) dove la *trasposta* di U_{pq} è uguale all'*inversa* perchè U_{pq} è una matrice **ortogonale**; U_{pq}^T cambia **solo le righe** p e q di A , mentre $A U_{pq}$ cambia **solo le colonne** p e q ; perciò gli elementi **cambiati** di A (in A') sono **solo** nelle **righe** e **colonne** p e q :

$$A' = \begin{pmatrix} \cdot & \cdot & a'_{1p} & \cdot & a'_{1q} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a'_{p1} & \cdot & a'_{pp} & \cdot & a'_{pq} & \cdot & a'_{pN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a'_{q1} & \cdot & a'_{qp} & \cdot & a'_{qq} & \cdot & a'_{qN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & a'_{Np} & \cdot & a'_{Nq} & \cdot & \cdot \end{pmatrix}$$

Metodo iterativo di Jacobi

Applicando esplicitamente la (4) e sfruttando la **simmetria** di A si ottengono le formule esplicite :

$$\left\{ \begin{array}{l} a'_{rp} = ca_{rp} - sa_{rq} \\ a'_{rq} = ca_{rq} + sa_{rp} \quad r \neq p, r \neq q \\ a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq} \\ a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq} \\ a'_{pq} = (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \end{array} \right. \quad (5)$$

L'**idea** del metodo di Jacobi è quella di **azzerare** gli elementi **fuori** dalla diagonale mediante una **serie di rotazioni** nel piano; ponendo $a'_{pq}=0$ nell'ultima delle (5) si ottiene:

Metodo iterativo di Jacobi

$$(c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) = 0 \Rightarrow \frac{(c^2 - s^2)}{sc} = \frac{a_{qq} - a_{pp}}{a_{pq}}$$

$$\Rightarrow \frac{(c^2 - s^2)}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}} \quad c = \cos \varphi, \quad s = \sin \varphi$$

$$\Rightarrow \frac{(c^2 - s^2)}{2sc} = \frac{\cos^2 \varphi - \sin^2 \varphi}{2 \cos \varphi \sin \varphi} = \frac{\cos(2\varphi)}{\sin(2\varphi)} = \cot(2\varphi) \equiv \vartheta$$

se adesso poniamo $t \equiv s/c = \tan(\varphi)$, allora possiamo scrivere:

$$t^2 + 2t\vartheta - 1 = 0 \quad (6)$$

Metodo iterativo di Jacobi

la radice **più piccola** di questa equazione corrisponde ad un angolo di rotazione $< \pi/4$ (in valore assoluto) e questa scelta dà la “riduzione” più stabile; la radice **più piccola** corrisponde a :

$$t = -\vartheta + \sqrt{\vartheta^2 + 1} \Rightarrow t = \frac{1}{\vartheta + \sqrt{\vartheta^2 + 1}} \quad \vartheta \geq 0$$

$$t = -\vartheta - \sqrt{\vartheta^2 + 1} \Rightarrow t = \frac{-1}{-\vartheta + \sqrt{\vartheta^2 + 1}} \quad \vartheta < 0$$

$$\Rightarrow t = \frac{\text{sgn}(\vartheta)}{|\vartheta| + \sqrt{\vartheta^2 + 1}} \Rightarrow \frac{1}{\sqrt{t^2 + 1}} = \frac{1}{\sqrt{\frac{s^2}{c^2} + 1}} = \frac{1}{\sqrt{\frac{(s^2 + c^2)}{c^2}}} = c$$

$$\Rightarrow \begin{cases} c = \frac{1}{\sqrt{t^2 + 1}} \\ s = tc \end{cases}$$

Metodo iterativo di Jacobi

Dal punto di vista pratico bisogna usare le equazioni (5) **numericamente** in modo da **minimizzare** gli errori di arrotondamento; a tal fine l'ultima delle (5) è sostituita dall'espressione :

$$a'_{pq} = 0 \quad (7)$$

e l'**idea**, nelle altre equazioni (5), è quella di porre la quantità **nuova** uguale a quella **vecchia** più una **piccola correzione**; ad esempio usiamo l'ultima delle (5) e la (7) per eliminare a_{qq} dalla terza delle (5) per ottenere:

$$\begin{aligned} a'_{pq} = 0 &= (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq}) \Rightarrow a_{qq} = a_{pp} + \frac{(c^2 - s^2)}{sc}a_{pq} \\ \Rightarrow a'_{pp} &= c^2a_{pp} + s^2a_{qq} - 2sca_{pq} \Rightarrow a'_{pp} = c^2a_{pp} + s^2a_{pp} + s\frac{(c^2 - s^2)}{c}a_{pq} - 2sca_{pq} = \\ &= (c^2 + s^2)a_{pp} + \frac{s}{c}(c^2 - s^2 - 2c^2)a_{pq} = a_{pp} - ta_{pq} \end{aligned}$$

Metodo iterativo di Jacobi

e analogamente le altre; allora si ottiene:

$$\begin{cases} a'_{pp} = a_{pp} - ta_{pq} \\ a'_{qq} = a_{qq} + ta_{pq} \\ a'_{rp} = a_{rp} - s(a_{rq} + \tau a_{rp}) \\ a'_{rq} = a_{rq} + s(a_{rp} - \tau a_{rq}) \end{cases}$$

dove: $\tau \equiv \frac{s}{1+c} = \tan(\varphi/2)$

e si può controllare la **convergenza** del metodo di Jacobi considerando la somma dei quadrati degli elementi **fuori** dalla diagonale:

$$S = \sum_{r \neq s} |a_{rs}|^2$$

Metodo iterativo di Jacobi

utilizzando le (5) si può dimostrare che:

$$S' = S - 2 |a_{pq}|^2$$

cioè la sequenza delle S **diminuisce** monotonicamente.

Alla fine si ottiene una matrice D , **diagonale** entro la precisione della macchina:

$$D = V^T A V, \quad V = U_1 U_2 U_3 \dots$$

allora gli elementi **diagonali** di D sono gli **autovalori** della matrice originale A e le **colonne** di V (poichè $AV=VD$) sono gli **autovettori**: possono essere ottenuti applicando la formula $V' = VU_i$ ad ogni passo del calcolo (dove inizialmente V è la matrice unitaria); allora si ottengono le equazioni:

Metodo iterativo di Jacobi

$$\begin{cases} v'_{rs} = v_{rs} & s \neq p, s \neq q \\ v'_{rp} = cv_{rp} - sv_{rq} \\ v'_{rq} = sv_{rp} + cv_{rq} \end{cases}$$

in pratica bisogna decidere in quale **ordine** azzerare gli elementi fuori diagonale; nell'algoritmo **originale** di Jacobi (1864) si azzerava ogni volta l'elemento fuori diagonale **più grande**, però una tale strategia va bene per il calcolo **manuale** ma non per il calcolo al computer in quanto solo la ricerca dell'elemento più grande rende ogni rotazione di Jacobi un processo di ordine N^2 (invece di N).

Metodo iterativo di Jacobi

Una strategia migliore per il calcolo numerico è il **metodo di Jacobi ciclico**, nel quale gli elementi vengono azzerati seguendo un **ordine preciso**, ad esempio: $U_{12}, U_{13}, \dots, U_{1N}$

$$U_{23}, U_{24}, \dots$$

un tale insieme di $N(N-1)/2$ rotazioni di Jacobi è chiamato uno “**sweep**” (una “spazzolata”); tipicamente sono richiesti tra 6 e 10 “sweeps” per raggiungere la convergenza, che corrisponde a $3N^2 \div 5N^2$ rotazioni di Jacobi e, poichè ogni rotazione richiede dell’ordine di ($4N$) operazioni (ognuna fatta di una moltiplicazione e di una addizione), il costo complessivo è dell’ordine di $12N^3 \div 20N^3$ operazioni ($\sim N^3$); se si vogliono calcolare anche gli **autovettori** allora ogni rotazione richiede $6N$ operazioni (al posto di $4N$) il che implica un costo aggiuntivo di circa il 50% .

Metodo iterativo di Jacobi

Il **programma** seguente, in C, implementa il metodo di Jacobi ciclico, descritto precedentemente, con due ulteriori **miglioramenti**:

- nei primi 3 “sweeps” si esegue la rotazione pq solo se:

$$|a_{pq}| > \delta = \frac{1}{5} \frac{S_0}{N^2}, \quad S_0 = \sum_{r < s} |a_{rs}|$$

- dopo 4 “sweeps”, se $|a_{pq}| \ll |a_{pp}|$ e $|a_{pq}| \ll |a_{qq}|$ allora si pone $|a_{pq}| = 0$ e si salta la rotazione; il criterio usato in pratica è:

$$|a_{pq}| < 10^{-(D+2)} |a_{pp}|$$

dove D è il numero di cifre decimali significative sul computer (e analogamente per $|a_{qq}|$).

Metodo iterativo di Jacobi

Jacobi.cpp

```
/*
*****
*   Calcola tutti gli AUTOVALORI e gli AUTOVETTORI di una matrice   *
*   REALE, SIMMETRICA col metodo di Jacobi:                          *
*   in "ouput" gli elementi della matrice 'a' sopra la diagonale sono *
*   distrutti, il vettore 'd' contiene gli autovalori e la matrice 'v' *
*   ha, come colonne, gli autovettori normalizzati; 'nrot' registra il *
*   numero di rotazioni di Jacobi necessarie,                        *
*****
*/

#include <iostream>
#include <cmath>

const int N = 3;                                /* ordine della matrice */

float **matrix(int nrl,int nrh,int ncl,int nch);
float *vector(int nl,int nh);
int jacobi(float **a,int n,float d[],float **v,int *nrot);

int main()
{
    using namespace std;
    float d[N];
    float **a,**v;
    int i,j,nrot;

    a=matrix(1,N,1,N);
    v=matrix(1,N,1,N);
```

← N.B. in questo programma vettori e matrici hanno elementi compresi tra **1** e **N** (più comodo !) invece che (come assunto implicitamente dal linguaggio C) tra **0** e **N-1** !

Metodo iterativo di Jacobi

```
/* assegna i valori della matrice a */

a[1][1]=-2.;
a[1][2]=2.;
a[1][3]=-1.;
a[2][1]=2.;
a[2][2]=1.;
a[2][3]=-2.;
a[3][1]=-1.;
a[3][2]=-2.;
a[3][3]=0.;

jacobi(a,N,d,v,&nrot);

cout << "\n AUTOVETTORI :\n";
for (i=1;i<=N;i++)
{
    for(j=1;j<=N;j++)
    {
        cout << i<< " "<<j<< " "<<v[i][j]<< endl;
    }
}
cout << "\n AUTOVALORI X:\n";
for (i=1;i<=N;i++)
{
    cout <<i<<" "<<d[i]<< endl;
}
cout<<"\n # di rotazioni:";
cout<<nrot<< endl;
}
/*-----end of main program-----*/
```

Metodo iterativo di Jacobi

```
#define ROTATE(a,i,j,k,l) g=a[i][j];h=a[k][l];a[i][j]=g-s*(h+g*tau);\
    a[k][l]=h+s*(g-h*tau);

int jacobi(float **a,int n,float d[],float **v,int *nrot)
{
    using namespace std;
    int j,iq,ip,i;
    float tresh,theta,tau,t,sm,s,h,g,c,*b,*z;

    b=vector(1,n);
    z=vector(1,n);
    for (ip=1;ip<=n;ip++) {
        for (iq=1;iq<=n;iq++) v[ip][iq]=0.0;
        v[ip][ip]=1.0;
    }
    for (ip=1;ip<=n;ip++) {
        b[ip]=d[ip]=a[ip][ip];
        z[ip]=0.0;
    }
    *nrot=0;
    for (i=1;i<=50;i++) {
        sm=0.0;
        for (ip=1;ip<=n-1;ip++) {
            for (iq=ip+1;iq<=n;iq++)
                sm += fabs(a[ip][iq]);
        }
        if (sm == 0.0) {
            return 0;
        }
        if (i<4)
            tresh=0.2*sm/(n*n);
        else
            tresh=0.0;
    }
}
```

Metodo iterativo di Jacobi

```
for (ip=1;ip<=n-1;ip++) {
    for (iq=ip+1;iq<=n;iq++) {
        g=100.0*fabs(a[ip][iq]);
        if (i>4 && fabs(d[ip])+g == fabs(d[ip])
            && fabs(d[iq])+g == fabs(d[iq]))
            a[ip][iq]=0.0;
        else if (fabs(a[ip][iq]) > tresh) {
            h=d[iq]-d[ip];
            if (fabs(h)+g == fabs(h))
                t=(a[ip][iq])/h;
            else {
                theta=0.5*h/(a[ip][iq]);
                t=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
                if (theta<0.0) t=-t;
            }
            c=1.0/sqrt(1.+t*t);
            s=t*c;
            tau=s/(1.+c);
            h=t*a[ip][iq];
            z[ip] -= h;
            z[iq] += h;
            d[ip] -= h;
            d[iq] += h;
            a[ip][iq]=0.0;
            for (j=1;j<=ip-1;j++) {
                ROTATE(a,j,ip,j,iq)
            }
            for (j=ip+1;j<=iq-1;j++) {
                ROTATE(a,ip,j,j,iq)
            }
            for (j=iq+1;j<=n;j++) {
                ROTATE(a,ip,j,iq,j)
            }
            for (j=1;j<=n;j++) {
                ROTATE(v,j,ip,j,iq)
            }
        }
        ++(*nrot);
    }
}
```


Metodo iterativo di Jacobi

```
        }  
    }  
    for (ip=1;ip<=n;ip++) {  
        b[ip] += z[ip];  
        d[ip] = b[ip];  
        z[ip] = 0.0;  
    }  
    }  
    cout <<"too many iterations in routine JACOBI"<< endl;  
}  
/*-----end of jacobi-----*/
```

Metodo iterativo di Jacobi

```
float *vector(int nl,int nh)
/* allocates an float vector with range [nl..nh] */
{
    using namespace std;
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) cout <<"allocation failure in vector"<< endl;
    return v-nl;
}

float **matrix(int nrl,int nrh,int ncl,int nch)
/* allocates a float matrix with range [nrl..nrh][ncl..nch] */
{
    using namespace std;
    int i;
    float **m;
    /* allocates pointers to rows */
    m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float));
    if (!m) cout<<"allocation failure 1 in matrix"<< endl;
    m -= nrl;

    /* allocates rows and set pointers to them */
    for (i=nrl;i<=nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) cout<<"allocation failure 2 in matrix"<< endl;
        m[i] -= ncl;
    }
    return m;
}
```

Metodo iterativo di Jacobi

esempio (a mano !): $AX = \lambda X$, con:

$$A = \begin{pmatrix} -2 & 2 \\ 2 & 1 \end{pmatrix}$$

allora basta una **sola** rotazione di Jacobi con:

$$A' = U_{12}^T A U_{12}$$

$$\begin{cases} a'_{11} = a_{11} - ta_{12} \\ a'_{22} = a_{22} + ta_{12} \\ t = s / c \end{cases} \Rightarrow \lambda_1 = -3, \quad \lambda_2 = 2$$
$$X_1 = \frac{c}{2} \begin{pmatrix} +2 \\ -1 \end{pmatrix}, \quad X_2 = \frac{c}{2} \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Metodo iterativo di Jacobi

esempio (numerico): $AX = \lambda X$, con:

$$A = \begin{pmatrix} -2 & 2 & -1 \\ 2 & 1 & -2 \\ -1 & -2 & 0 \end{pmatrix}$$

e il risultato, dopo 7 iterazioni (rotazioni) di Jacobi, è:

$$\lambda_1 = -3.000000, \quad \lambda_2 = 3.449490, \quad \lambda_3 = -1.449490$$

$$X_1 = \begin{pmatrix} +0.894427 \\ -0.447214 \\ 0.000000 \end{pmatrix}, \quad X_2 = \begin{pmatrix} +0.375266 \\ +0.750533 \\ -0.543945 \end{pmatrix}, \quad X_3 = \begin{pmatrix} 0.243259 \\ 0.486519 \\ 0.839121 \end{pmatrix}$$

Eq. di Schrödinger

Un tipico problema agli **autovalori** in Fisica è rappresentato dall'**equazione di Schrödinger** indipendente dal tempo :

$$\left(-\frac{\hbar^2}{2m} \nabla^2 + V(\vec{r}) \right) \psi_E(\vec{r}) = E \psi_E(\vec{r})$$

in **Meccanica Quantistica** descrive una **particella singola** (esempio: stati elettronici dell'atomo di Idrogeno).

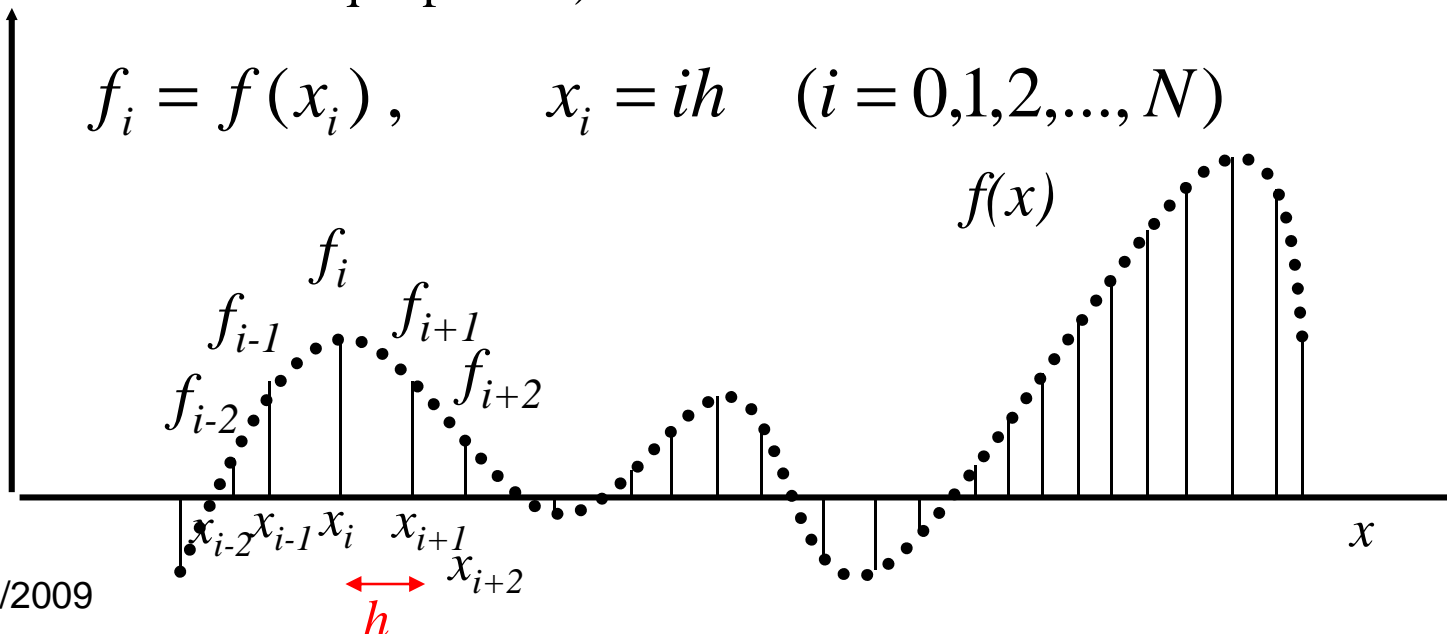
Eq. di Schrödinger

Il *Laplaciano* è definito come : $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$, $\vec{r} = (x, y, z)$

Allora, ad esempio, in **1D** bisogna risolvere equazioni del tipo:

$$\frac{d^2 f(x)}{dx^2} = g(x)$$

Per usare un approccio numerico, **discretizziamo** le funzioni su di una griglia (non necessariamente equispaziata):



Eq. di Schrödinger

Ricordando l'espressione già ricavata per stimare numericamente la derivata seconda:

$$\left. \frac{d^2 f(x)}{dx^2} \right|_{x=x_i} = \frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} + \mathcal{O}(h^2)$$

L'equazione si può scrivere come:

$$\frac{f_{i+1} + f_{i-1} - 2f_i}{h^2} \approx g_i$$

Allora l'eq. di **Schrödinger** diventa :

$$\frac{\psi_{i+1} + \psi_{i-1} - 2\psi_i}{h^2} \approx -\frac{2m}{\hbar^2} (E - V_i) \psi_i$$

Eq. di Schrödinger

Considerando *unità atomiche* ($m=1$, $\hbar=1$, $e=1$) l'eq. di **Schrödinger** diventa :

$$\frac{\psi_{i+1} + \psi_{i-1} - 2\psi_i}{h^2} = -2(E - V_i)\psi_i$$

\Downarrow

$$\frac{\psi_{i+1} + \psi_{i-1} - (2 + 2h^2V_i)\psi_i}{h^2} = -2E\psi_i$$

Eq. di Schrödinger

con opportune condizioni al contorno , ad esempio se :

$$\psi_{i=-N-1} = 0$$

$$\psi_{i=N+1} = 0$$

allora abbiamo :

$$\frac{\psi_{-N+1} - (2 + 2h^2 V_{-N})\psi_{-N}}{h^2} = -2E\psi_{-N}$$

$$\frac{\psi_{N-1} - (2 + 2h^2 V_N)\psi_N}{h^2} = -2E\psi_N$$

Esercizio 4b: particella in buca

- **usare** (con opportune modifiche) il programma precedente in modo da calcolare gli autovalori E_i dell'equazione di Schrödinger in **1D** (usare una griglia di $N=501$ punti in **1D**, tra $x=-10$ e $x=+10$), che descrive una particella sottoposta al **potenziale** di una **buca** (con $\alpha=1$ e $\lambda=4$ parametri dati) :

$$V(x) = \frac{\hbar^2}{2m} \alpha^2 \lambda (\lambda - 1) \left[\frac{1}{2} - \frac{1}{\cosh^2(\alpha x)} \right]$$

- **quanti** sono gli **stati legati** ?
- **confrontare** con la soluzione *analitica* (per stati **legati**) :

$$E_i = \frac{\hbar^2}{2m} \alpha^2 \left[\frac{\lambda(\lambda-1)}{2} - (\lambda-1-i)^2 \right] , \quad i = 0, 1, 2, \dots$$

Esercizio 4b: particella in buca

- suggerimento: usare *unità atomiche* (a.u.): $\hbar=m=1$;
- determinare e graficare (in corrispondenza dei punti della griglia) il *potenziale* $V(x)$ e gli *autovettori* corrispondenti ai 3 *autovalori più bassi* in energia;
- usando valori **diversi** del numero di punti N (e quindi dell'*ordine* della matrice) verificare approssimativamente come **aumenta** il numero delle *rotazioni di Jacobi* necessarie, all'aumentare di N .