

Metodi computazionali della Fisica
A.A. 2004-2005

Appunti lezioni nel sito:

<http://www.scienze.unipd.it/elearning>

Per consultazione:

Testi:

- Koonin-Meredith, "Computational Physics"
(Addison-Wesley)
- Landau-Paez, "Computational Physics"
(Problem solving with computers) (Wiley)

Esame: **Scritto**, *tipologia ancora da definire.*

Metodi computazionali della Fisica
A.A. 2004-2005

Scopo del corso:

- Presentare alcuni algoritmi fondamentali
- Mostrare la loro applicazione a casi di interesse fisico

Il corso non e':

- Un corso di programmazione
- Un corso di informatica

Metodi computazionali della Fisica A.A. 2004-2005

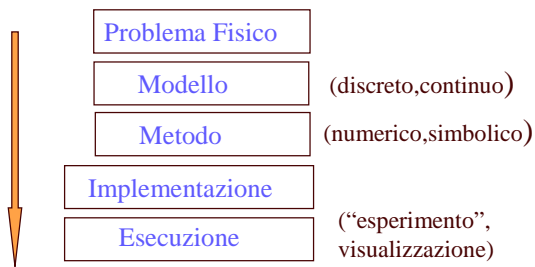
I computers servono per risolvere problemi la cui difficoltà e/o complessità sono tali da renderli intrattabili con metodi analitici.

Fisica computazionale

Approccio teorico (modelli)

Simulazione come esperimento virtuale

Approccio generale:



18/10/2004

3

Rappresentazione dei numeri in una certa base

Sia b la base \Rightarrow la stringa: $a_k a_{k-1} a_{k-2} \dots a_0 a_{-1} a_{-2} \dots a_{-k}$

rappresenta il valore $\sum_{i=-k}^k a_i b^i = a_k b^k + a_{k-1} b^{k-1} + \dots + a_0 b^0 + a_{-1} b^{-1} + \dots + a_{-k} b^{-k}$

Esempi:

$$741.36 = 7 \cdot 10^2 + 4 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 6 \cdot 10^{-2}$$

$$(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (9)_{10}$$

18/10/2004

4

Rappresentazione dei numeri in un computer

Unita' microscopiche di memoria: BITS 0 e 1

Un computer e' caratterizzato dal numero di bits usati per immagazzinare un numero ("WORD LENGTH", espressa in BYTES)

1 BYTE = 1B = 8 BITS

1K = 1KB = 2^{10} BYTES = 1024 BYTES

1 BYTE e' anche la quantita' di memoria necessaria per Immagazzinare un singolo carattere come "a"

18/10/2004

5

Rappresentazione dei numeri in un computer

I numeri sono rappresentati in forma binaria

Due rappresentazioni: *fixed point* o *floating point*.

Fixed point representation (N bit)

Primo bit rappresenta il segno di un intero e i restanti (N-1) rappresentano il valore



Gli interi rappresentabili in un computer ad N bits saranno (in modulo) quelli nell'intervallo $[0, 2^{N-1}]$.

18/10/2004

6

Rappresentazione dei numeri in fixed point (interi)

$$(1001)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (9)_{10}$$

- Un numero in questa rappresentazione e' esatto
- L'aritmetica tra numeri in questa rappresentazione e' esatta se:
 1. Il risultato non e' fuori dal range della rappresentazione.
 2. La divisione deve produrre un intero eliminando ogni resto intero.

18/10/2004

7

Rappresentazione dei numeri reali in virgola mobile (Floating point numbers)

In questa notazione (usata nei calcoli scientifici) il numero x si scrive:

$$x_{float} = (-1)^s \cdot \text{mantissa} \cdot b^{\text{exp} \text{fld} - \text{bias}}$$

segno Cifre significative del numero Base (=2)

Tipicamente **expfld** e' un *intero* ad **8 bits** nell'intervallo **[0,255]**

Bias=127 → l'esponente varia nell'intervallo **[-127,128]**

Poiche' **1 bit** e' usato per il segno (s) e **23 bits** per la mantissa (numero intero) si ha un totale di **32 bits**.

Si parla di **numeri reali in singola precisione** (32 BITS=4 BYTES). Hanno precisione di 6-7 decimali (1 su 2^{23}) e grandezza compresa tra

$$10^{-39} \text{ e } 10^{38}$$

18/10/2004

8

La **mantissa** e' rappresentata in memoria nella forma:

$$mantissa = m_1 \cdot 2^{-1} + m_2 \cdot 2^{-2} + \dots m_{23} \cdot 2^{-23}$$

intero 0,1

Esempio: Il numero **0.5** e' immagazzinato come:

$$\frac{1}{2} = 0 \quad \underbrace{0111}_{\text{Expfld}=127} \quad \underbrace{1111}_{\text{Expldf-bias}=0} \quad 1000 \quad 0000 \quad 0000 \quad 0000 \quad 0000 \quad 000$$

↓
segno

$$6 = 0 \quad \underbrace{1000}_{\text{Expfld}=130} \quad \underbrace{0010}_{\text{Expldf-bias}=3} \quad 1100 \quad 0000 \quad 0000 \quad 0000 \quad 0000 \quad 000$$

↓
segno

18/10/2004

9

Forma normalizzata della mantissa

Sia k il valore piu' piccolo per cui $m_k=1$:

$$\begin{aligned} mantissa &= m_1 \cdot 2^{-1} + m_2 \cdot 2^{-2} + \dots m_{23} \cdot 2^{-23} \\ &= 2^{-k} \left(1 + m_{k+1} \cdot 2^{-1} + \dots + m_{23} \cdot 2^{-23+k} + \dots \right) \\ &= 2^{-k} \left(1 + mantissa_{norm} \right) = \underbrace{2^{-k}}_{\text{Nell'esponente}} \left(\underbrace{1}_{\text{Omesso}} + mantissa_{norm} \right) \end{aligned}$$

Numeri piu' piccoli di 2^{-126} sono immagazzinati nella *forma denormalizzata*:

$$(-1)^s \cdot (0.FFFFFFF \dots) \cdot 2^{-126}$$

In questo modo si raggiunge il valore piu' piccolo 2^{-126-p} dove p sono i bits di precisione.

18/10/2004

10

Standard IEEE

(Institute of Electrical and Electronic Engineers)

Singola precisione S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFF
 0 1 8 9 31

Il valore V associato alla stringa si determina nel modo seguente:

- Se E=255 e F diverso da zero → V=NaN (“Non numero”)
- Se E=255 e F uguale a zero e S=1 → V= - infinito
- Se E=255 e F uguale a zero e S=0 → V = infinito
- Se $0 < E < 255$ → $V = (-1)^S * 2^{(E-127)} * (1.F)$
- Se E=0 e F diverso da zero → $V = (-1)^S * 2^{(-126)} * (0.F)$
- Se E=0 e F uguale a zero e S=1 → V=-0
- Se E=0 e F uguale a zero e S=0 → V=0

18/10/2004

11

Esempi

$$0 \ 10000000 \ 000000000000000000000000 = +1 * 2^{(128-127)} * 1.0 = 2$$

$$1 \ 10000001 \ 101000000000000000000000 = -1 * 2^{(129-127)} * 1.101 = -6.5$$

$$0 \ 00000000 \ 100000000000000000000000 = +1 * 2^{(-126)} * 0.1 = 2^{(-127)}$$

$$0 \ 00000000 \ 0000000000 \ 0000000000 \ 001 = +1 * 2^{(-126)} * \\ 0.0000000000 \ 0000000000 \ 001 = 2^{(-149)}$$

Valore piu' piccolo

18/10/2004

12

Standard IEEE

Doppia precisione

S EEEEEEEEEEE FF
0 1 11 63

Il valore V associato alla stringa si determina nel modo seguente:

- Se E=2047 e F diverso da zero → V=NaN (“Non numero”)
- Se E=2047 e F uguale a zero e S=1 → V= - infinito
- Se E=2047 e F uguale a zero e S=0 → V = infinito
- Se $0 < E < 2047$ → $V = (-1)^S * 2^{(E-1023)} * (1.F)$
- Se E=0 e F diverso da zero → $V = (-1)^S * 2^{(-1022)} * (0.F)$
- Se E=0 e F uguale a zero e S=1 → V=-0
- Se E=0 e F uguale a zero e S=0 → V=0

18/10/2004

13

In un computer a 32-BIT il **numero piu' grande** immagazzinabile e':

$$0 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 111 = 2^{128} = 3.4 \times 10^{38}$$

mentre il **numero piu' piccolo** immagazzinabile e':

$$0 \ 0000 \ 0000 \ 1000 \ 0000 \ 0000 \ 0000 \ 0000 \ 000 = 2^{-128} = 2.9 \times 10^{-39}$$

Se per i numeri reali si richiede la DOPPIA PRECISIONE
(DOUBLE PRECISION)



64 BIT=8 BYTES WORDS

→ 11 BITS per l'esponente , 52 per la mantissa

→ 16 decimali di precisione (1 parte su 2^{52}) e range
 $10^{-322} < x < 10^{308}$

18/10/2004

14

Errori di arrotondamento

Calcolo di $7+1.0 \times 10^{-9}$ in singola precisione

$7 = 0 \ 1000 \ 0010 \ 1110 \ 0000 \ 0000 \ 0000 \ 0000 \ 000,$
 $10^{-9} = 0 \ 0110 \ 0000 \ 1101 \ 0110 \ 1011 \ 1111 \ 1001 \ 010$

Gli esponenti sono diversi → non si possono sommare le mantisse!!

→ Si aumenta l'esponente del più piccolo mentre si diminuisce in maniera progressiva la mantissa *spostando i bits* a destra (i.e. inserendo zeri) fino a che i due numeri hanno gli stessi exp.

$10^{-9} = 0 \ 0110 \ 0001 \ 0110 \ 1011 \ 0101 \ 1111 \ 1100 \ 101 \ (0)$
 $= 0 \ 0110 \ 0010 \ 0011 \ 0101 \ 1010 \ 1111 \ 1110 \ 010 \ (10)$

 $= 0 \ 1000 \ 0010 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 000 \ (0001101 \dots)$

→ $7+1.0 \times 10^{-9} = 7$

18/10/2004

15

Precisione della macchina

Il modo in cui sono immagazzinati i numeri reali influisce sulla precisione dei calcoli.

Def. Si definisce MACHINE PRECISION il **massimo numero positivo** ϵ_m che può essere sommato al numero immagazzinato come "1" senza cambiarlo:

$$\epsilon_m + 1_c = 1_c,$$

Immagazzinato nella memoria del computer

Per un generico numero x si ha quindi: $x_c = x(1 + \epsilon), \quad |\epsilon| \leq \epsilon_m$

Dove $\epsilon \cong 10^{-7}$ per **singola precisione** $\epsilon \cong 10^{-16}$ **doppia precisione**

18/10/2004

16

Nota bene: ϵ_m non e' il piu' piccolo numero immagazzinabile dalla macchina. Questo numero dipende dall'esponente, mentre ϵ_m dipende da quanti bits sono nella mantissa.

Ogni operazione aritmetica tra floating numbers comporta un'introduzione di un errore almeno pari a ϵ_m .

L'iterazione di operazioni aritmetiche comporta una propagazione ed un aumento di tale errore (**errore di arrotondamento**).

18/10/2004

17

Errori ed indeterminazioni nei calcoli

I computers non sono infinitamente precisi!!

Errori di sintassi

Errori aleatori

Errori di troncamento

Fluttuazioni elettroniche, raggi cosmici ..

Per esempio sostituire una serie infinita con una somma finita oppure intervalli infinitesimi con intervalli finiti.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \cong \sum_{n=0}^N \frac{x^n}{n!} = e^x + E(x, N)$$

Errori di arrotondamento

Dovuti al fatto che ogni numero e' rappresentato Da un numero **finito** di bits.

$$2\left(\frac{1}{3}\right) - \frac{2}{3} = 2 \times 0.3333333 - 0.6666667 = -0.0000001 \neq 0.$$

18/10/2004

18

Errori da sottrazione e da moltiplicazione

Un'operazione eseguita su un computer fornisce sempre una risposta approssimata in quanto la memoria del computer e' finita. Gli errori si possono poi accumulare fino a rendere un programma instabile.

Errori da sottrazione:

$$\begin{aligned}
 a = b - c &\Rightarrow a_c = b_c - c_c, \\
 &a_c = b(1 + \varepsilon_b) - c(1 + \varepsilon_c), \\
 &\Rightarrow \frac{a_c}{a} = 1 + \varepsilon_b \frac{b}{a} - \frac{c}{a} \varepsilon_c.
 \end{aligned}$$

Se a e' piccolo $\rightarrow b \cong c$ e quindi $\frac{a_c}{a} = 1 + \varepsilon_a$, con $\varepsilon_a \cong \frac{b}{a}(\varepsilon_b - \varepsilon_c)$.

Quindi, anche se $\varepsilon_b - \varepsilon_c$ e' piccolo, viene moltiplicato da un numero molto grande rendendo ε_a grande

18/10/2004

19

Evitare, quando possibile, troppe sottrazioni.

Esempio:

$$\sum_{n=1}^{2N} (-1)^n \frac{n}{n+1}$$

(1) Somma finita con segni alternati



$$-\sum_{n=1}^N \frac{2n-1}{2n} + \sum_{n=1}^N \frac{2n}{2n+1}$$

(2) Solo una sottrazione!

Valori negativi

Valori positivi



$$\sum_{n=1}^N \frac{1}{2n(2n+1)}$$

(3) Nessuna sottrazione!

(1),(2) e (3) sono matematicamente equivalenti ma non numericamente!

18/10/2004

20

Errori da moltiplicazione:

$$\begin{aligned} a = b \times c &\Rightarrow a_c = b_c \times c_c, \\ &a_c = b(1 + \varepsilon_b) \times c(1 + \varepsilon_c), \\ &\Rightarrow \frac{a_c}{a} = (1 + \varepsilon_b)(1 + \varepsilon_c) \cong 1 + \varepsilon_b + \varepsilon_c \end{aligned}$$

Poiché ε_b e ε_c possono avere segno opposto, l'errore in a_c è talvolta maggiore e talvolta minore degli errori dei due moltiplicandi.

Si può mostrare che se ε ha segno casuale, dopo N moltiplicazioni:

$$\varepsilon_N \approx \sqrt{N} \varepsilon$$

Ci sono casi particolari in cui $\varepsilon_N \approx N \varepsilon$ o addirittura $\varepsilon_N \approx N! \varepsilon$

18/10/2004

21

Errori degli algoritmi

Algoritmo:

- Grandezza del passo h
- Numero d'iterazioni N

Buon algoritmo se per:

$$\begin{array}{ccc} \xrightarrow{h \rightarrow 0} & & \text{soluzione esatta} \\ \text{0} & & \\ \xrightarrow{N \rightarrow \infty} & & \end{array}$$

Def. Errore di approssimazione di un algoritmo:

differenza tra il valore esatto e valore ottenuto dall'algoritmo

Errore totale $\Rightarrow \varepsilon_{\text{tot}} = \varepsilon_{\text{appr}} + \varepsilon_{\text{arrot}}$

Per $N=N_c \gg 1$ o per $h=h_c \ll 1$, è possibile che $\varepsilon_{\text{arrot}} > \varepsilon_{\text{appro}}$

18/10/2004

22

Errore totale

Supponiamo che:

$$\varepsilon_{\text{appro}} \cong \frac{\alpha}{N^\beta}$$

α, β : parametri empirici

E che gli errori di arrotondamento siano scorrelati tra loro durante il calcolo, cioè:

$$\varepsilon_{\text{arrot}} \cong \sqrt{N} \varepsilon_m \longrightarrow \text{Machine precision}$$



$$\begin{aligned} \varepsilon_{\text{tot}} &= \varepsilon_{\text{appro}} + \varepsilon_{\text{arrot}} \\ &\cong \frac{\alpha}{N^\beta} + \sqrt{N} \varepsilon_m \end{aligned}$$

Ci sarà un valore di N dove un errore diventa più grande dell'altro.

18/10/2004

23

Esempio: $\alpha = 1, \beta = 2$

Supponiamo che:

$$\varepsilon_{\text{appro}} \cong \frac{1}{N^2}$$



$$\varepsilon_{\text{tot}} \cong \frac{1}{N^2} + \sqrt{N} \varepsilon_m$$

Valore estremale:

$$\frac{d\varepsilon_{\text{tot}}}{dN} = 0 \Rightarrow N^{\frac{5}{2}} = \frac{4}{\varepsilon_m}$$

E' un valore di minimo essendoci un massimo per $N \Rightarrow$ infinito e $N=0$

Per un computer a 32-bit in singola precisione

$$\varepsilon_m \cong 10^{-9}$$



$$N^{\frac{5}{2}} \cong \frac{4}{10^{-9}} \Rightarrow N \cong 1099,$$

$$\varepsilon_{\text{tot}} \cong \frac{1}{N^2} + \sqrt{N} \varepsilon_m = 8 \times 10^{-7} + 33 \times 10^{-7} \cong 4 \times 10^{-6}$$

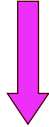
In generale la parte preponderante dell'errore e' dovuta agli errori di arrotondamento

18/10/2004

24

$$\begin{array}{r}
 100000 \times 2^{10} \\
 + 123456 \times 2^8 \\
 \hline
 101234.56 \times 2^{10}
 \end{array}$$

Esatto



$$\begin{array}{r}
 100000 \times 2^{10} \\
 + 001234 \times 2^{10} \\
 \hline
 101234 \times 2^{10}
 \end{array}$$

Shift esponente

Perdita dei bits di ordine basso