Condizionamento di un problema e stabilità dell'algoritmo risolutivo

Un problema matematico, che indichiamo con f è una descrizione precisa e senza ambiguità di un **legame** tra i dati del problema x (**input**) e i risultati corrispondenti y (**output**). In altre parole, è una funzione che trasforma i dati in risultati.

Un problema si dice **ben posto** se **la sua soluzione** soddisfa tre condizioni:

- **esiste**
- **è unica**
- **dipende** in modo continuo dai dati del problema (se i dati del problema cambiano in modo graduale, la soluzione deve cambiare in modo graduale)

Definiamo **funzione dato-risultato** o **applicazione risolvente** una funzione che associa i dati del problema alla sua soluzione.

E' importante osservare che, se un problema ammette una ed una sola soluzione, la funzione dato-risultato **è biiettiva**, cioè: ad ogni insieme di dati corrisponde una e una sola soluzione e ad ogni soluzione corrisponde un solo insieme di dati.

Per garantire che un problema sia ben posto, oltre a richiedere che la funzione dato-risultato sia biiettiva, è necessario che sia anche **continua**. In altre parole, se i dati del problema cambiano di poco, la soluzione deve cambiare di poco.

Condizionamento di un problema:

Quando un problema è ben posto (ossia ammette un'unica soluzione che dipende con continuità dai dati) si cerca di dare **una misura quantitativa di come la sua soluzione venga influenzata da una perturbazione dei dati**, se ne misura, cioè il condizionamento.

Il **condizionamento** di un problema permette di misurare **quanto** la soluzione cambia **in risposta** a un cambiamento nei dati

Il condizionamento è importante per diversi motivi:

- **Affidabilità dei risultati:** Se un problema è mal condizionato, i suoi risultati possono essere inaffidabili e sensibili a piccoli errori nei dati.
- **Interpretazione dei risultati:** Se un problema è mal condizionato, può essere difficile interpretare i suoi risultati e capire come dipendono dai dati.

Supponiamo che i dati siano affetti da una perturbazione. Indichiamo con $\tilde{x} = x + \delta x$ i dati affetti da una perturbazione δx

Come vengono propagati dal problema *f* gli errori presenti nei dati, nell'ipotesi ideale che non ci siano errori di calcolo ed indipendentemente dall'algoritmo utilizzato?

Quando a piccole perturbazioni sui dati *x*, corrispondono perturbazioni relative sul risultato *f(x)* dello stesso ordine di grandezza, il problema *y=f(x)* è detto **ben condizionato**, altrimenti è detto **mal condizionato**.

Uno stesso problema può essere mal condizionato per certi dati ma non per altri.

Sia $\frac{\|f(x)-f(\tilde{x})\|}{\|f(x)\|}$ la misura dell'errore relativo sui risultati

e $\frac{\|x-\tilde{x}\|}{\|x\|}$ la misura dell'errore relativo sui dati.

quando si studia il condizionamento del problema si vuole studiare la relazione che c'è tra di essi:

$$\frac{\|f(x) - f(\tilde{x})\|}{\|f(x)\|} \leq K \frac{\|x - \tilde{x}\|}{\|x\|}$$

K è detto indice di condizionamento quantifica l'entità con cui l'errore relativo sui dati si amplifica sull'errore relativo sui risultati del problema.

Se K è "piccolo" il problema è ben condizionato, l'errore relativo sui dati non viene amplificato sull' errore relativo sulla soluzione. Se K è "grande" il problema è ben condizionato, l' errore relativo sui dati viene amplificato in maniera incontrollata sull'errore relativo sulla soluzione.

Il condizionamento **è legato al problema numerico** e non ha alcun legame con gli errori di arrotondamento delle operazioni di macchina, né con il particolare algoritmo utilizzato.

Esempio 1: Studio del **condizionamento della valutazione di una funzione** $f: R \to R$ (differenziabile) in un punto *x*.

Supponiamo di voler calcolare il valore di una funzione *f*(*x*), ma il dato *x* a nostra disposizione è affetto da un errore. In questo caso, è importante stimare l'errore sulla funzione *f*(*x*) in base all'errore sul dato *x*.

Indichiamo con $\tilde{x} = x + \delta x$, il dato affetto da una perturbazione δx.

Consideriamo uno sviluppo in serie del primo ordine di *f(x)* in un intorno di *x*

$$f(x + \delta x) = f(x) + \delta x f'(x) + (\delta x)^2 \frac{f''(\xi)}{2} \quad , \xi \in (x, x + \delta x)$$

che possiamo riscrivere come:

$$f(\tilde{x}) = f(x) + (\tilde{x} - x) f'(x) + O((\tilde{x} - x)^2)$$

$$f(\tilde{x}) - f(x) \approx (\tilde{x} - x) f'(x)$$

$$\frac{f(\tilde{x}) - f(x)}{f(x)} \approx \frac{(\tilde{x} - x) f'(x)}{f(x)}$$

$$\left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right| \approx \left|\frac{f'(x)x}{f(x)}\right| \left|\frac{\tilde{x} - x}{x}\right|$$

Poniamo $K = \left|\frac{f'(x)x}{f(x)}\right|$, allora

$$\boldsymbol{\left|\frac{f(x + \delta x) - f(x)}{f(x)}\right| \approx K \left|\frac{\tilde{x} - x}{x}\right|}$$

$K = \left|\frac{f'(x)x}{f(x)}\right|$ **è detto indice di condizionamento del problema della valutazione di una funzione** $f: R \to R$ (differenziabile) in un punto *x*. Quantifica l'entità con cui l'errore relativo sui dati si amplifica sull'errore relativo sui risultati del problema.

Esempio:

Sia $f(x) = cos(x)$, x= 1.57079.

$cos(1.57079) = 6.32679489 \cdot 10^{-6}$

$$f'(x) = -sin(x);$$

$$K = \left|\frac{sin(x) \cdot x}{cos(x)}\right| =$$

$$= \left|\frac{\sin(1.57079) \cdot 1.57079}{cos(1.57079)}\right| = \left|\frac{0.999999999979986 \cdot 1.57079}{6.32679489 \cdot 10^{-6}}\right|$$

K= $2.48275 \cdot 10^5$

Ci aspettiamo che il problema sia mal-condizionato per x vicino a

$\frac{\pi}{2}$=(**1.57079**6326794897) o a multipli di $\frac{\pi}{2}$

Indichiamo con $\tilde{x} = x + \delta x$ il dato affetto da una perturbazione δx

x= 1.57079

$\tilde{x} = 1.57078$; $\boldsymbol{\delta x = |x - \tilde{x}| \approx 10^{-5}}$.

$$cos(x) = 6.32679489 \cdot 10^{-6}$$

$$cos(\tilde{x}) = 1.632679489 \cdot 10^{-5}$$

Stimiamo $\left|\frac{f(\tilde{x})-f(x)}{f(x)}\right|$ mediante la formula

$$\left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right| \approx \left|\frac{f'(x)x}{f(x)}\right| \left|\frac{\tilde{x} - x}{x}\right|$$

$\left|\frac{\tilde{x}-x}{x}\right| = \frac{10^{-5}}{1.57079}$=$\boldsymbol{0.636622 \ldots \cdot \ 10^{-5} = 0.636622 \ldots \cdot \ 10^{-3}}$ %

$$\left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right| \approx \mathbf{2.48275 \cdot 10^5} \frac{10^{-5}}{1.57079} = \mathbf{1.581} = \mathbf{150.81}\%$$

Ad un errore relativo sui dati percentuale del 0.00063 % corrisponde un errore relativo sulla soluzione del 150.81%.

N.B. 1) Uno stesso problema può essere mal condizionato per certi dati ma non per altri.

N.B. 2) Il condizionamento è legato al problema numerico e non ha alcun legame con gli errori di arrotondamento delle operazioni di macchina né con il particolare algoritmo utilizzato. Si tratta di una caratteristica del problema, che non dipende dal modo in cui la soluzione viene calcolata.

Esempio 2:

Calcolare gli zeri del polinomio $p(x) = (x-1)\cdot(x-2)\cdot\ldots(x-20) =$

$= x^{20} - 210x^{19} + \cdots$ i cui zeri sono **1,2,3,4,….,20**

Sia $\beta = 2, t = 30.$

Perturbiamo adesso il coefficiente di x^{19}

-210----------> -210+2^{-23}

Il polinomio diventa

$$p(x) + 2^{-23}x^{19}$$

Le radici diventano

1.00000 0000	10.09526 6145 ± 0.64350 0904i
2.00000 0000	11.79363 3881 ± 1.65232 9728i
3.00000 0000	13,99235 8137 ± 2.51883 0070i
4.00000 0000	16.73073 7466 ± 2.81262 4894i
4.99999 9928	19.50243 9400 ± 1.94033 0347i
6.00000 6944	
6.99969 7234	
8.00726 7603	
8.91725 0249	
20.84690 8101	

Esempio 3: Soluzione di un sistema lineare.

$$\begin{cases} x + y = 2 \\ 1001x + 1000y = 2001 \end{cases}$$

Le soluzioni sono x=1, y=1;

Perturbiamo il coefficiente della x dell'1%:

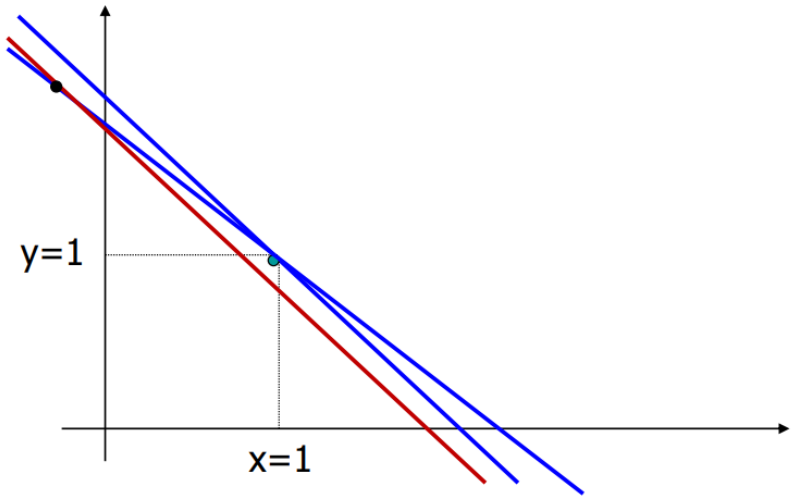
$$\begin{cases} \left(1 + \frac{1}{100}\right)x + y = 2 \\ 1001x + 1000y = 2001 \end{cases}$$

$x = -\frac{1}{9} = -0.1111$

$y = \frac{1901}{900} = 2.1122$

Errore sulla soluzione del 110%.

Interpretazione geometrica



Nel caso in cui si abbia dinanzi un problema mal condizionato, si possono seguire le seguenti strade:

1) Cambiare la formulazione del problema, per superare l’ostacolo.
2) Usare la precisione multipla nei calcoli
3) Usare tecniche di regolarizzazione, che sostituiscono al problema di partenza un problema leggermente modificato, ma che risulta ben condizionato.

Algoritmi e stabilità

Un' algoritmo, che indichiamo con Ψ, è una sequenza di operazioni di macchina che devono essere eseguite al fine di ottenere, in un numero finito di passi, da un vettore di numeri di macchina $\tilde{x}$, un output $\Psi(\tilde{x}) = \tilde{y}$.

Studiamo una nuova proprietà degli algoritmi: la **stabilità** che esprime il **comportamento dell'algoritmo considerato rispetto alla propagazione degli errori**.

Dato l'algoritmo Ψ, sequenza di operazioni di macchina, che traduce in operazioni di macchina il problema *f*, vogliamo vedere come la sequenza di operazioni di macchina eseguite sui numeri di macchina $\tilde{x}$, $\Psi(\tilde{x}) = \tilde{y}$, propaga l'errore iniziale.
La stabilità di un algoritmo valuta la reazione dell'algoritmo all'introduzione di perturbazioni nei dati iniziali.
Per analizzare, quindi, la stabilità di un algoritmo, si confronta la sua risposta con la risposta della funzione originale eseguita su dati perturbati. In questo modo, è possibile valutare l'effetto delle operazioni di macchina sul risultato finale.
Definiamo **l'Errore Algoritmico**:

$$E_{alg} = \left| \frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} \right|$$

L'errore algoritmico dipende solo da come il risultato viene calcolato, dall'algoritmo utilizzato..
Pertanto, influiscono sull'errore algoritmico:

- ▶ il numero di operazioni eseguite
- ▶ l'ordine in cui le operazioni vengono eseguite
- ▶ il tipo di operazioni eseguite

Nel calcolo di una **funzione dati-risultato** $f(x)$, l'accuratezza della soluzione numerica (di quanto si discosta la soluzione numerica, (ottenuta con un algoritmo $\Psi(x)$che esegue operazioni in aritmetica finita su numeri di macchina $\tilde{x}$) dalla risposta esatta della **funzione dati-risultato** $f(x)$)

$$E_{tot} = \left|\frac{\Psi(\tilde{x}) - f(x)}{f(x)}\right|$$

dipende sia dal condizionamento del problema che dalla stabilità algoritmica

Definito **errore inerente,** quello che deriva dalla rappresentazione finita dei numeri nel sistema di calcolo

$$\boldsymbol{E_{in} = \left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right|}$$

ed **errore algoritmico**, l'errore introdotto dalle operazioni aritmetiche in aritmetica finita durante l'esecuzione di un algoritmo.

$$\boldsymbol{E_{alg} = \left|\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\right|}$$

Verifichiamo che

$$\boldsymbol{E_{tot} \leq E_{in} + E_{alg}}$$

$$\frac{\Psi(\tilde{x}) - f(x)}{f(x)} = \frac{\Psi(\tilde{x})}{f(x)} - 1 =$$

$$= \frac{\Psi(\tilde{x})}{f(\tilde{x})}\frac{f(\tilde{x})}{f(x)} - 1.$$

$$= \frac{\Psi(\tilde{x}) - f(\tilde{x}) + f(\tilde{x})}{f(\tilde{x})} \cdot \frac{f(\tilde{x}) - f(x) + f(x)}{f(x)} - 1$$

$$= \left(\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} + 1\right)\left(\frac{f(\tilde{x}) - f(x)}{f(x)} + 1\right) \text{-1}$$

$$\frac{\Psi(\tilde{x}) - f(x)}{f(x)} = \frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\frac{f(\tilde{x}) - f(x)}{f(x)} + \frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} + \frac{f(\tilde{x}) - f(x)}{f(x)} + 1 - 1$$

$$\left|\frac{\Psi(\tilde{x}) - f(x)}{f(x)}\right| = \left|\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\frac{f(\tilde{x}) - f(x)}{f(x)} + \frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})} + \frac{f(\tilde{x}) - f(x)}{f(x)}\right| \leq$$

$$\left|\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\right|\left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right| + \left|\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\right| + \left|\frac{f(\tilde{x}) - f(x)}{f(x)}\right|$$

$$\boldsymbol{E_{tot}} \leq E_{alg} \cdot E_{in} + E_{alg} + E_{in}$$

(trascurando i prodotti degli errori relativi $E_{alg} \cdot E_{in}$,

$$\boldsymbol{E_{tot} \approx E_{in} + E_{alg}}$$

La bassa accuratezza dei risultati prodotti da un processo numerico può essere imputabile ad:
alto **mal condizionamento** intrinseco del problema
oppure
all'**instabilità** dell'algoritmo utilizzato per produrlo.

La stabilità dell'algoritmo non garantisce che il risultato calcolato sia accurato.

Infatti, per un problema mal condizionato la distinzione tra algoritmo stabile e instabile non è molto significativa in quanto l'errore totale risulta dominato dall'errore inerente. Quindi per un problema mal condizionato è opportuna, in generale, una sua riformulazione.

Si parla di **stabilità o instabilità numerica** intendendo che gli errori sui dati non sono (o sono) amplificati durante lo sviluppo dell'algoritmo:

$$\left|E_{alg}\right| \approx g(n) \cdot \varepsilon \qquad\qquad |\varepsilon| \leq u$$

n=numero di operazioni effettuare

$g(n)=c \cdot n$, $c>0$ **crescita dell'errore lineare**

$g(n)=c^n$, $c>1$, **crescita dell'errore esponenziale**.

Un algoritmo numerico è detto stabile se $g(n)$ è lineare, cioè l'errore algoritmico è dell'ordine di grandezza della precisione di macchina, instabile altrimenti.

Esempio

Valutare la funzione $f(x) = \frac{(1+x)-1}{x}$

mediante l'algoritmo $\psi(x) = ((1+x)-1)/x$ in x=1e-15

Valutiamo l'espressione in Python otteniamo: **y=1.11022302462516** invece di **y=1**

Il problema è **ben condizionato** perché:

$K = \left|\frac{f'(x)}{f(x)}x\right| = 0$ poiché $f'(x) = 0 \quad \forall x \neq 0$

Adesso calcoliamo l'errore algoritmico che commettiamo utilizzando l'algoritmo $\psi(x) = ((1+x)-1)/x$ che lavora su numeri di macchina

$$\boldsymbol{E_{alg} = \left|\frac{\Psi(\tilde{x}) - f(\tilde{x})}{f(\tilde{x})}\right|}$$

Traduciamo l'algoritmo in operazioni di macchina che lavorano su numeri di macchina:

$$\psi(x) = ((1+x)-1)/x$$

$fl(1+x) = (1+fl(x))(1+\varepsilon_s) \qquad |\varepsilon_s| \leq u$

$$\boldsymbol{fl(1+x)-1} = (fl(1+x)-1)\,(1+\varepsilon_d)$$
$$= [(1+fl(x))(1+\varepsilon_s)-1]\,(1+\varepsilon_d)$$
$$= [(1+x(1+\varepsilon_x))(1+\varepsilon_s)-1]\,(1+\varepsilon_d)$$

$|\varepsilon_x| \leq u, |\varepsilon_d| \leq u$

Per avere una notazione più semplificata

$$\boldsymbol{fl(1+x)-1} = [(1+x(1+\varepsilon\ ))(1+\varepsilon\ )-1]\,(1+\varepsilon\ )\ \ |\varepsilon\ \ | \leq u$$

$$(1+\varepsilon+x(1+2\varepsilon+\varepsilon^2)-1)(1+\varepsilon) \approx (x(1+2\varepsilon)+\ \varepsilon)(1+\varepsilon) \approx \boldsymbol{x(1+3\varepsilon)+\ \varepsilon}$$

$$\psi(\widetilde{x}) = \frac{fl((1+x)-1)}{fl(x)} \approx \frac{x(1+3\varepsilon)+\varepsilon}{x(1+\varepsilon)} \cdot (1+\varepsilon) \approx \frac{x(1+3\varepsilon)+\varepsilon}{x}$$

dove con la notazione $\tilde{x}$, ci riferiamo al fatto che l'algoritmo ha lavorato su numeri di macchina (perturbazione dei numeri reali).

Il valore in aritmetica reale di y è 1.

Calcoliamo quindi l'errore relativo

$$\left|\frac{\psi(\widetilde{x})-f(\widetilde{x})}{f(\widetilde{x})}\right| = \frac{\frac{x(1+3\varepsilon)+\varepsilon}{x}-1}{1} = \frac{x+3\varepsilon x+\varepsilon-x}{x} = 3\varepsilon+\frac{\varepsilon}{x}$$

Se x è piccolo l'errore relativo dell'algoritmo potrebbe essere grande.

L'algoritmo è instabile per valori di x più piccoli dell'unità di arrotondamento.

Stabilità dell'algoritmo della somma di n numeri finiti

Studieremo ora qual è l'errore da cui è affetta la somma "finita" (cioè ottenuta in aritmetica finita) di n numeri finiti, rispetto alla somma in aritmetica reale degli n numeri finiti, che indicheremo con S.

Siano $x_1, x_2, \ldots . x_n$ n **numeri finiti** da sommare.

L'algoritmo somma è il seguente:

$S:=x_1$

for i=2,…,n

$S:=S+x_i$

endfor

I passi eseguiti dall'algoritmo sono:

$S_2 = fl(S_1 + x_2)$

$S_3 = fl(S_2 + x_3)$

.

.

$S_n = fl(S_{n-1} + x_n)$

Quindi il risultato finale S_n differisce dal risultato teorico S.

L'errore algoritmico è $\left|\frac{S - S_n}{S}\right|$ è dato da:

$$\left|\frac{S - S_n}{S}\right| \leq \frac{(|x_1|n + |x_2|(n-1) + \cdots + |x_n|)}{|S|} \cdot 1.01 \cdot \varepsilon$$

dove $|\varepsilon| \leq u$

Da qui si vede che ogni addendo è moltiplicato per un peso fisso. Tale formula mostra che, assegnati i numeri finiti $x_1, x_2, \ldots . x_n$, la maggiorazione **dell'errore relativo della loro somma "finita" è minima se si sommano questi numeri in modo che i loro**

valori assoluti siano in ordine crescente, cioè: $|x_1| \leq |x_2| \leq \ldots \leq |x_n|$. Infatti, così operando,ai pesi (1.01 ε)n, (1.01 ε)(n-1) più grandi si associano i numeri $|x_1|, |x_2|, \ldots, |x_n|$ più piccoli. Ne segue che l'errore pesa meno e si avranno perciò risultati più attendibili.

La formula può essere riscritta come:

$$\left|\frac{S - S_n}{S}\right| \leq \frac{|x_{max}|}{S} 1.01 \cdot \varepsilon \, \frac{n(n+1)}{2}$$

che ci dice anche che l'errore dipende dal quadrato del numero degli elementi che si sommano.

Se $|x_{\max}|$ è elevato ma S è piccola, come può avvenire se si sommano addendi di segno opposto e modulo simile

$$\Rightarrow \frac{|x_{\max}|}{|S|} \text{ è grande}$$

cioè l'errore **relativo può crescere molto.**

Errore nel moltiplicare n numeri finiti.

Siano $x_1, x_2, \ldots\ldots x_n$ n numeri di macchina

L'algoritmo prodotto è il seguente:

P:=x_1

for i=2,…,n

P := P×x_i

endfor

I passi eseguiti dall'algoritmo sono:

$P_2 = fl(x_1 \times x_2)$

$P_3 = fl(P_2 \times x_3)$

.

$P_n = fl(P_{n-1} \times x_n)$

L'errore relativo è

$$\left|\frac{P-P_n}{P}\right| \leq 1.01 \cdot (n-1) \cdot \varepsilon$$

dove $|\varepsilon| \leq u$

L'errore algoritmico del prodotto di *n* numeri di macchina quindi cresce linearmente con *n e* non dipende dai valori degli elementi di cui si fa il prodotto..

L’analisi della stabilità di un metodo numerico, iniziata da Von Neumann e Goldstine nel 1947, si può effettuare seguendo strategie alternative:

Analisi in avanti:

si stimano le variazioni sulla soluzione dovute sia a perturbazione nei dati, sia ad errori intrinseci al metodo numerico.

Analisi all’indietro:

si considera la soluzione calcolata con i numeri finiti come soluzione esatta di un problema perturbato e si studia a quale perturbazione sui dati corrisponde la soluzione trovata. Vedremo un’applicazione di questa tecnica nello studiare la stabilità della fattorizzazione di una matrice.

Bontà di un algoritmo

Un algoritmo per essere definito "buono" oltre ad essere **generale e robusto**, applicabile cioè ad un qualsiasi insieme di dati di un certo dominio, e **stabile**, deve richiedere il **numero di operazioni minimo** possibile per ottenere il risultato ed **allocare la quantità di memoria minima possibile.**

Si definisce **complessità computazionale** di un algoritmo il numero di operazioni aritmetiche floating point richieste per la sua esecuzione. L'unità di misura è il **flop** (floating point operation): 1 flop (1 operazione elementare +,-,*, /)

Esempio 1.

La soluzione di un sistema lineare di ordine *n,* con il metodo di Cramer, ha una complessità del'ordine di O((n+1)!), risolvere il sistema lineare di ordine *n*, con il metodo di Gauss ha una complessità di $O(n^3)$.

Nell'ipotesi che il tempo di esecuzione di una singola operazione floating point è di 10^{-7} secondi, si ha il seguente confronto di tempi per i due metodi, al variare dell'ordine del sistema

n	Metodo di CRAMER	Metodo di GAUSS
10	4 secondi	$3.3x10^{-5}$sec
14	36 ore	$9.1x10^{-5}$sec
18	386 anni	$1.9x10^{-4}$sec

Quindi il metodo di Cramer non è consigliabile nella soluzione di un sistema lineare a causa della sua complessità computazionale che cresce in maniera fattoriale con l'ordine del sistema.

Esempio 2.

Sia A una matrice di dimensione $m \times n$, e sia x un vettore colonna di n componenti, il prodotto matrice vettore è così definito:

$$y = Ax \qquad y_i = \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1,..m$$

Per calcolare il generico elemento y_i sono necessarie n moltiplicazioni ed n somme, quindi poiché gli elementi del vettore y da calcolare sono m, il costo computazionale è $2n \cdot m$. Nel caso in cui $m=n$, la complessità è un $O(n^2)$.

Esempio 3.

Calcolare un polinomio in un punto:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Consideriamo il seguente algoritmo:

Algoritmo 1

$p=a_0$, $r=1$

for i=1,..,n

$r=r \cdot x$ *aggiorna la potenza di x, moltiplicando per x la potenza precedente*

$p=p+a_i \cdot r$ *al passo i, somma al polinomio di grado i-1 il monomio di grado i*

end

Complessità computazionale:

Per ogni valore i del ciclo vengono realizzate 2 moltiplicazioni ed una somma.

Quindi la complessità è di 2n moltiplicazioni ed n somme.

E' possibile valutare un polinomio in un punto mediante il metodo di **Ruffini-Horner:**

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + a_n x)\ldots))$$

Per esempio:

$p(x)=6+12x+15x^2-12x^3$ ⟹ 6+x(12+x(15-12x))

L’algoritmo è il seguente:

Algoritmo 2 (di Ruffini Horner)

$p=a_n$

for i=1,..n

$p=a_{n-i}+p\cdot x$

end

Questo algoritmo valuta il polinomio in un punto con una complessità computazionale di n moltiplicazioni ed n addizioni.

E’ possibile dimostrare che l’algoritmo di Ruffini-Horner è più stabile rispetto all’**Algoritmo 1**.