Analisi degli errori

1 Introduzione

I sitemi di rappresentazione numerica sono *posizionali*: ogni cifra occupa una posizione corrispondente ad una potenza della base del sistema adottato.

Sistema decimale (base 10)

$$10258.5 = 1 * 10^4 + 0 * 10^3 + 2 * 10^2 + 5 * 10^1 + 8 * 10^0 + 5 * 10^{-1}$$

Sistema binario (base 2)

$$(10010.1)_2 = 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^-1 = (18.5)_{10}$$

Sistema esadecimale (base 16)

$$(2A1)_{16} = 2 * 16^2 + A * 16^1 + 1 * 16^0 = (673)_{10}$$

Le cifre di questi sistemi sono:

decimale: 0,1,...,9
binario: 0,1
esadecimale: 0,1,2,...,9,A,B,C,D,E,F

Quindi, scelta una base n >1, ogni reale a pu essere scritto:

$$a = \pm(a_m N^m + a_{m-1} N^{m-1} + \ldots + a_0 + a_{-1} N^{-1} + \ldots)$$

$$0 \leq a_i \leq N - 1$$

Tale rappresentazione unica tranne se la parte frazionaria contiene infinite cifre consecutive $a_{-k} = N - 1$.

Una rappresentazione equivalente é quella di considerare il nuovo numero ottenuto sopprimendo la successione e aggiungendo una unitá all'ultima cifra rimasta.

Per esempio nel sistema decimale:

0.72999...9.. e 0.73

rappresentano lo stesso numero.

Per la rappresentazione dei numeri in diverse basi si ha il seguente:

TEOREMA.

Siano $b \in \aleph, b \geq 2, x \in \Re, x \neq 0 \Rightarrow \exists_1 e \in \mathrm{Z}, a_{ii=1,2,..}$ $a_i \in \aleph$:

$$x = \pm(\sum_{i=1}^{\infty} a_i b^{-i}) b^e$$

$0 \leq a_i \leq b - 1, a_1 \neq 0, a_i$ definitivamente $\neq b - 1$

2 Rappresentazione numerica

Poich in un calcolatore lo spazio di memoria finito, la sommatoria precedente pu estendersi fino a t $< \infty$:

$$x = \pm(\sum_{i=1}^{t} a_i b^{-i})b^e \; t < \infty \quad L \leq e \leq U$$

Abbiamo due tipi di rappresentazione.

Rappresentazione in virgola fissa.

Sono fissati il numero di cifre N che rappresenta il numero e il numero di cifre prima e dopo la virgola
$N_1, N_2 : N = N_1 + N_2$

Es: $N = 10, N_1 = 4, N_2 = 6$
$27.325 \rightarrow 0027 \quad 325000$
$0.024 \rightarrow 0000 \quad 024000$

Quindi se N sono le posizioni di memoria ed una per il segno, N - K - 1 per la parte intera e K per quella decimale, si ha:

$$x = (-1)^s b^{-k} \sum_{j=0}^{N-2} a_j b^j$$

dove b la base ed s scelto in base al segno di x.

Rappresentazione in virgola mobile

In questo caso, la posizione della virgola di un numero decimale non fissa ma data dall' esponente $\forall x \in \Re$ si pu scrivere come:

$$x = (-1)^s \, mb^e$$

dove $m \in \Re$. Tale rappresentazione non unica. Infatti:

$$x = mb^e = m'b^{e-1} = m''b^{e+1} = \ldots$$
$$m' = mb, \quad m'' = \frac{m}{b}$$

Tale rappresentazione si dice *normalizzata* se:

$$b^{-1} \leq |m| < 1$$

In tal caso chiamiamo m mantissa ed e esponente (caratteristica) di x.

$$x = (-1)_s m b^e, \quad m = a_1 a_2 \dots a_t$$

$$t \in N, \quad 0 \leq a_i \leq b-1, \quad 0 \leq m \leq b^t - 1, \quad L < e < U$$

Lo spazio riservato a $\forall x \in \Re$:

s	e	$\|m\|$

L'insieme dei numeri la cui mantissa rappresentabile da t cifre e la cui caratteristica compresa tra L ed U, che sono interi che variano per $\forall$ calcolatore, detto insieme di numeri macchina che indicheremo con F.

$$F(t, b, L, U) = \{0\} \cup \{x \in \Re : x = \pm(\sum_{i=1}^{t} a_i b^{-i}) b^e\}$$

poich 0 ha una rappresentazione particolare.

F un insieme la cui cardinalita' e':

$$\text{card } F = 1 + 2(b-1)b^{t-1}(U - L + 1)$$

Per memorizzare e spesso si agisce nel seguente modo: fissato L si memorizza: $e^* = e - L$ che sempre non negativa, tenendo presente che, in tal caso, il numero memorizzato a meno di b^L.
Poiche' in un calcolatore a 32 bits si hanno 1 bit per il segno e 8 bit per l'esponente, si ha:

$$-127 \leq e \leq 127 \Rightarrow 0 \leq e^* \leq 255 \quad (L = -127, U = 128),$$

e 23 bit per la mantissa.
Il pi grande e il pi piccolo numero rappresentabili sono:

$$x_{min} = (.1)_2 \cdot 2^{-127} = \frac{1}{2} \cdot 2^{-127} = 2^{-128}$$

$$x_{max} = (.1 \dots 1)_2 \cdot 2^{128} \approx 2^{128}$$

Infatti:

$$(1 \dots 1)_2 = \sum_{i=1}^{23} (\frac{1}{2})^i = \frac{1 - (\frac{1}{2})^{23}}{1 - \frac{1}{2}} = 1 - (\frac{1}{2})^{23} = 1 - 2^{-23} \approx 1$$

Cambiando l'ultimo bit della mantissa si ha, per x_{min}:

$$(\tfrac{1}{2} + 2^{-23}) \cdot 2^{-127} = x_{min} + 2^{-150}$$

che una differenza molto piccola.
Cambiando l'ultimo bit della mantissa si ha, per x_{max}:

$$((1 \dots 1)_2 - 2^{-23})2^{128} = x_{max} - 2^{105}$$

che una differenza molto grande. E' quindi meglio rappresentare numeri piccoli.

3 Conversione da base 10 a base qualunque

Sia N un numero in base 10 e convertiamolo in un numero in base $B \in N$

Sia: $N = (a_j B^j) + (a_{j-1} B^{j-1}) + \ldots + (a_1 B^1) + a_0 B^0 = (a_j a_{j-1} \ldots a_1 a_0)_B$

Dividendo N per B si ha:

$\frac{N}{B} = (a_j B^{j-1}) + (a_{j-1} Bj - 2) + \ldots + (a_1 B^0) + \frac{a_0}{B} = N_0 + \frac{a_0}{B}$

dove $N_0 < N$ e quindi $N = BN_0 + a_0$.

Quindi a_0, l'ultima cifra della rappresentazione, il resto intero di N/B.

Per ottenere la penultima cifra si deve dividere N_0/B

$\frac{N_0}{B} = (a_j B^{j-2} + \ldots + (a_2 B^0) + \frac{a_1}{B} = N_1 + \frac{a_1}{B}$

Il procedimento si arresta ad a_j per il quale si abbia $N_j = 0$.

Es.: $(741)_{10} = (a_9 a_8 a_7 \ldots a_1 a_0)_2 = (1011100101)_2$

4 Conversione da decimale a binario

Un numero decimale, per essere rappresentato nel computer, viene convertito in binario. Tale conversione pu comportare una rappresentazione approssimata. Per vedere ci vediamo come si opera tale conversione. Si hanno due casi.

- Numero > 1. Si divide per 2, se c' resto si mette 1 altrimenti 0 e si assegnano potenze di 2 crescenti
  esempio:

37	2	$1 * 2^0$	
18	2	$0 * 2^1$	
9	2	$1 * 2^2$	
4	2	$0 * 2^3$	$\rightarrow (100101)_2 = (37)_{10}$
2	2	$0 * 2^4$	
1	2	$1 * 2^5$	
0			

- Numero > 1 Si divide per ovvero si moltiplica per 2. Se il prodotto e' < 1 si ha 0 altrimenti si ha 1, si sottrae 1 e si procede come prima. Esempio:

0.2	2	$0 * 2^{-1}$	
0.4	2	$0 * 2^{-2}$	
0.8	2	$1 * 2^{-3}$	
1.6$\rightarrow$ 0.6	2	$1 * 2^{-4}$	
1.2$\rightarrow$ 0.2	2	$0 * 2^{-5}$	$\Rightarrow (0.2)_{10} = \overline{(0011)}_2$
0.4	2	$0 * 2^{-6}$	
0.8	2	$1 * 2^{-7}$	
1.6$\rightarrow$ 0.6	2	$1 * 2^{-8}$	
1.2$\rightarrow$ 0.2	2		

  Pertanto la rappresentazione del numero 0.2 che e' finita in decimale da' una rappresentazione approssimata in binario.

5 Rounding e chopping

Nell'aritmetica in virgola mobile (floating point) si ha il problema:

$$\text{dato} \quad x \in R \quad \text{come scegliere} \quad fl(x) \in F = \{0\} \cup \{x \in R : x = (\sum_{i=1}^{t} a_i b^{-i}) b^e\}$$

Si hanno i seguenti casi:

- $e < L$ underflow: $fl(x) = 0$; segnale di *warning.*
- $e > U$ overflow; segnale d'errore e arresto del programma.
- $L \leq e \leq U$ si possono avere 2 casi:

  i) $a_m = 0, M \geq t + 1 \Rightarrow x \in F$

  ii) $x \notin F, \quad fl(x) \neq x$ e si ha:

a) Chopping o troncamento.
In tal caso si esclude la parte a destra della t-esima cifra

$$fl(x) = \pm(\sum_{i=1}^{t} a_i b^{-i}) b^e$$

$$\text{e quindi} \quad fl(x) < x$$

b) Rounding o arrotondamento

$$x = \pm(\frac{a_1}{b} + \ldots + \frac{a_t}{b_t} + \frac{a_{t+1}}{b^{t+1}} + \ldots) b^e$$

$$fl(x) = \pm(\frac{a_1}{b} + \ldots + \frac{a_{t-1}}{b^{t-1}} + \frac{\alpha_t}{b^t}) b^e$$

con:

$$\alpha_t = \begin{cases} a_t & \text{se} \quad 0 \leq a_{t+1} < b/2 \\ a_t + 1 & \text{se} \quad \frac{b}{2} \leq a_{t+1} \leq b - 1 \end{cases}$$

cioe' si aggiunge $\frac{1}{2} b^{-t}$ e si tronca alla t-esima cifra.
Nel primo caso $fl(x) < x$, nel secondo $fl(x) > x$.

Poiche' le mantisse dei numeri macchina $\overline{p}$, che sono $b^{-1} \leq |\overline{p}| < 1$, non hanno pi di t cifre, la distanza tra due mantisse di macchina b^{-t}. Quindi se $\overline{p_1}$, $\overline{p_2} > 0$ e $\overline{p_2}$ consecutivo a $\overline{p_1}$, si ha: $\overline{p_2} = \overline{p_1} + b^{-t}$
Quindi con a) tutti tali p vengono sostituiti con $\overline{p1}$ e quindi $|p - \overline{p_1}| < b^{-t}$.

Invece con b) tutte le mantisse in $(\overline{p_1}, \overline{p_1} + \frac{1}{2}b^{-t})$ sono sostituite con $\overline{p_1}$ e tutte le mantisse in $(\overline{p_1} + \frac{1}{2}b^{-t}, \overline{p_2})$ con $\overline{p_2}$. Pertanto se $\overline{p}$ e' la mantissa associata a p si ha: $|p - \overline{p}| \leq \frac{1}{2}b^{-t}$

Quindi, per l'errore assoluto si ha:

$$|x - fl(x)| \leq \begin{cases} b^{-t}b^e & \text{chopping} \\ \frac{1}{2}b^{-t}b^e & \text{rounding} \end{cases}$$

per l'errore relativo:

$$\frac{|x - fl(x)|}{|x|} \leq \begin{cases} b^{-t+1} & \text{chopping} \\ \frac{1}{2}b^{-t+1} & \text{rounding} \end{cases}$$

L'errore relativo generalmente e' piu' importante. Mostriamolo con due esempi. Supponiamo di avere due modi di calcolare un valore x_c da un valore dato x e siano E_A ed E_R l'errore assoluto e quello relativo, rispettivamente.
Calcolo A:

$$x = 0.5 * 10^{-4}, \quad x_c = 0.4 * 10^{-4}$$
$$E_A = 0.1 * 10^{-4}$$
$$E_R = 0.2$$

ovvero, mentre apparentemente i valori di x ed x_c differiscono di poco, l'errore relativo e' del 20% . Calcolo B:

$$x = 5000, \quad x_c = 4950$$
$$E_A = 50$$
$$E_R = 0.01$$

In questo calcolo, invece, i valori di x ed x_c sembrano molto diversi ma l'errore e' invece piccolo, dell' 1%.
Si definisce precisione o epsilon di macchina l'estremo superiore dell'errore relativo, cioe':

$$\epsilon_M = \begin{cases} b^{-t+1} & \text{chopping} \\ \frac{1}{2}b^{-t+1} & \text{rounding} \end{cases}$$

In pratica ϵ_M e' il piu' piccolo numero macchina per cui $1 + \epsilon_M > 1$. Cioe' ϵ_M rappresenta la distanza tra 1 ed il successivo numero floating - point. Non ha senso cercare delle approssimazioni con precisione inferiore ad ϵ_M. In Matlab ϵ_M e' calcolato con la variabile di sistema *eps*.

6 Distribuzione dei numeri floating - point

I numeri floating-point non sono equispaziati ma si addensano in prossimita' del piu' piccolo numero rappresentabile. La spaziatura tra due $x_1, x_2 \in F$ vale al piu' $b^{1-t}|x_2|$. Invece, all'interno dell'intervallo $[b^e, b^{e+1}]$ la distanza relativa tra un numero e il successivo piu' grande ha un andamento periodico che dipende dalla mantissa. Tale fenomeno oscillatorio e' tanto piu' pronunciato quanto piu' grande e' la base b. Anche per questo motivo si preferisce usare basi piccole.

7 Operazioni aritmetiche

Altra fonte di errori e' il fatto che il risultato di operazioni aritmetiche puo' non essere un numero macchina.
Indichiamo con op la generica operazione aritmetica.
Siano: $x = fl(x), y = fl(x)$.
Se: $(x \text{ op } y) \notin F$ si deve definire: $\boxed{\text{op}}$ il cui risultato $\in F$:

$$x \boxed{\text{op}} y = fl(x \,\text{op}\, y)$$

valida se non c' overflow.
Un esempio e' dato dalla somma. In tal caso, si rendono uguali gli esponenti, si sommano le mantisse in un accumulatore con $2t$ cifre e infine si aggiustano gli esponenti.
Cio' puo' portare alla non validita' delle proprieta' commutativa, distribuitiva ed associativa della somma e della moltiplicazione.
Come esempio, supponiamo di avere 4 cifre per la mantissa e si vuole eseguire la seguente somma:

$$\sum_{i=1}^{11} x_i$$

con $x_1 = 0.5055 * 10^4, \quad x_i = 0.4000 * 10^0, \quad i = 2, 11.$
Da cui: $x_1 + x_2 = 0.5055 * 10^4 + 0.00004 * 10^4 = 0.50554 * 10^4 = 0.5055 * 10^4$ che da' un risultato errato. Se invece si esegue la somma:

$$\sum_{i=1}^{11} x_i = x_1 + \sum_{i=2}^{11} x_i = 0.5055 * 10^4 + 0.0004 * 10^4 = 0.5059 * 10^4$$

si ottiene il risultato corretto. Pertanto, per sommare molti numeri dello stesso segno si deve eseguire la somma dal piu' piccolo al piu' grande per minimizzare la propagazione degli errori.

Ci chiediamo adesso quale delle quattro operazioni puo' provocare una perdita di precisione.
Abbiamo detto che i risultati di operazioni aritmetiche tra numeri macchina in generale non sono numeri di macchina. L'operazione di macchina associa a due numeri di macchina un terzo numero di macchina. Indichiamo con $\oplus \ominus \odot \oslash$ le operazioni di macchina:

$$\begin{gathered}
\overline{a} = fl(a) \\
\overline{b} = fl(b) \\
\overline{a} \oplus \overline{b} = fl(\overline{a} + \overline{b}) = (\overline{a} + \overline{b})(1 + \epsilon_A) \\
\overline{a} \ominus \overline{b} = fl(\overline{a} - \overline{b}) = (\overline{a} - \overline{b})(1 + \epsilon_S) \\
\overline{a} \odot \overline{b} = fl(\overline{a} * \overline{b}) = (\overline{a} * \overline{b})(1 + \epsilon_P) \\
\overline{a} \oslash \overline{b} = fl(\overline{a}/\overline{b}) = (\overline{a}/\overline{b})(1 + \epsilon_D) \\
|\epsilon_i| \leq eps \equiv \epsilon_M
\end{gathered}$$

Esaminiamo l'errore relativo:

$$\frac{|(x_1 op x_2) - (\overline{x_1} op \overline{x_2})|}{|x_1 op x_2|} = \frac{|(x_1 op x_2) - [x_1(1+\epsilon_1)] op [x_2(1+\epsilon_2)]|}{|x_1 op x_2|}$$

Prodotto:

$$\frac{x_1 x_2 - x_1(1+\epsilon_1)x_2(1+\epsilon_2)}{x_1 x_2} = -\epsilon_1 - \epsilon_2 - \epsilon_1\epsilon_2 \approx -\epsilon_1\epsilon_2$$

Divisione:

$$\frac{\frac{x_1}{x_2} - \frac{x_1(1+\epsilon_1)}{x_2(1+\epsilon_2)}}{\frac{x_1}{x_2}} = \frac{-\epsilon_1 + \epsilon_2}{1+\epsilon_2} \approx -\epsilon_1 + \epsilon_2$$

Somma algebrica:

$$\frac{(x_1 + x_2) - [x_1(1+\epsilon_1) + x_2(1+\epsilon_2)]}{x_1 + x_2} = -\frac{x_1}{x_1 + x_2}\epsilon_1 - \frac{x_2}{x_1 + x_2}\epsilon_2$$

$$\left|\frac{x_i}{x_1 + x_2}\right| \to \infty \quad per \quad x_1 + x_2 \to 0$$

da cui possiamo dedurre una tabella per l'errore assoluto e per l'errore relativo nelle quattro operazioni.
Indichiamo, per brevita', con $\delta_{x^*} = x - x^*$ e con $\epsilon_{x^*} = \frac{\delta x^*}{x^*}$

operazione	errore assoluto	errore relativo
$+$	$\delta_{x_1} + \delta_{x_2}$	$\frac{x_1}{x_1+x_2}\epsilon_{x_1} + \frac{x_2}{x_1+x_2}\epsilon_{x_2}$
$-$	$\delta_{x_1} - \delta_{x_2}$	$\frac{x_1}{x_1-x_2}\epsilon_{x_1} + \frac{x_2}{x_1-x_2}\epsilon_{x_2}$
$*$	$x_2\delta_{x_1} + x_1\delta_{x_2}$	$\epsilon_{x_1} + \epsilon_{x_2}$
$:$	$\frac{\delta_{x_1}}{x_2} - \frac{x_1}{x^2{}_2}\delta_{x_2}$	$\epsilon_{x_1} - \epsilon_{x_2}$

$+-$ Nessun problema per l'errore assoluto, mentre l'errrore relativo grande se $x_1 \approx x_2$ (fenomeno di cancellazione)

$*$ l'errore relativo e' ok invece l' errore assoluto dipende dall'ordine di grandezza dei fattori

$:$ l'errore relativo e' ok mentre l'errore assoluto e' grande se $x_2 \approx 0$.

Esempio di cancellazione

$$f(x) = x(\sqrt{x+1} - \sqrt{x})$$

e supponiamo di avere 6 cifre decimali per la mantissa. Indichiamo con $\overline{f}(x)$ il valore calcolato.

x	$\overline{f}(x)$	$f(x)$
1	0.4142210	0.4142214
10	1.54340	1.54347
10^2	4.99000	4.98756
10^3	15.000	15.8074
10^4	50.0000	49.9988
10^5	100.000	198.113

Le cifre sottolineate sono affette da errore; si noti che per 10^5 l'informazione persa.
Per evitare l'errore commesso nel calcolare la differenza di numeri vicini si puo' procedere cosi':

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

$$x\frac{x+1-x}{\sqrt{x+1} + \sqrt{x}} = \frac{x}{\sqrt{x+1} + \sqrt{x}}$$

8 Condizionamento e stabilita'

Consideriamo il problema: trovare x tale che:

$$F(x) = d \tag{1}$$

dove d il dato da cui dipende la soluzione x ed f la relazione funzionale che lega x e d. Diremo che tale problema e' ben posto se, per un certo dato, la soluzione esiste, e' unica e dipende con continuita' dai dati.
La dipendenza continua dai dati significa che piccole perturbazioni sui dati danno luogo a piccole variazioni della soluzione, dove *piccolo* puo' essere inteso in senso relativo o assoluto.
D'altronde abbiamo visto che si commettono errori sia nel rappresentare i numeri reali, sia nell'esecuzione di oprazioni aritmetiche. Nascono, a questo punto, due problemi. Ci si chiede:
1) alterando i dati del problema di quanto si altera la soluzione;

2) come si propagano gli errori.
Il primo problema e' connesso con la dipendenza continua dai dati della soluzione e puo' essere stimato con il numero di condizionamento del problema, numero che non dipende dall'uso dell'aritmetica finita del calcolatore ma dal tipo di problema (1).

Supponiamo di alterare di δd i dati di (1) e ci chiediamo quanto sia δx ovvero di quanto si altera la soluzione:

$$d + \delta d \rightarrow x + \delta x.$$

Diremo *numero di condizionamento relativo* :

$$K = \frac{||\delta x||/||x||}{||\delta d||/||d||}$$

Se $x = 0$, $d = 0$ si calcola il *numero di condizionamento assoluto* :

$$K_{ass} = ||\delta x||/||\delta d||$$

Se K e' grande il problema e' mal condizionato. Ma se un problema e' ben posto e K e' grande basta riformulare il problema.
Il secondo problema dipende dalla stabilita' dell'algoritmo. Ricordiamo che per algoritmo intendiamo una successione finita di passi che trasformi un vettore di dati in un corrispondente output. Ad ogni problema numerico si possono associare pi algoritmi . Un algoritmo e' stabile se la propagazione degli errori, dovuti all'aritmetica di macchina, e' limitata. Un algoritmo e' piu' stabile di un altro se in esso l'influenza degli errori e' minore.

Esempio di problema mal condizionato:

$$\begin{aligned} x - \quad & y \quad & = 1 \\ x - \quad & 1.00001y & = 0 \end{aligned}$$

La soluzione e': x=100001, y=100000
ma:

$$\begin{aligned} x - \quad & y \quad & = 1 \\ x - \quad & 0.99999y & = 0 \end{aligned}$$

ha soluzione:
x=-99999, y=-100000

Pertanto una variazione di $2 * 10^{-5}$ nei dati ha portato ad un cambiamento di $2 * 10^5$ nella soluzione e il numero di condizionamento e' quindi dato da $2 * 10^{10}$

-Esempio di algoritmo instabile
Sia:

$$s_n = 10^6 + \sum_{i=1}^{n} \frac{1}{i} \qquad n = 1, \ldots, N$$

ovvero: $s_0 = 10^6, \quad s_n = s_{n-1} + \frac{1}{n} \qquad n = 1, \ldots, N$
Algoritmo 1:

$s = 10^6$
for $n = 1, N$
$s = s + \frac{1}{n}$
end

Si ha che per $n \geq 33$ i valori di s non cambiano piu'.
L'algoritmo puo' essere reso stabile nel seguente modo.
Algoritmo 2:

$s_0 = 0$
for $n = 1, N$
$s_n = s_{n-1} + \frac{1}{n}$
end
$s_n = s_n + 10^6$

9 Schema di Horner (nested multiplication)

Fin qui non ci siamo occupati della complessita' computazionale di un algoritmo. Questo problema riguarda il numero delle operazioni che l'algoritmo deve eseguire. Chiaramente, tra due algoritmi stabili, sceglieremo quello che e' piu' efficiente, ovvero che necessita di un numero minore di operazioni, perche' tale strategia permette di diminuire il tempo di CPU. Un esempio di calcolo efficiente e' dato dallo schema di Horner che permette di ridurre il numero di operazioni per il calcolo di un polinomio. Supponiamo di voler eseguire il calcolo di:

$$f(x) = ax^3 + bx^2 + cx + d$$

In tal caso, sono necessarie 3 somme e 6 moltiplicazioni per un totale di 9 operazioni floating-point (9 *flops).*
In generale, per calcolare: $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$
occorrono:

n somme e $\frac{n(n+1)}{2}$ moltiplicazioni, per un totale di:
$\frac{n(n+1)}{2} + n = \frac{n(n+3)}{2} \sim n^2$ *flops.*

Se invece si applica lo schema di Horner:

$$f(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + x_{a_n}) \ldots)))$$

occorrono n somme ed n moltiplicazioni per un totale di $\sim n$ *flops.*