Rappresentazione dei numeri reali in un calcolatore

Numeri floating point

$$\alpha = \text{segno}(\alpha) m \beta^p$$

- $\text{segno}(\alpha)$ vale ± 1 a seconda del segno di α;
- $m = \sum_{i=1}^{\infty} (\alpha_i \beta^{-i}) = (0.\alpha_1\alpha_2 \dots)_\beta$ è un numero reale < 1 chiamato **mantissa**;
- β^p è un numero intero chiamato **esponente** di α

Se α_1 (prima cifra della mantissa) è $\neq 0$, si dice che il numero è rappresentato in forma **normalizzata**.

Errore di rappresentazione

Rappresentare un numero reale in un calcolatore comporta inevitabilmente degli **errori di rappresentazione**: un calcolatore ha a disposizione un numero **finito** di cifre per rappresentare un numero reale, il quale però potrebbe essere composto da un numero **infinito** di cifre (es. numeri irrazionali o numeri periodici), o semplicemente da più cifre di quante il calcolatore riesce a gestire.

L'**errore assoluto** commesso approssimando un numero $\alpha \in \mathbb{R}$ con un altro numero $\alpha^* \in \mathbb{R}$ è:

$$E_a = |\alpha - \alpha^*|$$

L'**errore relativo** è:

$$E_r = \frac{E_a}{|\alpha|}$$

Teorema dell'errore di rappresentazione dei numeri reali

L'errore relativo che si commette approssimando un numero reale α con un numero floating $fl(\alpha)$ point non supera mai la **precisione di macchina** *u*:

$$\frac{|fl(\alpha) - \alpha|}{|\alpha|} \leq k\beta^{1-t} = u \quad \text{equiv.} \quad fl(\alpha) = \alpha(1+\epsilon) \quad |\epsilon| \leq u$$

dove *k* assume un valore diverso a seconda che la rappresentazione $fl(\alpha)$ sia ottenuta per arrotondamento o per troncamento.

La precisione di macchina è il più piccolo numero reale *u* tale che $fl(1+u) \neq 1$.

Analisi degli errori

Errore inerente ed errore algoritmico

L'**errore inerente** è l'errore che si commette se le operazioni fossero eseguite in aritmetica esatta, ma i dati fossero memorizzati con un numero di cifre finito. Dipende solo dai dati e da come essi sono legati alla soluzione del problema.

L'**errore algoritmico** è l'errore che si commetterebbe se i dati fossero rappresentati esattamente ma le operazioni fossero eseguite in aritmetica finita. Dipende sia dall'algoritmo che dai dati.

Condizionamento

Un problema si dice **ben condizionato** se a piccole perturbazioni sugli input corrispondono altrettanto piccole perturbazioni sugli output.

Il condizionamento è una proprietà della funzione che lega i dati del problema alla sua soluzione. Non dipende da come le soluzioni vengono calcolate (quindi non dipende dall'algoritmo).

Esempio

Calcolare il valore $\alpha = 2 - \sqrt{4 - c}$.

Per $c = 4$, $\alpha = 2$.

Per $c^* = 4 - 10^{-6}$, $\alpha^* = 2 - 10^{-3}$.

Per valori di c vicini a 4 il problema è mal condizionato, perché le variazioni delle soluzioni sono di 3 ordini di grandezza superiori alle variazioni sui dati (10^{-3} contro 10^{-6}).

Stabilità

Un algoritmo si dice **stabile** se non è troppo sensibile agli errori di rappresentazione dovuti all'aritmetica finita.

La stabilità di un algoritmo dipende dal tipo e dall'ordine con cui vengono eseguite le operazioni.

Norme

Norme vettoriali

La funzione $\|\cdot\|: \mathbb{R}^n \to \mathbb{R}$ è una **norma vettoriale** se:

- $\|x\| \geq 0 \; \forall x \in \mathbb{R}^n$ e $\|x\| = 0 \Leftrightarrow x = 0$;
- $\|\lambda x\| = |\lambda| \|x\| \; \forall \lambda \in \mathbb{R}$;
- **proprietà triangolare**: $\|x + y\| \leq \|x\| + \|y\|$

norma ∞	$\|x\|_\infty = \max_{i=1,\dots,n} \|x_i\|$
norma 1	$\|x\|_1 = \sum_{i=1}^{n} \|x_i\|$
norma 2	$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = x^T x$

Le norme vettoriali sono tutte equivalenti. Per ogni coppia di norme $\|\cdot\|_*$, $\|\cdot\|_.$, qualunque esse siano, esistono sempre due costanti m ed M tali che:

$$m\|x\|_* \leq \|x\|_. \leq M\|x\|_*$$

Norme matriciali

La funzione $\|\cdot\|: \mathbb{R}^{n \times n} \to \mathbb{R}$ è una **norma matriciale** se:

- $\|A\| \geq 0 \forall A \in \mathbb{R}^{n \times n}$ e $\|A\| = 0 \Leftrightarrow A = 0$;
- $\|\lambda A\| = |\lambda| \|A\|$;
- **proprietà triangolare**: $\|A + B\| \leq \|A\| + \|B\|$
- **proprietà submoltiplicativa**: $\|AB\| \leq \|A\| \cdot \|B\|$
- **compatibilità con le norme vettoriali**: $\|Ax\| \leq \|A\| \cdot \|x\|$

Le norme sono importanti perché esprimono l'idea di una **misura**, in modo simile al valore assoluto per gli scalari. Per valutare gli errori tra vettori, infatti, si utilizzano le norme:

$$\frac{\|x - y\|}{\|x\|}$$

Sistemi lineari

Definizione del problema

Input: matrice dei coefficienti *A* e vettore (colonna) dei termini noti *b*:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Output: vettore $x \in \mathbb{R}^n$ che soddisfa l'uguaglianza

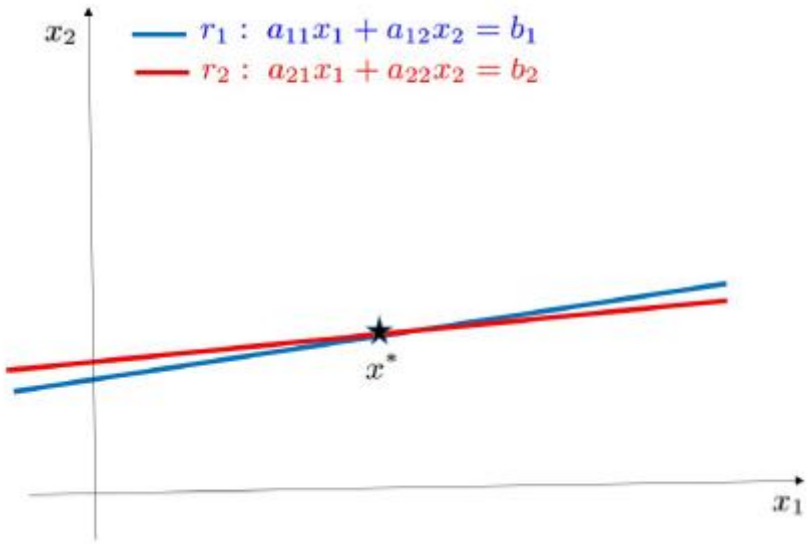
$$Ax = b$$

Se *A* è invertibile allora esiste una ed un'unica soluzione per il sistema (teorema di Rouché-Capelli).

Analisi del condizionamento

Un sistema lineare è mal condizionato quando esistono in *A* delle righe che sono "quasi" linearmente dipendenti.

Nel caso di un sistema con 2 equazioni in 2 incognite, dal punto di vista grafico il sistema è mal condizionato se le due rette sono quasi coincidenti..



Indice di condizionamento di una matrice

Sia x^* la soluzione del sistema $Ax = b$ e $\tilde{x}$ la soluzione del problema perturbato $Ax = b + \Delta b$.

$$A\tilde{x} = Ax^* + \Delta b \Rightarrow A(\tilde{x} - x^*) = \Delta b \Rightarrow \boxed{\tilde{x} - x^* = A^{-1}\Delta b}$$

Passando alle norme:

$$\|b\| = \|Ax^*\| \xrightarrow{\text{prop.submoltiplicativa}} \|b\| \leq \|A\|\|x^*\| \Rightarrow \boxed{\frac{1}{\|x^*\|} \leq \frac{\|A\|}{\|b\|}}$$

Passiamo alle norme nella 1° espressione e dividiamo entrambi i membri per $\|x^*\|$:

$$\tilde{x} - x^* = A^{-1}\Delta b \Rightarrow \frac{\|\tilde{x} - x^*\|}{\|x^*\|} = \frac{\|A^{-1}\Delta b\|}{\|x^*\|}$$

Applicando la proprietà submoltiplicativa al 2° membro si ottiene:

$$\frac{\|A^{-1}\Delta b\|}{\|x^*\|} \leq \frac{1}{\|x^*\|}\|A^{-1}\| \cdot \|\Delta \mathrm{b}\|$$

Quindi possiamo usare la maggiorazione ottenuta dall'espressione (2) per dire che:

$$\frac{\|A^{-1}\Delta b\|}{\|x^*\|} \leq \|A\|\|A^{-1}\|\frac{\|\Delta b\|}{\|b\|}$$

Il numero $k(A) = \|A\|\|A^{-1}\|$ è il **numero di condizionamento** di A. Abbiamo quindi una maggiorazione per l'errore relativo sulle soluzioni:

$$\frac{\|\tilde{x} - x^*\|}{\|x^*\|} \leq k(A)\frac{\|\Delta b\|}{\|b\|}$$

L'indice di condizionamento di una matrice, nel caso dei sistemi lineari, funge da **amplificatore** per l'errore.

Metodi per la risoluzione di sistemi lineari

Esistono due classi di metodi:

- **diretti**: con un numero finito di operazioni si calcola la soluzione;
- **iterativi**: si costruisce una **successione di vettori** che per $n \to \infty$ converge alla soluzione

Calcolo della matrice inversa

In linea teorica è possibile risolvere il sistema $Ax = b$ calcolando l'**inversa** di A:

$$x = A^{-1}b$$

Nella pratica, però, non si calcola mai l'inversa di una matrice, perché:

- è troppo costoso dal punto di vista computazionale;
- la stabilità è pessima

Sistemi diagonali

$$\begin{cases} d_1 x_1 = b_1 \\ d_2 x_2 = b_2 \\ \dots \\ d_n x_n = b_n \end{cases} \qquad A = \begin{pmatrix} d_1 & & \\ & d_2 & \\ & & \ddots \\ & & & d_n \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Si calcolano i componenti della soluzione come $x_i = \frac{b_i}{d_i}$:

```
function [x] = risolutore_sistema_diagonale(A, b)
    if (length(A) ~= length(b))
        error("Le dimensioni di A e di b non coincidono. Impossibile risolvere
il sistema.")
    end

    n = length(A);
    x = zeros(n, 1);

    for i = 1 : n
        x(i, 1) = b(i, 1) / A(i, i);
    end
```

Costo computazionale: $\Theta(n)$.

Sistemi triangolari

$$A = \underbrace{\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ & & \ddots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}}_{\text{inferiore}} \qquad A = \underbrace{\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ & a_{22} & \dots & a_{2n} \\ & & \ddots & \\ & & & a_{nn} \end{pmatrix}}_{\text{superiore}}$$

$$\underbrace{\begin{cases} a_{11}x_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}}_{\text{inferiore}} \qquad \underbrace{\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \dots \\ a_{nn}x_n = b_n \end{cases}}_{\text{superiore}}$$

Si esegue una **sostituzione in avanti o all'indietro**, in base a se il sistema è triangolare inferiore o superiore.

sostituzione in avanti

$$x_j = \frac{b_j - \sum_{i=1}^{j-1} r_{ji}x_i}{r_{jj}} \qquad j = 1, \dots, n$$

```
function [x] = risolutore_sistema_triangolare_inf(A, b)
    if (length(A) ~= length(b))
        error("Le dimensioni di A e b non coincidono. Impossibile risolvere il
sistema.");
    end

    n = length(A);
    x = zeros(n, 1);

    % calcolo la 1° componente fuori dal for perché non devo fare nessuna somma
    x(1) = b(1) / A(1, 1);

    % visto che ho già calcolato una componente, nel for devo fare un'iterazione
in meno
    % quindi inizio da 2 anzichè da 1
    for j = 2 : n
        x(j) = (b(j) - sum(A(j, 1 : j - 1) * x(1 : j - 1))) / A(j, j);
    end
```

sostituzione all'indietro

$$x_j = \frac{b_j - \sum_{i=j+1}^{n} r_{ji} x_i}{r_{jj}} \qquad j = n, \dots, 1$$

```
function [x] = risolutore_sistema_triangolare_sup(A, b)
    if (length(A) ~= length(b))
        error("Le dimensioni di A e b non coincidono. Impossibile risolvere il
sistema.");
    end

    n = length(A);
    x = zeros(n, 1);

    % calcolo l'ultima componente fuori dal for perchè non devo fare nessuna
somma
    x(n) = b(n) / A(n, n);

    % visto che una componente l'ho già calcolata, nel for devo fare
un'iterazione in meno
    % quindi inizio da (n - 1) anzichè da n
    for j = n - 1 : -1 : 1
        x(j) = (b(j) - sum(A(j, j + 1 : n) * x(j + 1 : n))) / A(j, j);
    end
```

Il costo computazionale è $\simeq O(n^2)$.

Stabilità degli algoritmi di sostituzione

Gli algoritmi di sostituzione diventano instabili quando gli elementi del triangolo inferiore sono molto grandi rispetto agli elementi diagonali.

Dimostrazione

Siano $x^* = (x_1^*, x_2^*)$ e $\tilde{x} = (\tilde{x}_1, \tilde{x}_2)$ rispettivamente la soluzione esatta e la soluzione calcolata in aritmetica finita.

Dal discorso sulla precisione di macchina sappiamo che:

$$\tilde{x}_1 = x_1^*(1 + \epsilon_1) \qquad |\epsilon_1| \leq u$$

Applichiamo l'algoritmo di sostituzione per calcolare $\tilde{x}_2$:

$$\tilde{x}_2 = \frac{b_2 - a_{21}\tilde{x}_1}{a_{22}} = \frac{b_2 - a_{21}x_1^*(1+\epsilon_1)}{a_{22}} = \frac{b_2 - a_{21}x_1^*}{a_{22}} + \frac{a_{21}x_1^*}{a_{22}}\epsilon_1 = x_2^* + \frac{a_{21}x_1^*}{a_{22}}\epsilon_1$$

Metodo di Cramer

Il metodo di Cramer può essere utilizzato per risolvere sistemi lineari qualsiasi. Consiste nel calcolare $n + 1$ determinanti di matrici di ordine *n*.

All'atto pratico anche questo metodo non viene mai utilizzato, perché ha un costo computazionale troppo alto (calcolare il determinante con il teorema di Laplace ha un costo di $n!$ somme e prodotti).

Metodo di Gauss (semplice)

Dato il sistema $Ax = b$, il metodo di Gauss consiste in due passaggi:

1. **fattorizzazione di Gauss**: si calcolano una matrice **triangolare inferiore** *L* ed una matrice **triangolare superiore** *U* tali che $A = LU$;
2. si risolve un sistema triangolare equivalente a quello iniziale. Dall'uguaglianza $A = LU$ si riscrive il sistema come:

$$LUx = b \Rightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

Costruzione della matrice U

La matrice U si costruisce a partire dalla matrice A andando ad **azzerare** gli elementi del **triangolo inferiore** mediante opportune combinazioni lineari. Al passo k:

- si calcola il moltiplicatore per la combinazione lineare:

$$m_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \qquad i = k+1, \dots, n$$

- si esegue la combinazione lineare tra la riga i e la riga k, con coefficiente $-m_{ik}$:

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

Costruzione della matrice L

Ogni combinazione lineare fatta al passo k per azzerare i coefficienti sotto al pivot delle righe successive può essere vista come un **prodotto matriciale** tra una matrice L_k e la matrice A_k.

La matrice L_k è una **matrice identità** che contiene i moltiplicatori scelti al passo k, cambiati di segno. I moltiplicatori servono per annullare il **triangolo inferiore** di A, quindi tutte le matrici L_k sono delle **triangolari inferiori a diagonale unitaria**.

La matrice L_k è detta *k*-esima **trasformazione elementare di Gauss** ed è fatta in questo modo:

$$L_k = I - m^{(k)} e_k^T$$

- $m^{(k)}$ è il vettore colonna dei moltiplicatori scelti al passo k;
- e_k^T il *k*-esimo vettore della **base canonica** (trasposto perché preso come colonna)

La sequenza dei prodotti matriciali che permette di ottenere la matrice U è:

$$L_{n-1} L_{n-2} \dots L_2 L_1 A = U$$

Per passare alla scrittura $A = LU$ si porta a destra il prodotto $L_{n-1} \dots L_1$ e l'inversa di questo prodotto si chiama L:

$$A = (L_{n-1} L_{n-2} \dots L_1)^{-1} U \Rightarrow \mathrm{A} = \underbrace{L_1^{-1} L_2^{-1} \dots L_{n-1}^{-1}}_{L} U \Rightarrow A = LU$$

Teorema

$$L_k = I - m^{(k)} e_k^T \Rightarrow L_k^{-1} = I + m^{(k)} e_k^T$$

Dimostrazione

Si tratta di dimostrare che $L_k L_k^{-1} = I$.

$$\left(I - m^{(k)} e_k^T\right)\left(I + m^{(k)} e_k^T\right) = I + m^{(k)} e_k^T - m^{(k)} e_k^T - m^{(k)} \underbrace{e_k^T m^{(k)}}_{=0} e_k^T = I$$

Teorema

$$L_k^{-1} L_j^{-1} = \begin{pmatrix} 1 & & & & \\ m_{k+1.k} & 1 & & & \\ \vdots & m_{j+1,j} & 1 & & \\ \vdots & \vdots & 0 & \ddots & \\ m_{nk} & m_{nj} & & & 1 \end{pmatrix} = I + m^{(k)} e_k^T + m^{(j)} e_j^T$$

Dimostrazione

$$\left(I + m^{(k)} e_k^T\right)\left(I + m^{(j)} e_j^T\right) = I + m^{(j)} e_j^T + m^{(k)} e_k^T + m^{(k)} e_k^T m^{(j)} e_j^T$$

$e_k^T m^{(j)} = 0$, quindi si ha la tesi.

Codice Matlab

Per ottimizzare l'algoritmo, anziché costruire due matrici L, U si eseguono tutte le operazioni direttamente sulla matrice A e le matrici L, U si estraggono alla fine dalla matrice A:

```
function [L, U] = fattorizzazione_gauss_semplice(A)
    n = length(A);

    for k = 1 : n - 1
        if (abs(A(k, k)) < eps)
            error("Pivot nullo. Fattorizzazione non eseguibile.");
        end

        % costruisco i moltiplicatori e li salvo nella matrice
        A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k);

        % faccio la combinazione lineare con la k + 1-esima riga
        A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - (A(k + 1 : n, k) *
A(k, k + 1 : n));
    end

    % la matrice L è il triangolo inferiore di A, SENZA LA DIAGONALE
    % la somma con eye(n) serve per dare ad L una diagonale unitaria
    L = tril(A, -1) + eye(n);

    % U è il triangolo superiore di A, con diagonale inclusa
    U = triu(A);
```

```
function [x] = risolutore_sistema_gauss_semplice(A, b)
    [L, U] = fattorizzazione_gauss_semplice(A);
    y = risolutore_sistema_triangolare_inf(L, b);
    x = risolutore_sistema_triangolare_sup(U, y);
```

Ipotesi per il metodo di Gauss semplice

Per usare il metodo di Gauss bisogna che tutti i pivot siano $\neq 0$. Questo però lo si scopre solo man mano che si procede con la fattorizzazione.

Un controllo che si può fare subito invece è quello sui **minori principali** di A: per poter applicare il metodo di Gauss devono essere tutti $\neq 0$, eccetto al più l'ultimo.

Questo NON significa imporre la condizione $det(A) \neq 0$: il determinante può essere $= 0$ ma il metodo di Gauss potrebbe essere utilizzabile lo stesso.

Una classe di matrici che soddisfa questa ipotesi è quella delle matrici **strettamente a diagonale dominante**.

Costo computazionale

- costo fattorizzazione: $\simeq O(n^3)$;
- costo risoluzione sistema equivalente: $\simeq O(n^2)$ (2 sistemi triangolari)

Costo totale: $\simeq O(n^3)$.

Metodo di Gauss con pivoting

Per estendere l'applicabilità del metodo di Gauss a tutte le matrici nonsingolari (e non soltanto quelle strettamente a diagonale dominante) si introduce la possibilità di **scambiare** le righe di A. Se gli stessi scambi vengono fatti anche sul vettore dei termini noti b, ciò equivale a scambiare di posto 2 equazioni all'interno del sistema.

Matrici di permutazione elementare

Una **matrice di permutazione elementare** è una **matrice identità** dove la riga i è stata scambiata con la riga j.

Per scambiare più di 2 righe si **moltiplicano** tra di loro più matrici di permutazione elementari:

$$P = P_{uv} \cdot P_{ij}$$

Proprietà delle matrici di permutazione elementari

- $det(P_{ij}) = -det(I) = -1$ (proprietà del determinante con scambi di riga);
- sono simmetriche: $P_{ij} = P_{ij}^T$
- sono ortogonali: $P_{ij}^T = P_{ij}^{-1}$

Moltiplicare la matrice A per una matrice di permutazione elementare P restituisce la matrice A con le righe (o colonne) scambiate:

- $P_{ij}A$ si ottiene scambiando la riga i con la riga j di A;
- AP_{ij} si ottiene scambiando la colonna i con la colonna j di A

Al passo k, oltre alla matrice L_k che contiene i moltiplicatori, si definisce anche una matrice P_k per eseguire gli scambi di riga necessari per procedere con la fattorizzazione.

A causa degli scambi di riga, la fattorizzazione non è più $A = LU$ ma $PA = LU$, con:

$$P = P_{n-1}P_{n-2} \dots P_2P_1$$

$$L = (L_1^{-1}P_2 \dots P_{n-1})(L_2^{-1}P_3 \dots P_{n-1})(L_{n-1}) = \prod_{k=1}^{n-1}[(P_{n-1} \dots P_{k+1})L_k]$$

Il sistema equivalente diventa:

$$\begin{cases} Ly = Pb \\ Ux = y \end{cases}$$

La fattorizzazione $PA = LU$ non è unica, perché ad ogni passo si può scegliere tra più permutazioni.

- in teoria qualunque permutazione che porti ad avere un pivot $\neq 0$ va bene;
- in pratica scegliere le permutazioni a caso potrebbe peggiorare la stabilità dell'algoritmo, quindi si usa la strategia del **pivoting parziale** (ad ogni passo si prende come pivot l'**elemento più grande**, in valore assoluto, della colonna che si sta considerando)

Codice Matlab

```
function [L, U, P] = fattorizzazione_gauss_pivoting(A)
    n = length(A);
    P = eye(n);

    for k = 1 : n - 1
        % cerco il massimo (in valore assoluto) nella k-esima colonna
        [~, index] = max(abs(A(k : n, k)));

        % l'indice del max è compreso tra [1, n - k]
        % questa istruzione serve per riportarlo ad un valore compreso tra [1,
n]
        index = index + k - 1;

        if (index ~= k) % faccio lo scambio solo se serve davvero
            % scambio la riga "index" con la riga "k" nella matrice A
            tmp = A(k, :);
            A(k, :) = A(index, :);
            A(index, :) = tmp;

            % scambio la riga "index" con la riga "k" nella matrice P
            tmp = P(k, :);
            P(k, :) = P(index, :);
            P(index, :) = tmp;
        end

        if (abs(A(k, k)) < eps)
            error("Pivot nullo. Fattorizzazione non eseguibile.");
        end

        % proseguo con il metodo di Gauss "standard"

        % costruisco i moltiplicatori e li salvo nella matrice A
        A(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k);

        % faccio la combinazione lineare con la k + 1-esima riga
        A(k + 1 : n, k + 1 : n) = A(k + 1 : n, k + 1 : n) - (A(k + 1 : n, k) *
A(k, k + 1 : n));
    end

    % la matrice L è il triangolo inferiore di A, SENZA LA DIAGONALE
    % la somma con eye(n) serve per dare ad L una diagonale unitaria
    L = tril(A, -1) + eye(n);

    % U è il triangolo superiore di A, con diagonale inclusa
    U = triu(A);
```

```
function [x] = risolutore_sistema_gauss_pivoting(A, b)
    [L, U, P] = fattorizzazione_gauss_pivoting(A);
    y = risolutore_sistema_triangolare_inf(L, P * b);
    x = risolutore_sistema_triangolare_sup(U, y);
```

Fattorizzazione di Gauss per matrici simmetriche – fattorizzazione LDL^T

Teorema

Se A è simmetrica e tutti i suoi minori principali sono $\neq 0$, allora esistono:

- una matrice L triangolare superiore con diagonale unitaria;
- una matrice diagonale D con tutti gli elementi diagonali $\neq 0$

tali che

$$A = LDL^T$$

Dimostrazione

Visto che le ipotesi del teorema di Gauss sono verificate, possiamo scrivere $A = LU$.

Consideriamo una matrice diagonale D i cui coefficienti sono gli stessi della U:

$$D = \begin{pmatrix} u_{11} & & \\ & \ddots & \\ & & u_{nn} \end{pmatrix}$$

e riscriviamo $A = LU$ come $A = LDD^{-1}U$. La matrice $D^{-1}U$:

- è **triangolare superiore** perché U è triangolare superiore;
- $d_{ii}^{-1} = \frac{1}{u_{ii}} \Rightarrow D^{-1}U$ ha **diagonale unitaria**;

Dimostriamo che $D^{-1}U = L^T$. Partiamo dalla simmetrica di A:

$$A = LDD^{-1}U \Rightarrow A^T = (LDD^{-1}U)^T = (D^{-1}U)^T D^T L^T = (D^{-1}U)DL^T$$

Riscriviamo l'uguaglianza $A = A^T$ utilizzando L, D, U:

$$A = A^T \Leftrightarrow LDD^{-1}U = (D^{-1}U)^T DL^T \Rightarrow \boxed{DD^{-1}U(L^T)^{-1} = L^{-1}(D^{-1}U)^T D}$$

Analizziamo quest'uguaglianza:

- a sinistra c'è una **matrice triangolare superiore**;
- a destra abbiamo una **matrice triangolare inferiore**;

siccome due matrici sono uguali solo se lo sono coefficiente per coefficiente, significa che le due matrici in realtà sono delle **matrici diagonali**, in particolare delle **matrici identità** perché sia $D^{-1}U$ che L^T hanno diagonale unitaria.

Quindi:

$$D^{-1}U(L^T)^{-1} = I \xrightarrow{\text{tolgo } (L^T)^{-1} \text{ a sinistra}} \boxed{D^{-1}U = L^T}$$

Metodo di pavimentazione

La dimostrazione del teorema $A = LDL^T$ implica che per fattorizzare A occorra la matrice U, in quanto i coefficienti diagonali di D sono quelli di U.

Il **metodo di pavimentazione** evita di calcolare la matrice U, andando a **dimezzare** il costo computazionale dell'algoritmo.

Eseguendo il prodotto matriciale $A = LDL^T$ si ottiene la relazione:

$$a_{ij} = d_{jj}l_{ij} + \sum_{k=1}^{j-1} l_{ik}d_{kk}l_{jk} \qquad j = 1, \dots, n,\ i = j, \dots, n$$

Scomponendo la formula per calcolare d_{jj} ed l_{ij} (le effettive incognite del problema) si ottiene:

$$d_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk} \qquad i = j$$

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}d_{kk}l_{jk}}{d_{jj}} \qquad i > j$$

Dalle formule si nota che il prodotto $l_{jk}d_{kk}$ si esegue sia per calcolare d_{jj} che per calcolare l_{ij}. Si introduce una nuova matrice P per memorizzarlo, in modo da eseguire il prodotto una sola volta:

$$p_{jk} = l_{jk}d_{kk} \qquad \begin{matrix} j = 1, \dots, n \\ k = 1, \dots, j-1 \end{matrix}$$

```
function [L, D] = fattorizzazione_ldl(A)
    if (~issymmetric(A))
        error("La matrice A non è simmetrica. Impossibile procedere con la fattorizzazione.")
    end

    n = length(A);
    L = eye(n);
    D = zeros(n);
    P = zeros(n);

    for j = 1 : n
        P(j, 1 : j - 1) = L(j, 1 : j - 1) * D(1 : j - 1, 1 : j - 1);
        D(j, j) = A(j, j) - sum(L(j, 1 : j - 1) .* P(j, 1 : j - 1));

        if (abs(D(j, j)) < eps)
            error("Elemento diagonale nullo. Fattorizzazione non esegubiile.")
        end

        for i = j + 1 : n
            L(i, j) = (A(i, j) - sum(L(i, 1 : j - 1) .* P(j, 1 : j - 1))) / D(j, j);
        end
    end
```

Fattorizzazione di Cholesky

Si applica a **matrici simmetriche definite positive**. Queste matrici hanno 3 proprietà:

- $x^T Ax \geq 0 \forall x \in R^n$ e $x^T Ax = 0 \Leftrightarrow x = 0$;
- tutti i minori principali sono > 0 (quindi soddisfano le ipotesi della fattorizzazione LDL^T);
- tutti gli autovalori sono reali e positivi

Teorema

Una matrice simmetrica A è **definita positiva** se e solo se esiste una **matrice triangolare inferiore** $\mathcal{L}$ con elementi diagonali **positivi** tale che

$$A = \mathcal{L}\mathcal{L}^{\mathcal{T}}$$

Dimostrazione 1: A simmetrica definita positiva ⇒ esiste la matrice $\mathcal{L}$

Dal teorema di Gauss e dalla proprietà 1 delle matrici simmetriche definite positive:

$$A = LDL^T \Rightarrow x^T LDL^T x > 0 \quad \forall x \neq 0$$

Sia $y = L^T x$. L è invertibile per il teorema di Gauss ed $x \neq 0$ per ipotesi, quindi anche $y \neq 0$:

$$y^T Dy > 0 \quad \forall y \neq 0$$

Scriviamo D come prodotto di due matrici $D = \Delta\Delta$, dove Δ è la matrice diagonale che ha come coefficienti le **radici quadrate** dei coefficienti di D:

$$\Delta = \begin{pmatrix} \sqrt{d_{11}} & & \\ & \ddots & \\ & & \sqrt{d_{nn}} \end{pmatrix}$$

e riscriviamo $A = LDL^T$ come $L\Delta\Delta L^T$. Raggruppando i termini:

$$A = (L\Delta)(\Delta L^T) \Rightarrow A = (L\Delta)(L\Delta)^T$$

Per avere la tesi è sufficiente chiamare $\mathcal{L} = L\Delta$.

Dimostrazione 2: esiste $\mathcal{L} \Rightarrow A$ è simmetrica definita positiva

Dobbiamo dimostrare che $x^T Ax = 0 \Leftrightarrow x = 0$.

Sia $y = \mathcal{L}^{\mathcal{T}} x$:

$$x^T \mathcal{L}\mathcal{L}^T x = y^T y = \|y\|_2^2 = 0 \Leftrightarrow y = 0$$

Siccome $y = \mathcal{L}^{\mathcal{T}} x$, con $\mathcal{L}^{\mathcal{T}} \neq 0$ per ipotesi, l'uguaglianza è vera se e solo se $x = 0$, dunque il teorema è dimostrato.

Algoritmo

Dal teorema sappiamo che $\mathcal{L} = L$, ovvero:

$$\ell_{jj} = \sqrt{d_{jj}} \qquad \ell_{jk} = l_{jk\sqrt{d_{kk}}} \qquad \forall k = 1, \dots, j-1$$

Perciò dalle regole di pavimentazione viste per la fattorizzazione LDL^T:

$$\ell_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2} \qquad i = j$$

$$\ell_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}}{\ell_{jj}} \qquad i > j$$

Le sommatorie possono essere ottimizzate scrivendole come prodotti matriciali (più efficienti).

```
function [L] = fattorizzazione_cholesky_ottimizzata(A)
    if (~issymmetric(A))
        error("La matrice A non è simmetrica. Impossibile procedere con la fattorizzazione.")
    end

    n = length(A);

    for j = 1 : n
        A(j, j) = sqrt(A(j, j) - (A(j, 1 : j - 1) * A(j, 1 : j - 1)'));

        if (abs(A(j, j)) < eps)
            error("L ha un valore negativo. Impossibile procedere con la fattorizzazione.");
        end

        for i = j + 1 : n
            A(i, j) = (A(i, j) - (A(i, 1 : j - 1) * A(j, 1 : j - 1)')) / A(j, j);

            if (abs(A(i, j)) < eps)
                error("L ha un valore negativo. Impossibile procedere con la fattorizzazione.");
            end
        end
    end

    L = tril(A);
```

Stabilità

La fattorizzazione di Cholesky è la più stabile di tutte perché, essendo A simmetrica definita positiva, ogni pivot è il valore massimo all'interno della sua colonna. Inoltre gli elementi del triangolo inferiore hanno un limite superiore che **non dipende** dalle dimensioni di A.

Fattorizzazione QR

Se A è una matrice nonsingolare, allora esistono una **matrice ortogonale** Q ed una **matrice triangolare superiore** R tali che $A = QR$.

Trasformazioni di Householder

Definizione

Dato un vettore $v \neq 0$, si chiama **trasformazione elementare di Householder** associata a v la matrice

$$U = I - \frac{1}{\alpha} vv^T \qquad \alpha = \frac{\|v\|_2^2}{2}$$

La matrice U:

- è **simmetrica**;
- è **ortogonale**;

Costo computazionale del prodotto matrice-vettore

Se U è la trasformazione di Householder associata al vettore $v \neq 0$ ed y è un altro vettore, per calcolare il prodotto $z = Uy$ non è necessario calcolare esplicitamente la matrice U, se si usa in modo furbo la proprietà associativa:

$$z = Uy = \left(I - \frac{1}{\alpha} vv^T\right) y = y - \frac{1}{\alpha} vv^T y = y - \frac{1}{\alpha} v \underbrace{(v^T y)}_{\text{scalare}}$$

Il prodotto $v^T y$ restituisce uno scalare. Se si esegue per primo, anziché eseguire vv^T, il prodotto Uy ha un **costo lineare** anziché quadratico.

Proprietà σ

È una proprietà delle trasformazioni di Householder che permette di **azzerare** i coefficienti di un vettore colonna ad eccezione del primo.

Proprietà

Siano z un vettore $\neq 0$ ed U la trasformazione di Householder associata al vettore $v = z + \sigma e_1$ e $\sigma = \|z\|$. Allora

$$Uz = -\sigma e_1 = \begin{pmatrix} -\sigma \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Dimostrazione

Scriviamo esplicitamente l'α della trasformazione di Householder, utilizzando $v = z + \sigma e_1$:

$$\alpha = \frac{v^T v}{2} = \frac{(z + \sigma e_1)^T (z + \sigma e_1)}{2} = \sigma^2 + \sigma z_1$$

Scriviamo esplicitamente il prodotto Uz:

$$Uz = \left[I - \frac{(z + \sigma e_1)(z + \sigma e_1)^T}{\sigma^2 + \sigma z_1} \right] z$$

Risolvendo questo prodotto si ottiene esattamente $-\sigma e_1$.

Caso particolare

$$z = \begin{pmatrix} -z_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad z_1 > 0 \Rightarrow \sigma = \|z\| = -z_1$$

In questo caso $v = z + \sigma e_1 = 0$, dunque la proprietà σ non è applicabile.

Per aggirare questo problema semplicemente si sceglie $U = I$, anziché prendere la trasformazione di Householder:

$$Uz = Iz = z = \begin{pmatrix} -z_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = -z_1 e_1 = -\sigma e_1$$

Fattorizzazione QR mediante trasformazioni di Householder

La matrice A la si pensa come "concatenazione" di n vettori colonna:

$$A = (a_1 \quad a_2 \quad \dots \quad a_n) \qquad a_i = \begin{pmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{ni} \end{pmatrix}$$

Si applicano le trasformazioni elementari di Householder alle varie colonne di A fino a renderla una **matrice triangolare superiore**. La matrice finale sarà la matrice R.

Passo 1:

- $z = a_1 \Rightarrow v_1 = a_1 + \sigma_1 e_1$ (con $\sigma_1 = \|a_1\|$);
- $U =$ trasf. Elementare di Householder associata a v_1;
- $A_2 = U_1 A =$ si moltiplicano tutte le colonne di A per la matrice U_1

$$A_2 = \begin{pmatrix} -\sigma_1 & a_{12}^{(2)} & \dots & a_{1n}^{(n)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{2n}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix}$$

Passo 2:

Ora si deve applicare la proprietà σ alla 2° colonna di A, ma solo a partire dal 2° coefficiente:

$$\tilde{A}_2 = \begin{pmatrix} a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ a_{32}^{(2)} & \dots & a_{32}^{(2)} \\ \vdots & \ddots & \vdots \\ a_{2n}^{(2)} & \dots & a_{nn}^{(2)} \end{pmatrix}$$

- $z = a_2^{(2)} \Rightarrow v_2 = a_2^{(2)} + \sigma_2 e_1$ (con $\sigma_2 = \left\| a_2^{(2)} \right\|$;
- $\widetilde{U}_2 =$ trasf. Elementare di Householder associata a v_2;

$\widetilde{U}_2$ è una matrice $(n-1) \times (n-1)$, perché stiamo considerando la matrice $\tilde{A}_2$, che è di dimensioni $(n-1) \times (n-1)$. Per poter eseguire il prodotto con A_2 ed ottenere quindi A_3, bisogna definire una matrice U_2 di dimensioni $n \times n$, fatta come la matrice $\widetilde{U}_2$. Le righe e colonne che le mancano per diventare una $n \times n$ vengono prese dalla matrice identità.

Al passo k la matrice A_k sarà quindi una **triangolare superiore** fino alla $k-1$-esima colonna.

Come con Gauss, anche qui si ottiene una sequenza di prodotti matriciali:

$$U_{n-1} U_{n-2} \dots U_2 U_1 A = R$$

Il risultato $A = QR$ si ottiene chiamando $Q^T = U_{n-1} \dots U_2 U_1$.

Nota sulle matrici rettangolari

La fattorizzazione QR si può applicare anche quando A è **rettangolare**:

$$A = Q\begin{pmatrix} R \\ 0 \end{pmatrix}$$

Dove $Q, R \in \mathbb{R}^{n\times n}$ e 0 è una "matrice resto" di $m - n$ righe che si "appende" in fondo ad R per renderla di dimensioni $m \times n$.

Codice Matlab

```
% Fattorizzazione QR che non calcola la U ad ogni passaggio.

function [Q, A] = fattorizzazione_qr_ottimizzata(A)
    n = length(A);
    Q = eye(n);
    I = eye(n);

    for i = 1 : n - 1
        a = A(i : n, i);
        sigma = norm(a, 2);

        % controlla se la matrice è invertibile
        if (sigma < eps)
            error("Fattorizzazione non eseguibile.")
        end

        if (sigma ~= -A(i, i))
            e = I(i : n, i);
            v = a + sigma * e;
            alpha = 0.5 * power(norm(v, 2), 2);

            % Per la A devo moltiplicare solo le ultime (n - i) colonne,
            % perchè quelle prima sono già a posto.
            for j = i : n
                A(i : n, j) = A(i : n, j) - v * (v' * A(i : n, j)) / alpha;
            end

            % Per la Q invece devo moltiplicare TUTTE le sue colonne.
            for j = 1 : n
                Q(i : n, j) = Q(i : n, j) - v * (v' * Q(i : n, j)) / alpha;
            end
        end
    end

    Q = Q';
```

Confronto tra gli algoritmi di fattorizzazione

Algoritmo	Ipotesi	Complessità	Stabilità
Gauss con pivoting parziale	A nonsingolare	$O\left(\frac{n^3}{3}\right)$	Debole
LDL^T	A simmetrica con minori principali $\neq 0$	$O\left(\frac{n^3}{6}\right)$	Debole
Cholesky	A simmetrica definita positiva	$O\left(\frac{n^3}{6}\right)$	Forte
QR	Colonne di A linearmente indipendenti	$O\left(\frac{2n^3}{3}\right)$	Debole (ma migliore di Gauss)

Metodi iterativi

Anziché cercare di fattorizzare A, si costruisce una **successione di vettori** $\{x^{(k)}\}_{k\in\mathbb{N}}$ che per $n \to \infty$ converge alla soluzione x^* del sistema.

Definizione di convergenza per vettori

Si dice che una successione di vettori $\{x^{(k)}\}_{k\in\mathbb{N}}$ converge ad un vettore x^* se, per una qualche norma vettoriale:

$$\lim_{k\to\infty} \left\| x^{(k)} - x^* \right\| = 0$$

In questo caso si scrive che

$$\lim_{k\to\infty} x^{(k)} = x^*$$

Costruzione del metodo iterativo

Si parte dal problema $Ax = b$, poi si somma il termine Mx ad entrambi i membri dell'uguaglianza:

$$Ax = b \Rightarrow Mx + Ax = Mx + b \Rightarrow \boxed{x = (I - M^{-1}A)x + M^{-1}b}$$

Se chiamiamo $G = (I - M^{-1}A)$ (**matrice di convergenza**) e $c = M^{-1}b$ (**matrice del metodo**), si ottiene:

$$x = Gx + c$$

Un metodo iterativo si dice **convergente** se **per ogni scelta** di $x^{(0)}$ la successione generata converge ad x^*.

La convergenza di un metodo iterativo dipende **solo** dalla scelta di M.

Condizioni per la convergenza di un metodo iterativo

Teorema (condizione sufficiente)

Se $\|G\| < 1$, allora il metodo iterativo

$$x^{(k+1)} = Gx^{(k)} + c \qquad k = 0,1, \ldots.$$

è convergente.

Dimostrazione

Definiamo il vettore $e^{(k)} = x^{(k)} - x^*$. La convergenza si ha se e solo se $\lim_{k\to\infty} e^{(k)} = 0$.

Sapendo che $x^{(k)}$ è soluzione di $Gx^{(k-1)} + c$ ed x^* è soluzione di $Gx^* + c$, possiamo scrivere:

$$e^{(k)} = x^{(k)} - x^* = Gx^{(k-1)} + c - Gx^* - c = G\left(x^{(k-1)} - x^*\right)$$

$x^{(k-1)} - x^* = e^{(k-1)}$, quindi:

$$e^{(k)} = Ge^{(k-1)}$$

$e^{(k-1)} = G(x^{(k-2)} - x^* \Rightarrow e^{(k)} = GGe^{(k-2)} = G^2e^{(k-2)}$. Andando avanti fino ad ottenere $e^{(k-k=0)}$:

$$e^{(k)} = G^k e^{(0)}$$

Passiamo ora al limite:

$$\lim_{k\to\infty} x^{(k)} - x^* = \lim_{k\to\infty} e^{(k)} = \lim_{k\to\infty} G^k e^{(0)}$$

E poi alle norme, ricordando la **proprietà submoltiplicativa**:

$$\lim_{k\to\infty} \left\|x^{(k)} - x^*\right\| \le \lim_{k\to\infty} \left\|G^k\right\| \left\|e^{(0)}\right\| \le \lim_{k\to\infty} \|G\|^k \left\|e^{(0)}\right\| = \left\|e^{(0)}\right\| \cdot \left(\lim_{k\to\infty} \|G\|^k\right)$$

Se $\|G\| < 1$, allora $\|G\|^k = 0$ per $k \to \infty$, dunque si ha la convergenza per ogni punto iniziale.

Teorema (condizione necessaria e sufficiente)

Il metodo iterativo

$$x^{(k+1)} = Gx^{(k)} + c \qquad k = 0,1,2, \ldots$$

è convergente se e solo se

$$\rho(G) < 1$$

Velocità di convergenza

$$\left\|e^{(k)}\right\| \simeq \rho(G)^k$$

Più $\rho(G)$ è piccolo e più è veloce la convergenza di $x^{(k)}$ ad x^*.

Criteri d'arresto

Non essendo possibile calcolare infiniti valori di $x^{(k)}$, si devono quindi trovare dei **criteri d'arresto** che permettono di terminare le iterazioni con la garanzia che l'ultima iterata approssimi x^* entro una certa tolleranza ϵ fissata a priori.

Errore assoluto ed errore relativo

Proposizione

$$\left\|x^{(k+1)} - x^{(k)}\right\| \leq \tau \Rightarrow \left\|x^{(k)} - x^*\right\| \leq \epsilon \qquad \begin{matrix} \tau > 0 \\ \epsilon = \tau\|(G-I)^{-1}\| \end{matrix}$$

Dimostrazione

Se x^* è la soluzione del sistema, allora $x^* = Gx^* + c \Rightarrow c = x^* - Gx^*$, quindi:

$$\begin{aligned} x^{(k+1)} - x^{(k)} &= Gx^{(k)} + c - x^{(k)} = Gx^{(k)} - Gx^* + x^* - x^{(k)} \\ &= G\left(x^{(k)} - x^*\right) - \left(x^{(k)} - x^*\right) = (G-I)\left(x^{(k)} - x^*\right) \end{aligned}$$

Se il metodo è convergente, allora la matrice $(G-I)$ è invertibile, quindi risolvendo l'ultima uguaglianza:

$$\left(x^{(k)} - x^*\right) = (G-I)^{-1}\left(x^{(k+1)} - x^{(k)}\right)$$

Passando alle norme:

$$\left\|x^{(k)} - x^*\right\| \leq \|(G-I)^{-1}\|\left\|x^{(k+1)} - x^{(k)}\right\|$$

Se assumiamo che $x^{(k+1)}$ ed x^* abbiano lo stesso ordine di grandezza, dividendo entrambi i membri per $\left\|x^{(k+1)}\right\|$ si ottiene un limite superiore per l'**errore relativo**:

$$\frac{\left\|x^{(k)} - x^*\right\|}{\left\|x^{(k+1)}\right\|} \leq \|(G-I)^{-1}\| \frac{\left\|x^{(k+1)} - x^{(k)}\right\|}{\left\|x^{(k+1)}\right\|}$$

Se $\|(G-I)^{-1}\|$ non è troppo grande, allora:

$$\frac{\left\|x^{(k+1)} - x^{(k)}\right\|}{\left\|x^{(k+1)}\right\|} \leq \tau \Rightarrow \frac{\left\|x^{(k)} - x^*\right\|}{\left\|x^*\right\|} \simeq \tau$$

Criterio del residuo

Il **residuo** all'iterata k-esima è definito come:

$$r^{(k)} = b - Ax^{(k)}$$

Il residuo è quindi una quantità che si annulla quando $x^{(k)}$ è la soluzione del sistema.

Proposizione

$$\frac{\left\|r^{(k)}\right\|}{\|b\|} \leq \tau \Rightarrow \frac{\left\|x^{(k)} - x^*\right\|}{\|x^*\|} \leq \kappa(A)\tau \qquad \tau > 0$$

Dimostrazione

Dall'analisi del **condizionamento dei sistemi lineari** sappiamo che

$$\frac{\left\|x^{(k)} - x^*\right\|}{\|x^*\|} \leq \kappa(A) \frac{\left\|r^{(k)}\right\|}{\|b\|}$$

Quindi se A è ben condizionata e vale il criterio del residuo, allora

$$\frac{\left\|x^{(k)} - x^*\right\|}{\|x^*\|} \simeq \tau$$

Complessità computazionale

Per un metodo iterativo non è possibile conoscere a priori il numero di iterazioni che verranno fatte, si può quindi stimare solo il costo di una **singola** iterazione. Questo è pari al costo del **prodotto matrice-vettore**, ovvero $\simeq O(n^2)$.

I metodi iterativi possono essere una valida alternativa alla fattorizzazione se:

- l'accuratezza con cui si vuole approssimare la soluzione è abbastanza bassa da richiedere poche iterazioni;
- le matrici A ed M hanno una struttura particolare per cui il costo del prodotto matrice-vettore è molto inferiore ad n^2

Scelta della matrice del metodo

La scelta ideale, ma non pratica, sarebbe quella $M = A$. Questa scelta permette di ottenere la soluzione del sistema con un'unica iterazione, ma significa calcolare l'inversa di A.

Decomposizione di A

Si pensa alla matrice A come risultato della differenza di due matrici M, N:

$$A = M - N$$

per cui il metodo iterativo diventa

$$Mx^{(k+1)} = Nx^{(k)} + b$$

$x^{(k+1)}$ è quindi la soluzione del sistema $Mv = p$, con $p = Nx^{(k)} + b$.

La matrice A può essere scomposta in 3 matrici D, E, F tali che $A = D - E - F$, dove:

- D è una matrice diagonale;
- E è una matrice triangolare inferiore con i coefficienti cambiati di segno;
- F è una matrice triangolare superiore con i coefficienti cambiati di segno

Metodo di Jacobi

Prevede di scegliere $M = D$ e quindi $N = E + F$:

$$Dx^{(k+1)} = (E + F)x^{(k)} + b$$

```
function [x_curr, k] = jacobi(x_start, A, b, tau, Kmax)

    M = diag(diag(A));
    N = M - A;
    x_curr = x_start;

    for k = 1 : Kmax
        x_next = M \ (N * x_curr + b);
        r = b - A * x_next;

        if (norm(x_next - x_curr) / norm(x_next) < tau) && (norm(r) / norm(b) <
tau)
            break
        else
            x_curr = x_next;
        end
    end

    if k == Kmax
        fprintf("Attenzione: convergenza non raggiunta dopo %d passi.", Kmax);
    end
```

Metodo di Gauss-Seidel

Prevede di scegliere $M = D - E$ e quindi $N = F$:

$$(D - E)x^{(k+1)} = Fx^{(k)} + b$$

```
function [x_curr, k] = gauss_seidel(x_start, A, b, tau, Kmax)

    M = tril(A);
    N = -triu(A, 1);
    x_curr = x_start;

    for k = 1 : Kmax
        x_next = M \ (N * x_curr + b);
        r = b - A * x_next;

        if (norm(x_next - x_curr) / norm(x_next) < tau) && (norm(r) / norm(b) <
tau)
            break
        else
            x_curr = x_next;
        end
    end

    if k == Kmax
        fprintf("Attenzione: convergenza non raggiunta dopo %d passi.\n", Kmax);
    end
```

Teorema

Se A è strettamente diagonale dominante, allora entrambi i metodi di Jacobi e Gauss-Seidel convergono.

Equazioni nonlineari

Data una funzione f definita su un intervallo $[a, b]$, trovare gli **zeri** di f significa risolvere l'equazione nonlineare $f(x) = 0$.

Teorema del valore medio

Se $f: [a, b] \to \mathbb{R}$ è **continua** in $[a, b]$ ed $f(a)f(b) < 0$, allora esiste **almeno** un valore $\overline{x} \in [a, b]$ tale che $f(\overline{x}) = 0$.

Condizionamento del problema

Sia x_* una radice di f. Consideriamo un punto $\tilde{x}$ soluzione del problema perturbato $f(x) = \delta$.

Supponendo che f sia derivabile:

$$f'(x_*) = \lim_{x \to x_*} \frac{f(x) - f(x_*)}{x - x_*}$$

In un intorno di x_* si ha che:

$$\frac{f(x) - f(x_*)}{x - x_*} \simeq f'(x_*) \Rightarrow x - x_* \simeq \frac{f(x) - f(x_*)}{f'(x_*)}$$

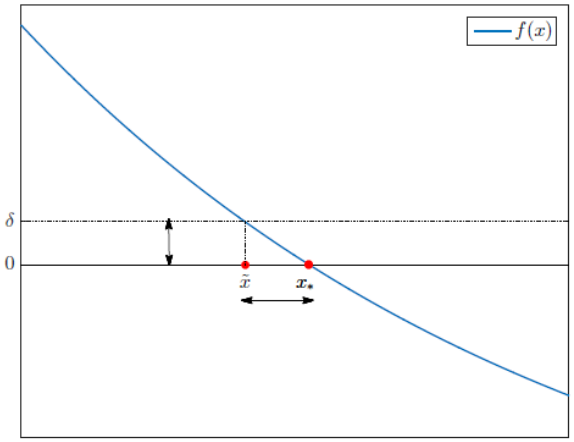
Passando al valore assoluto:

$$|x_* - \tilde{x}| \simeq \frac{|f(x_*) - f(\tilde{x})|}{|f'(x_*)|} = \frac{\delta}{|f'(x_*)|}$$

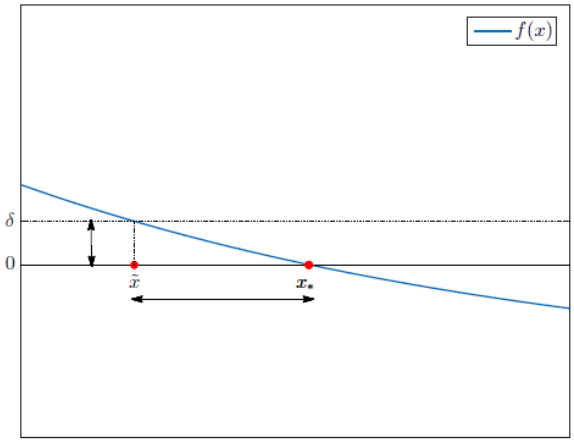
quindi il condizionamento del problema è inversamente proporzionale ad $|f'(x_*)|$.

Dal punto di vista grafico, il problema è ben condizionato se scelto un intervallo δ sull'asse y il corrispondente intervallo sull'asse x ha circa la stessa ampiezza:

Problema ben condizionato



Problema mal condizionato



Metodo di bisezione

Teorema

La successione dei punti medi $\{c_k\}_{k\in\mathbb{N}}$ generata dal metodo di bisezione converge ad una radice di f nell'intervallo $[a, b]$ e:

$$|c_k - x_*| \leq \frac{b-a}{2^k} \quad \Rightarrow \quad |c_k - x_*| = O\left(\frac{1}{2^k}\right)$$

Con questo teorema sappiamo quante iterazioni sono necessarie per approssimare x_* con una tolleranza $\tau > 0$:

$$\frac{b-a}{2^k} \leq \tau \Leftrightarrow k \geq log_2 \frac{b-a}{\tau} \Rightarrow |c_k - x_*| \leq \tau$$

Formula stabile per il calcolo del punto medio

$$a + \frac{b-a}{2}$$

Costo computazionale

Dipende dalla funzione che si sta considerando.

Il costo computazionale si misura in **numero di valutazioni di funzione per iterazione**. Nel caso del metodo di bisezione, ad ogni passo viene calcolata **una volta** una funzione non lineare.

Codice Matlab

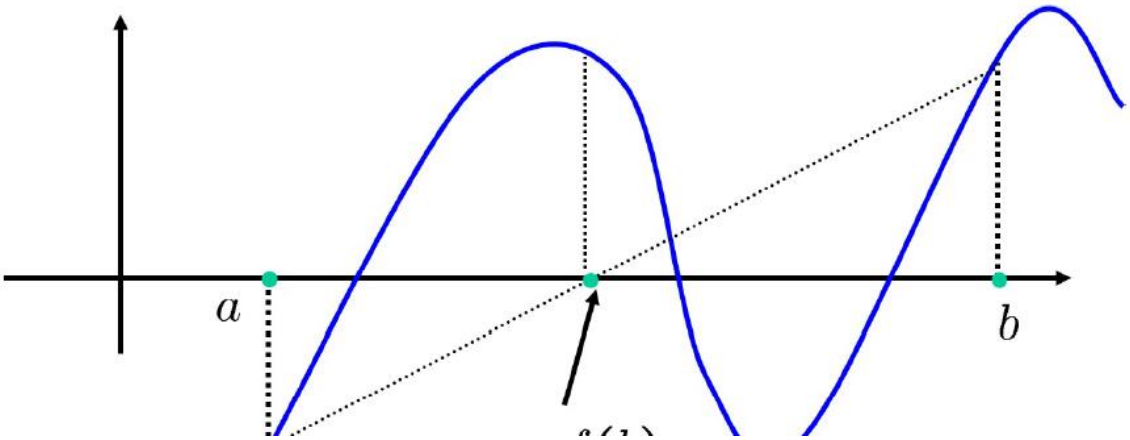
```
function c = bisez(a, b, tau, f_name)
    N = ceil(log2((b - a) / tau));
    fa = feval(f_name, a);
    fb = feval(f_name, b);

    for k = 1 : N
        c = a + ((b - a) / 2);
        fc = feval(f_name, c);

        if fc == 0
            break
        elseif fc * fb < 0
            a = c;
            fa = fc;
        else
            b = c;
            fb = fc;
        end
    end
```

Metodo di Regula Falsi

Variante del metodo di bisezione. Anziché prendere c_k come punto medio di $[a_k, b_k]$ lo si prende come ascissa del punto d'intersezione tra l'asse *x* e la retta che passa per $\big(a_k, f(a_k)\big)$, $\big(b_k, f(b_k)\big)$.



Ordine di convergenza di una successione

Definizione

Sia $\{x_k\}_{k\in\mathbb{N}}$ una successione che converge ad un punto x_*. Si dice che la successione ha **ordine di convergenza** p se:

$$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|^p} = C \qquad p \geq 1,\ C \in \mathbb{R}$$

Dalla definizione di limite, segue che:

$$|x_{k+1} - x_*| \leq C|x_k - x_*|^p$$

perciò più p è grande, più è grande la **riduzione dell'errore** tra x_{k+1} ed x_k.

Il metodo di bisezione (e quello di Regula Falsi) sono di ordine 1. Hanno una bassa complessità computazionale ma sono anche lenti nel convergere alla soluzione.

Metodo di Newton

L'iterata x_{k+1} viene calcolata come l'intersezione tra l'asse *x* e la **retta tangente** al grafico di f nel punto $\big(x_k, f(x_k)\big)$:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Ordine del metodo di Newton

Il metodo di Newton è di ordine 2. È più veloce nel convergere rispetto al metodo di bisezione, ma ha anche un costo computazionale più alto (bisogna valutare 2 funzioni ad ogni iterazione).

Criteri d'arresto

errore relativo	residuo
$\frac{\lvert x_{k+1}x_k \rvert}{\lvert x_{k+1} \rvert} \leq \tau$	$\lvert f(x_k) \rvert \leq \tau$

Interpolazione

Dati n punti (x_i, y_i), con $x_i \in [a, b]$, il problema dell'interpolazione consiste nel costruire una funzione $g: [a, b] \to \mathbb{R}$ il cui grafico passa per tutti i punti (x_i, y_i), ossia soddisfa le **condizioni di interpolazione**:

$$g(x_i) = y_i \qquad \forall i = 0, \dots, n$$

Teorema fondamentale dell'algebra

Dati $n + 1$ punti del piano esiste **un unico polinomio** di grado $\leq n$ (uno in meno rispetto al numero di punti) che li interpola.

Questo polinomio è detto **polinomio di interpolazione**.

Risoluzione tramite matrice di Vandermonde

Un generico polinomio $p_n(x)$ può essere scritto nella sua forma canonica:

$$p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$$

ed è quindi possibile scrivere le condizioni di interpolazione in un **sistema lineare** di $n + 1$ equazioni:

$$\begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_2^n = y_1 \\ \dots \\ a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n \end{cases}$$

Le incognite di questo sistema sono quindi i coefficienti $a_0, \dots, a_n$.

Il sistema si può scrivere in forma matriciale come $V\alpha = y$:

$$V = \begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ 1 & x_1 & \dots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix} \qquad \alpha = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \qquad y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

- una volta trovati i coefficienti del polinomio, si può calcolare il valore di $p_n(x)$ per un qualunque valore di x. In particolare, i coefficienti non dipendono dai nodi;
- questo metodo ha un costo computazionale elevato ($\simeq O(n^3)$), dovuto alla fattorizzazione di V;
- la matrice V è spesso molto mal condizionata

Metodo di Lagrange

Si esprime p_n come **combinazione lineare** di altri polinomi di grado n, i quali dipendono dai nodi.

Esempio per $n + 1 = 2$:

$$p_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0} = y_0 L_0(x) + y_1 L_1(x)$$

In generale:

$$p_n = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x)$$

Per come sono fatti, i polinomi L_k hanno un'importante proprietà:

$$\begin{cases} L_k(x_j) = 1 & k = j \\ L_k(x_j) = 0 & k \neq j \end{cases}$$

In forma esplicita:

$$L_k(x) = \frac{(x - x_0)(x - x_1) \cdot \ldots \cdot (x - x_{k-1})(x - x_{k+1}) \cdot \ldots \cdot (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdot \ldots \cdot (x_k - x_{k-1})(x_k - x_{k+1}) \cdot \ldots \cdot (x_k - x_n)}$$

$$= \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i} \qquad k = 0, \ldots, n$$

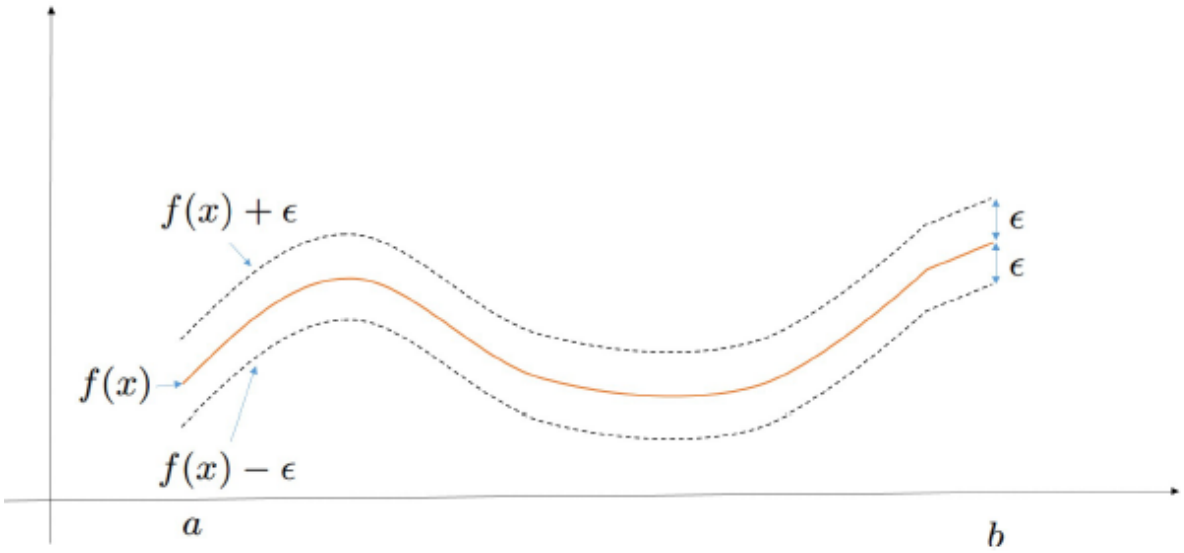
L'insieme dei polinomi $\{L_k\}_{k=0,\ldots,n}$ viene detto **base di Lagrange**. I polinomi L_k sono quindi **linearmente indipendenti**.

Analisi degli errori nell'interpolazione

Data una funzione $f: [a, b]$, la sua **norma infinito** è:

$$\|f\|_\infty = \max_{x\in[a,b]} |f(x)|$$

Se $\|f - g\| < \epsilon$ (con $\epsilon > 0$), significa che il grafico di g si trova in un "canale" di raggio ϵ centrata sul grafico di f:



Per studiare l'errore, definiamo la funzione $R_n(x) = f(x) - p_n(x)$ e studiamo $\|R_n\|_\infty$.

Teorema

Sia $f: C^{n+1}([a, b])$ con $x_i \in [a, b]$ e sia $p_n(x)$ il polinomio di interpolazione. Allora:

$$R_n(x) = \frac{\omega_{x_0,\dots,x_n}}{(n+1)!} f^{(n+1)}(\xi)$$

dove $\xi \in [a, b]$ e $\omega_{x_0,\dots,x_n} = (x - x_0)(x - x_1) \cdot \dots \cdot (x - x_n)$.

Con questo teorema si può stimare R_n senza conoscere p_n.

Studiamo ora $\|R_n\|_\infty = \max_{x\in[a,b]} |f(x) - p_n(x)|$.

Siccome $f^{(n+1)}$ ed $\omega_{x_0,\dots,x_n}$ sono funzioni continue, per il teorema di Weierstrass hanno massimo in $[a, b]$, quindi definiamo:

$$M_{n+1}^f = \max_{x\in[a,b]} \left|f^{(n+1)}(x)\right| \qquad \omega^*_{x_0,\dots,x_n} = \max_{x\in[a,b]} \left|\omega_{x_0,\dots,x_n}\right|$$

Visto che il massimo è un maggiorante, $|R_n|$ si può maggiorare come:

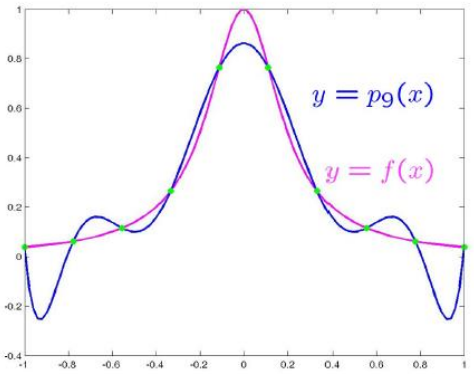
$$|R_n(x)| = \frac{\omega^*_{x_0,\dots,x_n}}{(n+1)!} f^{(n+1)}(\xi) \leq \frac{\omega^*_{x_0,\dots,x_n} \cdot M_{n+1}^f}{(n+1)!}$$

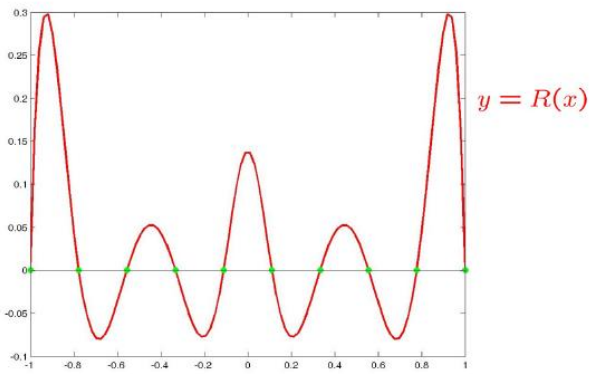
La maggiorazione vale anche per $\max|R_n(x)|$, quindi per $\|R_n\|_\infty$:

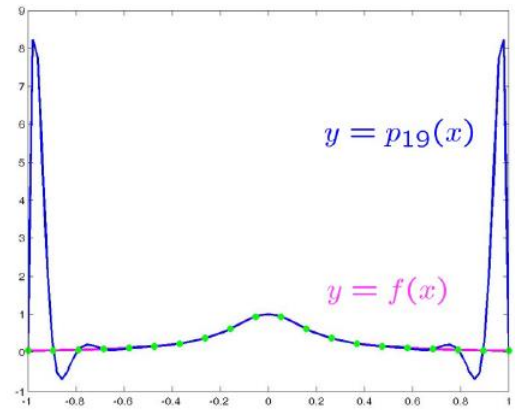
$$\|R_n\|_\infty = \max_{x\in[a,b]} |R_n(x)| \leq \frac{\omega^*_{x_0,\dots,x_n} \cdot M^f_{n+1}}{(n+1)!}$$

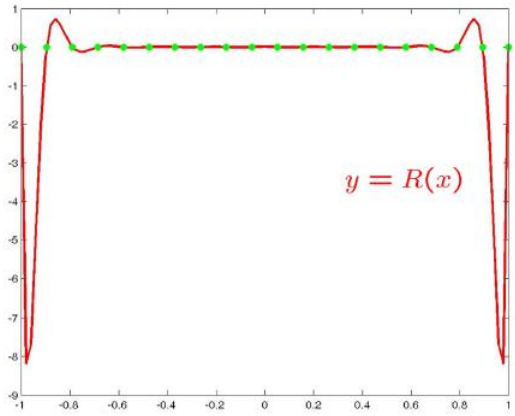
Da questa formula sembrerebbe che al crescere di n (numero di nodi) diminuisca $\|R_n\|_\infty$, e quindi l'errore. In generale, però, non è vero.

Un esempio è il **fenomeno di Runge**: al crescere di n, e quindi del grado del polinomio, l'errore **aumenta**:









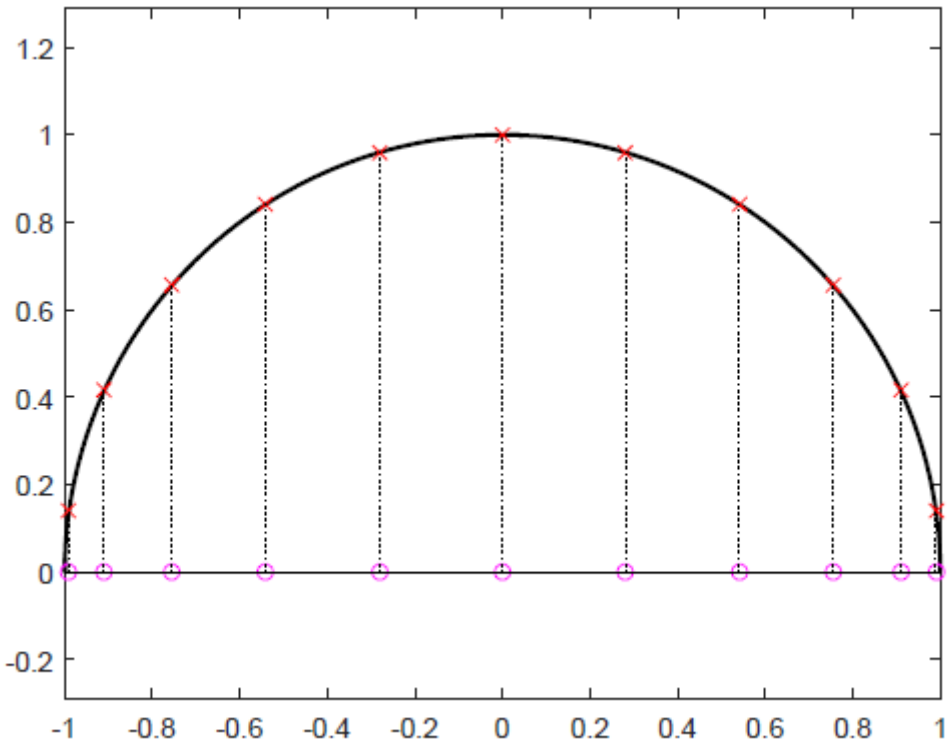
Se al centro della funzione l'approssimazione è buona, agli estremi si ha un errore molto grande. Ciò è dovuto alla scelta dei nodi: per le funzioni di Runge i nodi devono essere più densi in prossimità degli estremi della funzione.

Nodi di Chebychev

I nodi di Chebychev vanno a risolvere il problema del fenomeno di Runge.

$$x_i = cos\left(\frac{2i+1}{2(n+1)}\pi\right)$$

Nel piano si nota come questi nodi vadano a partizionare in modo uniforme una **semicirconferenza** (anziché un segmento dell'asse *x*):



È possibile dimostrare che la quantità $\omega^*_{x_0,\dots,x_n}$, calcolata quando $x_0, \dots, x_n$ sono i nodi di Chebychev, è la più piccola rispetto ad una qualunque altra scelta di nodi. Il limite superiore per $\|R_n\|$ dipende quindi solo dal massimo di $f^{(n+1)}$.

Interpolazione polinomiale a tratti

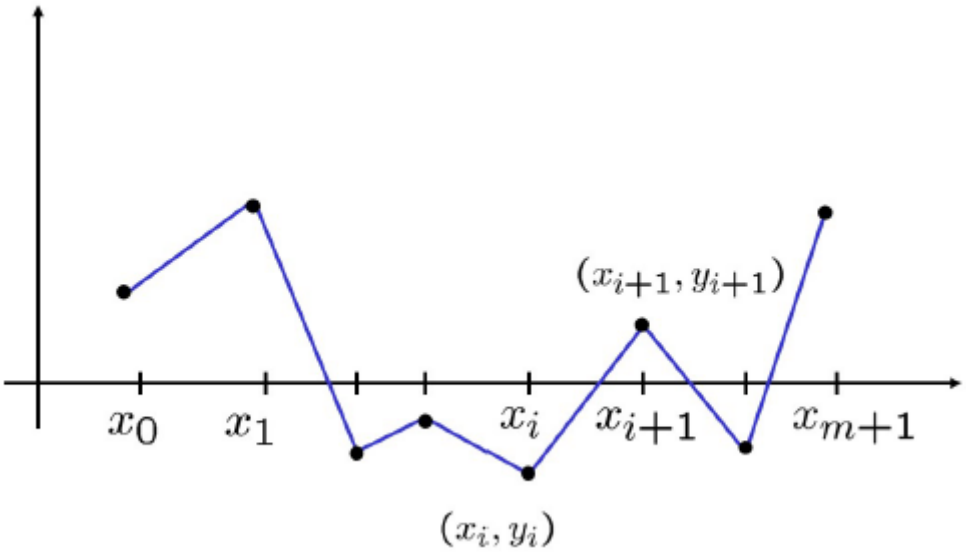
Problemi principali del polinomio di interpolazione:

- se i punti da interpolare sono molti, il grado del polinomio sarà molto alto, il cui costo per calcolarlo sarà altrettanto alto;
- può essere soggetto al fenomeno di Runge. Anche se i nodi di Chebychev risolvono questo problema, spesso non è possibile utilizzarli (i nodi sono dati in input, l'algoritmo per l'interpolazione non li può modificare)

Anziché costruire una funzione che sia un polinomio sull'interno intervallo $[a, b]$, si costruiscono tanti polinomi nei singoli intervalli formati da 2 punti.

Il caso più semplice è quello delle **funzioni lineari a tratti**: fissati $m + 2$ punti di interpolazione, si costruisce la funzione $s(x)$ su ogni intervallo $[x_i, x_{i+1}]$:

$$s(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) \qquad x \in [x_i, x_{i+1}],\ i = 0, \dots, m$$



Errore nell'interpolazione lineare a tratti

Teorema

Sia $f \in C^2([a,b])$ e siano $x_0, \dots, x_{m+1} \in [a,b]$. Indichiamo con M_2^f il **massimo** di $|f''(y)|$ in $[a,b]$ e con h l'**ampiezza massima** fra tutti gli intervalli $[x_i, x_{i+1}]$:

$$h = \max_{i \in \{0,\dots,m\}} (x_{i+1} - x_i)$$

Se $s(x)$ è il polinomio lineare a tratti tale che $s(x_i) = f(x_i)$ per $i = 0, \dots, m+1$, allora

$$\|f - s\|_\infty \leq \frac{M_2^f}{8} h^2$$

Nel caso particolare di **partizione uniforme** di un intervallo $[a,b]$ si ha che:

$$h = \frac{b-a}{m+1} \Rightarrow \|f - s\|_\infty \leq \frac{M_2^f (b-a)^2}{8(m+1)^2}$$

e quindi è vero che + nodi = - errore, dunque non si verifica mai il fenomeno di Runge.

Svantaggi dell'interpolazione lineare a tratti

La s è continua ma non derivabile.

Funzioni spline

Definizione

Sia $x_0, \dots, x_{m+1}$ una partizione di un intervallo $[a,b]$ tale che $x_0 = a$ ed $x_{m+1} = b$. Si definisce **spline di grado n relativa alla partizione** $x_0, \dots, x_{m+1}$ **ogni** funzione $s(x)$ tale che:

- ristretta all'intervallo $[x_i, x_{i+1}]$, è un polinomio $s_i(x)$ di grado $\leq n$ per $i = 0, \dots, m$;
- $s \in C^{(n-1)}([a,b])$

Osservazioni da questa definizione:

- la funzione s ristretta ad ogni intervallo $[x, x_{i+1}]$ è un polinomio di grado $\leq n$, quindi è $C^{(n-1)}$;
- dire che $s \in C^{(n-1)}([a,b])$ equivale a dire che
  $$s_i^{(k)}(x_{i+1}) = s_{i+1}^{(k)}(x_{i+1}) \qquad k = 0, \dots, n,\ i = 0, \dots, m-1$$
  ovvero, dal punto di vista grafico, i vari "pezzi" di spline si "agganciano bene" al "pezzo" precedente (in particolare $s_i'(x) = s_{i+1}'(x)$
- le funzioni lineari a tratti sono delle spline di grado 1

Le spline più usate sono quelle di grado 3, perché offrono il miglior compromesso tra approssimazione e costo per costruirle.

Per costruire le spline si prendono $m+2$ punti di interpolazione (e non m) perché le condizioni di derivabilità delle spline devono valere per gli m punti interni. In realtà sarebbe possibile scegliere anche m punti: le condizioni di derivabilità varrebbero per gli $m-2$ punti interni.

Nell'interpolazione a tratti, il grado dei polinomi è scelto arbitrariamente. Non dipende dal numero di nodi.

Dimensioni dello spazio delle funzioni spline

Considerando una spline s di grado n relativa ad una partizione di $m+2$ punti:

- ogni "pezzo" di spline s_i è un polinomio di grado n, quindi dipende da $n+1$ parametri;
- in tutto ci sono $m+1$ "pezzi" di spline

Una spline quindi dipende da $(n+1)(m+1)$ parametri. Non tutti però sono liberi, infatti la condizione di regolarità

$$s_i^{(k)}(x_{i+1}) = s_{i+1}^{(k)}(x_{i+1}) \qquad k = 0, \dots, n-1,\ i = 0, \dots, m-1$$

fissa nm parametri. Quindi in totale una spline di grado n ha

$$(n+1)(m+1) - nm = n + m + 1$$

parametri liberi. Il numero di parametri liberi viene chiamato **grado di libertà** o **dimensione dello spazio** delle spline di grado n relative ad $m+2$ punti.

La spline deve anche passare per i punti di interpolazione dati, ovvero:

$$s(x_i) = y_i \qquad i = 0, \dots, m+1$$

Queste condizioni fissano $m+2$ parametri, dunque i gradi di libertà rimanenti sono

$$n + m + n - m - 2 = n - 1$$

Osservazioni:

- il numero di parametri liberi dipende solo dal grado della spline, scelto arbitrariamente. Non dipende dal numero di nodi;
- se $n > 1$ esistono **infinite** spline che interpolano i punti (x_i, y_i). Se $n = 1$ invece ne esiste una sola. Per questo si parla di **famiglia** di spline di interpolazione.

Costruzione di una spline cubica per l'interpolazione

Dati (x_i, y_i) punti, per $i = 0, \dots, m+1$, per costruire una spline di grado 3 che li interpola si devono calcolare i coefficienti del polinomio:

$$s_i(x) = \alpha_i + \beta_i(x - x_i) + \gamma_i(x - x_i)^2 + \delta_i(x - x_i)^3 \qquad i = 0, \dots, m$$

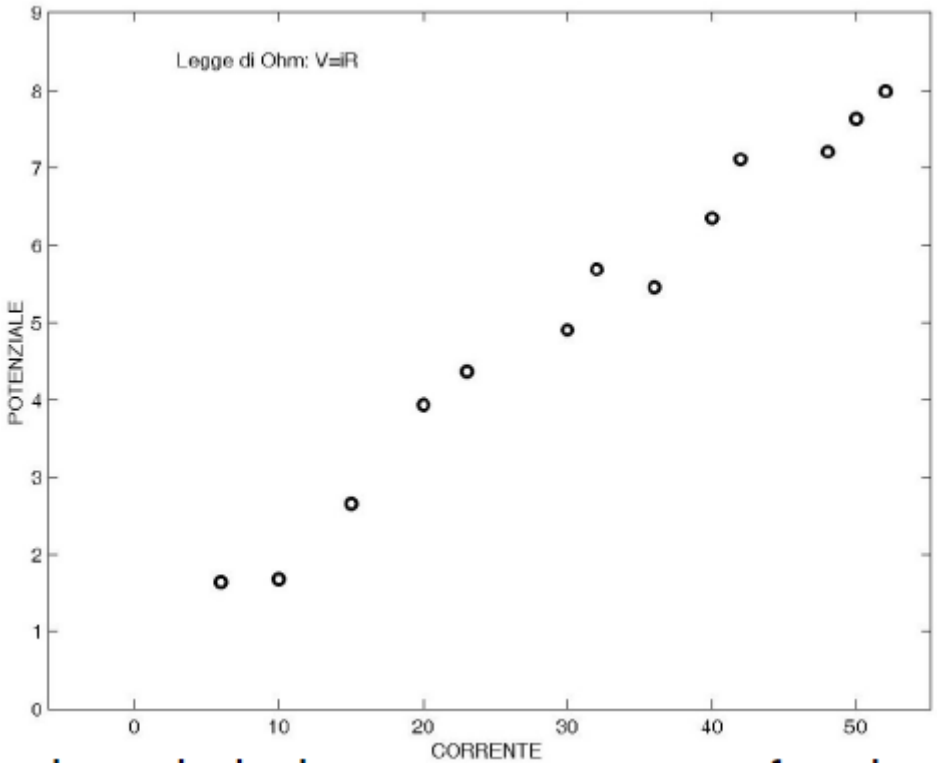
I polinomi s_i devono soddisfare le condizioni di interpolazione e di regolarità. Per trovare i coefficienti adatti a queste condizioni si risolve un **sistema lineare**.
La risoluzione di questo sistema permette di fissare tutti i parametri della spline ad eccezione degli ultimi $n-1$. Per fissare anche questi e rendere quindi la spline unica ci sono diverse tecniche:

- spline cubica naturale: si impone che $s''(x_0) = s''(x_{m+1}) = 0$ (ovvero si impone un **punto di flesso** ad entrambi gli estremi della funzione);
- spline cubica periodica: se $y_0 = y_{m+1}$, si può imporre che $s'(x_0) = s'(x_{m+1})$ e $s''(x_0) = s''(x_{m+1})$;

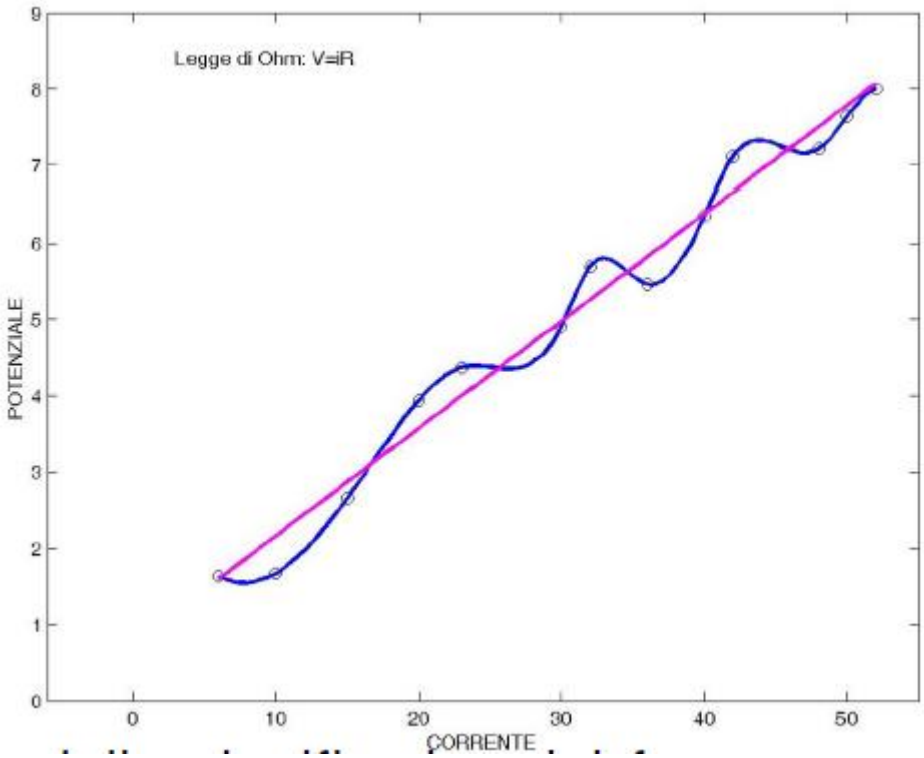
- spline cubica not-a-knot: usata da Matlab, è un polinomio di grado 3 se ristretto agli intervalli $[x_0, x_2]$, $[x_{m-1}, x_{m+1}]$.

Criterio dei minimi quadrati

Esempio: vogliamo calcolare sperimentalmente la resistenza di un filo elettrico. Facciamo variare la differenza di potenziale e misuriamo la corrente.



Sappiamo che la legge che lega potenziale è corrente è una retta: $V = iR$. Tuttavia se si interpolano questi punti tramite un polinomio non si ottiene una retta:



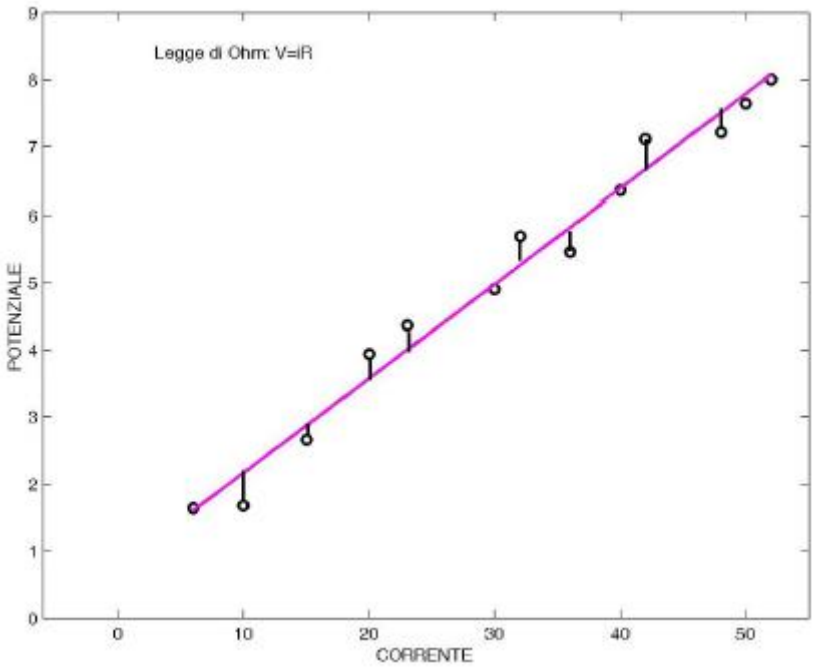
Questo è dovuto al fatto che le misurazioni che abbiamo fatto, per definizione, contengono degli **errori**.

In questo caso noi sappiamo che la legge che "lega" i dati è una retta. Tra tutte le rette di equazione $y = a_0 + a_1 x$ vogliamo trovare quella che "spiega" meglio i dati (x_i, y_i) (per $i = 0, \dots, m$).

Per trovare i coefficienti di questa retta si utilizza il **criterio dei minimi quadrati**. Si tratta di trovare i valori a_0, a_1 che **minimizzano** la funzione

$$Q(a_0, a_1) = \sum_{i=1}^{m} (a_0 + a_1 x_i - y_i)^2$$

Dal punto di vista grafico, questo significa trovare una retta la cui **distanza cumulativa** dai dati è **minima** tra tutte le rette possibili.



Il tipo di modello scelto (cioè se si vuole avere una retta, una parabola, ecc.) è fissato a priori, solitamente in base alle caratteristiche del fenomeno che si vuole osservare (es. nel caso della legge di Ohm, dalla teoria si sa già che la funzione che lega corrente e potenziale è una retta).

La **retta di regressione** può essere scritta in forma matriciale:

$$q(a_0, a_1) = \begin{pmatrix} a_0 + a_1 x_1 \\ a_0 + a_1 x_2 \\ \vdots \\ a_0 + a_1 x_m \end{pmatrix}, y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \Rightarrow Q(a_0, a_1)$$
$$= \|q(a_0, a_1) - y\|^2$$

Cambiamo scrittura per separare le incognite (i coefficienti a_0, a_1) dai dati del problema (le x_i):

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix}, \ \alpha = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$$

Riscriviamo quindi Q in funzione della matrice A e del vettore α:

$$q(a_0, a_1) = A\alpha \Rightarrow Q(a_0, a_1) = \|A\alpha - y\|^2$$

Lo stesso approccio si può estendere anche a polinomi di grado superiore:

$$Q(a_0, \dots a_{n-1}) = \sum_{i=1}^{m} (f(a_0, \dots, a_{n-1}; x_i) - y_i)^2$$

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{n-1} \end{pmatrix} \qquad \alpha = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

$$Q(a_0, \dots, a_{n-1}) = \|A\alpha - y\|^2$$

Se $n = m$, questo caso corrisponde al polinomio di interpolazione, calcolato con la matrice di Vandermonde.

Calcolo della soluzione (caso non degenere)

Avendo definito $Q(a_0, \dots, a_{n-1}) = \|A\alpha - y\|^2$, calcolare la soluzione del problema significa trovare

$$\min_{\alpha \in \mathbb{R}^n} \|A\alpha - y^2\|^2$$

In base alle caratteristiche di A si possono verificare 2 casi:

- caso **non degenere**: $A \in \mathbb{R}^{m \times n}$ con $n \leq m$ e le colonne di A sono linearmente indipendenti;
- caso **generale**: $A \in \mathbb{R}^{m \times n}$ con $n \leq m$ e $k \leq n$ colonne linearmente indipendenti

Teorema

Se $A \in \mathbb{R}^{m \times n}$ con $n \leq m$ ha rango n, allora la soluzione del problema

$$\min_{\alpha \in \mathbb{R}^n} \|A\alpha - y^2\|^2$$

è unica.

Dimostrazione

Richiamiamo alcune proprietà:

- siamo nelle ipotesi per poter applicare la **fattorizzazione QR**:
$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \Leftrightarrow Q^T A = \begin{pmatrix} R \\ 0 \end{pmatrix}$$
- se Q è ortogonale, allora $\|Qx\|^2 = x\text{^}TQ\text{^}TQx = x^Tx = \|x\|^2$;
- siano $z_1 \in \mathbb{R}^n$ e $z_2 \in \mathbb{R}^{m-n}$. Consideriamo il vettore $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$, ottenuto "concatenando" z_1 e z_2. Allora:
$$\|z\|^2 = \|z_1\|^2 + \|z_2\|^2$$

Viste queste proprietà, riscriviamo il problema come:

$$\|A\alpha - y\|^2 = \|Q^T A\alpha - Q^T y\|^2$$

Chiamiamo $\tilde{y} = Q^T y$ e consideriamolo come "concatenazione" di due vettori $\tilde{y}_1$ e $\tilde{y}_2$, quindi $\tilde{y} = \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix}$:

$$\|Q^T A\alpha - \tilde{y}\|^2 = \left\| \begin{pmatrix} R \\ 0 \end{pmatrix} \alpha - \begin{pmatrix} \tilde{y}_1 \\ \tilde{y}_2 \end{pmatrix} \right\|^2 = \|R\alpha - \tilde{y}_1\|^2 + \|\tilde{y}_2\|^2$$

Il vettore $\tilde{y}_2$ dipende esclusivamente dai dati, quindi non si può fare nulla per minimizzarlo. Bisogna quindi agire sull'addendo di destra.

$\|R\alpha - \tilde{y}_1\|$ invece è minimo quando $R\alpha = \tilde{y}_1$. Si tratta quindi di risolvere un sistema lineare!

Essendo R triangolare superiore e nonsingolare (proprietà della fattorizzazione QR), la soluzione del sistema è unica, dunque è unico anche il minimo di $\|A\alpha - y\|^2$, come volevasi dimostrare.

La dimostrazione di questo teorema ci ha dato un metodo per calcolare la soluzione:

1. si esegue la fattorizzazione QR di A;
2. si calcola il vettore $\tilde{y} = Q^T y$;
3. si estraggono le prime n componenti di $\tilde{y}$ e si salvano nel vettore $\tilde{y}_1$;
4. si risolve il sistema triangolare superiore $R\alpha = \tilde{y}_1$

Matlab

Funzioni principali

- $cond(A)$: calcola il **numero di condizionamento** della matrice A;
- $condest(A)$: calcola una **stima** del **numero di condizionamento** della matrice A;
- $plot(x, y)$: disegna i punti contenuti nei vettori x ed y su un grafico. Di default questi punti vengono uniti tra di loro, ma aggiungendo l'opzione $'o'$ i punti vengono disegnati con un cerchietto;

Algoritmo di `mldivide`

Vedi diagramma completo qui: https://it.mathworks.com/help/matlab/ref/mldivide_full.png.

In ordine, le verifiche che fa sono:

1. se A è rettangolare, usa la fattorizzazione QR;
2. se A è triangolare o lo può diventare scambiando righe/colonne tra di loro, usa il risolutore per sistemi triangolari;
3. se A è simmetrica:
    a. se la diagonale è tutta positiva o tutta negativa, prova ad usare la fattorizzazione di Cholesky;
        i. se Cholesky non ha funzionato, usa la fattorizzazione LU;
4. se nessuna delle condizioni sopra si è verificata, usa la fattorizzazione LU