

Metodi computazionali della Fisica

A.A. 2009-2010 (P.L. Silvestrelli, psil@pd.infn.it)

Appunti lezioni (cartella “RISORSE”) in: <http://elearning.scienze.unipd.it/>

Testi **consigliati**:

- **Barone** et al., “*Programmazione Scientifica*” (Pearson)
- **Crisanti**, “*yaC Primer, Parte II Applicazioni*”(Aracne)

Testi **utili** (per consultazione):

- **Koonin-Meredith**, “*Computational Physics*” (Addison-Wesley)
- **Landau-Paez**, “*Computational Physics*” (Wiley)
- **Press**, “*Numerical Recipes (in C++)*” (Cambridge University Press)
- **Thijssen**, “*Computational Physics*” (Cambridge University Press)
- **Allen-Tildesley**, “*Computer Simulation of liquids*”(Clarendon Press Oxford)

Esame: **Relazioni scritte + colloquio**

Metodi computazionali della Fisica

A.A. 2009-2010

Scopo del corso:

- Presentare **alcuni algoritmi** fondamentali
- Mostrare la loro **applicazione** a casi di interesse fisico
- Fare **esperienze pratiche** in *Aula Informatica*.

Finalità:

- Capire che la Fisica Computazionale è **importante/indispensabile**
- Capire che è spesso possibile creare **propri** programmi per la ricerca
- Avere un atteggiamento **critico** rispetto ai programmi già disponibili.

Metodi computazionali della Fisica

A.A. 2009-2010

Scopo del corso:

- Presentare **alcuni algoritmi** fondamentali
- Mostrare la loro **applicazione** a casi di interesse fisico
- Fare **esperienze pratiche** in Aula Informatica.

Il corso non è:

- Un corso di programmazione
- Un corso di informatica.

Linguaggio di programmazione :

Come linguaggio di programmazione si adotta il C++ ; chi lo desidera può utilizzare anche il C .

Consegna 5 relazioni (una per coppia) :

Possibilmente (verrà premiata la puntualità; comunque prima dell'esame orale) :

- le *prime* 2 entro il 9 novembre 2009;
- le *seconde* 3 entro il 14 dicembre 2009 .

N.B. : l'esame finale (colloquio orale sulle relazioni) è comunque individuale !

Struttura relazioni :

- breve **introduzione** teorica sul **problema fisico** trattato e sull'**algoritmo** numerico utilizzato;
- **risultati** ottenuti, eventualmente corredati da **tabelle e grafici** (ricordare le unità di misura !), opportunamente **commentati**;
- **listati** dei programmi utilizzati, mettendo in evidenza le parti **modificate** o **sviluppate**.

Informazioni esame :

- è necessario isciversi all'esame, e consegnare tutte le relazioni (in formato cartaceo) con congruo anticipo (almeno alcuni giorni prima);
- l'esame orale consiste in un **colloquio sulle relazioni** presentate, con approfondimento sia di aspetti di **programmazione** sia riguardanti l'**applicazione** degli algoritmi a sistemi fisici;
- **ORARIO RICEVIMENTO:**
giovedì dalle ore 15.30 alle 17 (stanza 333 Dip. Di Fisica);

N.B. : l'esame è comunque **individuale**, dunque ognuno è responsabile di **ogni parte** delle relazioni !

Date appelli :

- *18 dicembre 2009* (ore 9)
- *8 gennaio 2010* (ore 9)
- *5 luglio 2010* (ore 9)
- *26 luglio 2010* (ore 9)
- *23 settembre 2010* (ore 9)

Accesso Aula Informatica (polo didattico) :

L'accesso (la **frequenza** non è strettamente obbligatoria ma **caldamente consigliata...**) avviene (**2 studenti per terminale**) secondo il seguente orario (settimana di *sospensione* dal 2 al 6 novembre) :

- **Martedì ore : 11.30-13.15** (**I turno**),
- **Mercoledì ore : 11.30-13.15** (**II turno**)
(**al polo didattico, NON in aula LUF2 !**)

Per informazioni (problemi account,...) :

http://www.fisica.unipd.it/calcolo/aula_info.html

N.B. : da macchine **Windows** si può collegarsi alle macchine **Linux** usando **ssh** : ad esempio, **ssh 192.168.1.29**

Accesso Aula Informatica (polo didattico).

Criterio suddivisione turni :

Se *X* e *Y* sono le lettere **iniziali** dei due studenti componenti la coppia:

IF

(X è tra A e M compresi) AND (Y è tra A e M compresi)

OR

(X è tra N e Z compresi) AND (Y è tra N e Z compresi)

THEN

*frequentano il **I turno** (Martedì 11.30-13.15)*

ELSE

*frequentano il **II turno** (Mercoledì 11.30-13.15)*

ENDIF

Metodi computazionali della Fisica

A.A. 2009-2010

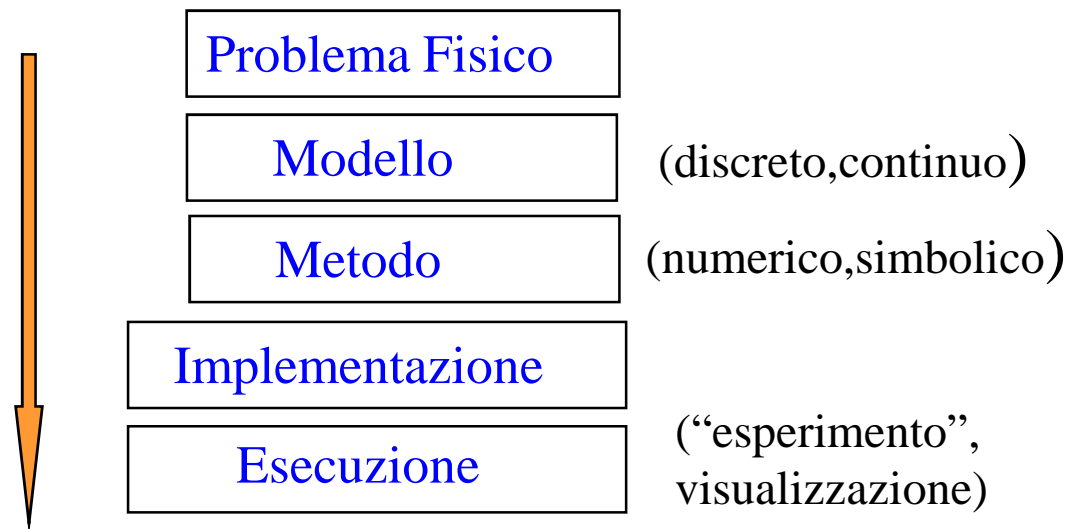
I computers servono per risolvere problemi la cui difficoltà e/o complessità sono tali da renderli **intrattabili** con metodi analitici.

Fisica computazionale

Approccio teorico (modelli)

Simulazione come *esperimento virtuale*

Approccio generale:



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:

$$\varepsilon_m + 1_c = 1_c,$$

Immagazzinato nella
memoria del computer

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

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

Nota bene: ϵ_m non è il più 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**).

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 è rappresentato da un numero **finito** di bits.

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

Errori da sottrazione e da moltiplicazione

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

Errori da sottrazione:

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

Se **a** è 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 $\varepsilon_b - \varepsilon_c$ è piccolo, viene moltiplicato da un numero molto grande rendendo ε_a grande !

Evitare, quando possibile, troppe sottrazioni !

Esempio:

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

(1) Somma finita
con segni alternati,
N sottrazioni

$$1 - \ln 2 = 0.306853$$

$$N \rightarrow \infty$$

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

(2) Solo **una** sottrazione!

Valori negativi

Valori positivi

Numericamente,

con $N=10^6$ e *float* :

0.306853 0.306853

///

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

(3) **Nessuna** sottrazione!

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

Errori da moltiplicazione:

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

Poichè 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 $\varepsilon_N \approx N! \varepsilon$

Errori degli algoritmi

Algoritmo:

- Grandezza del passo **h**
- Numero d'iterazioni **N**

Buon algoritmo se per:

$$\begin{array}{c} \xrightarrow{h \rightarrow 0} \\ 0 \\ \xrightarrow{N \rightarrow \infty} \end{array}$$

soluzione esatta

Def. Errore di approssimazione di un algoritmo:

differenza tra il valore esatto e valore ottenuto dall'algoritmo

Errore totale $\longrightarrow \mathcal{E}_{\text{tot}} = \mathcal{E}_{\text{appro}} + \mathcal{E}_{\text{arrot}}$

Per $N=N_c \gg 1$ o per $h=h_c \ll 1$, è possibile che

$$\mathcal{E}_{\text{arrot}} > \mathcal{E}_{\text{appro}}$$

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.

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

Supponiamo che:

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

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 $N \Rightarrow \text{infinito}$ e $N=0$

Per un computer a 32-bit in singola precisione

$$\varepsilon_m \cong 10^{-7}$$

$$\begin{aligned} N^{\frac{5}{2}} &\cong \frac{4}{10^{-7}} \Rightarrow N \cong 1099, \\ \varepsilon_{\text{tot}} &\cong \frac{1}{N^2} + \sqrt{N} \varepsilon_m = 8 \times 10^{-7} + 33 \times 10^{-7} \cong 4 \times 10^{-6} \end{aligned}$$

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

Ripasso elementi di programmazione in C++ (*esempio.cpp*) :

```
/* Semplice programma in C++ che calcola, in  
maniera approssimata, exp(x) come somma  
di una serie finita: */  
  
#include <iostream>  
#include <cmath>  
int main()  
{  
    using namespace std;  
    double x, esp, nfac, an;  
    int n, nmax;  
  
    // legge i valori di input:  
    cout << " inserisci nmax" << endl;  
    cin >> nmax;  
    cout << " inserisci x" << endl;  
    cin >> x;  
  
    esp=0.;  
    nfac=0.;  
  
    // loop:  
    for (n=0; n <= nmax; n++)  
    {  
        // calcola il fattoriale:  
        if (n<=1)  
            nfac=1.;  
        else  
            nfac*=n;  
        an=n;  
        esp=esp+pow(x,an)/nfac;  
    }  
    cout << " esp = " << esp << endl;  
    return 0;  
}
```

commento

Direttive per il preprocessore
(include files di header per utilizzare le librerie standard)

funzione da cui inizia l'esecuzione

→ Rende visibili le definizioni del C++

→ Dichiarazioni delle variabili

→ fine di un'istruzione

→ Inizializzazione delle variabili

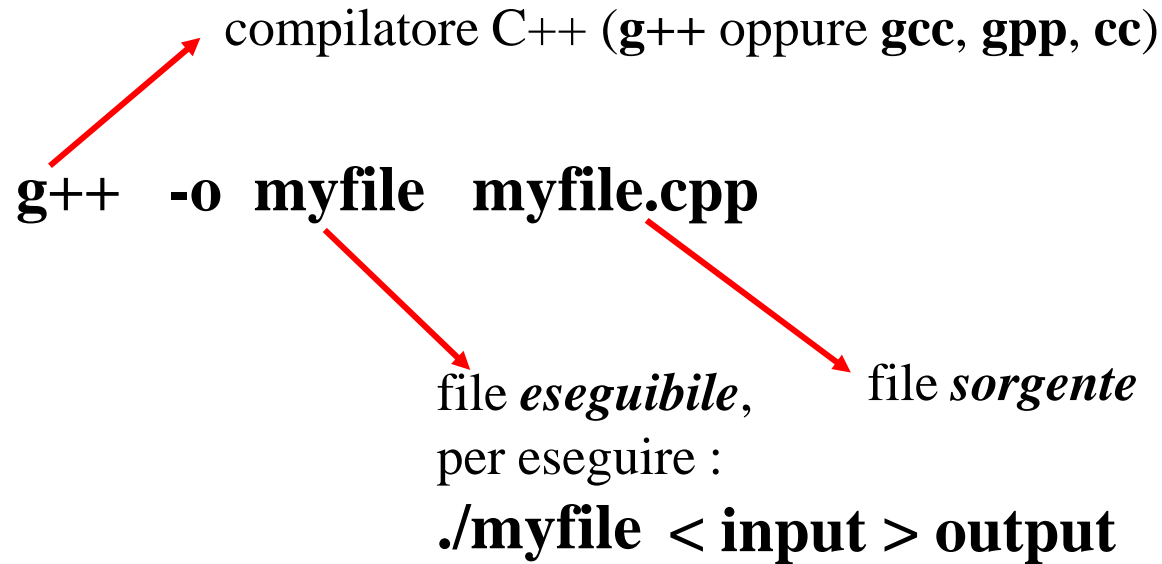
Corpo della funzione

→ Istruzione per un ciclo, con *n* che va da 0 a *nmax*

→ Istruzione condizionale

→ funzione della libreria matematica

- **Compilazione in ambiente Linux/Unix :**


g++ -o myfile myfile.cpp
compilatore C++ (**g++** oppure **gcc**, **gpp**, **cc**)
file *eseguibile*,
per eseguire :
./myfile < input > output
file *sorgente*

- **text editor :** emacs , vi , ...

- **programma per fare grafici :** gnuplot , xmgr , ...

```
g++ -o esempio esempio.cpp
```

```
./esempio
```

```
inserisci nmax
```

```
7
```

```
inserisci x
```

```
1.
```

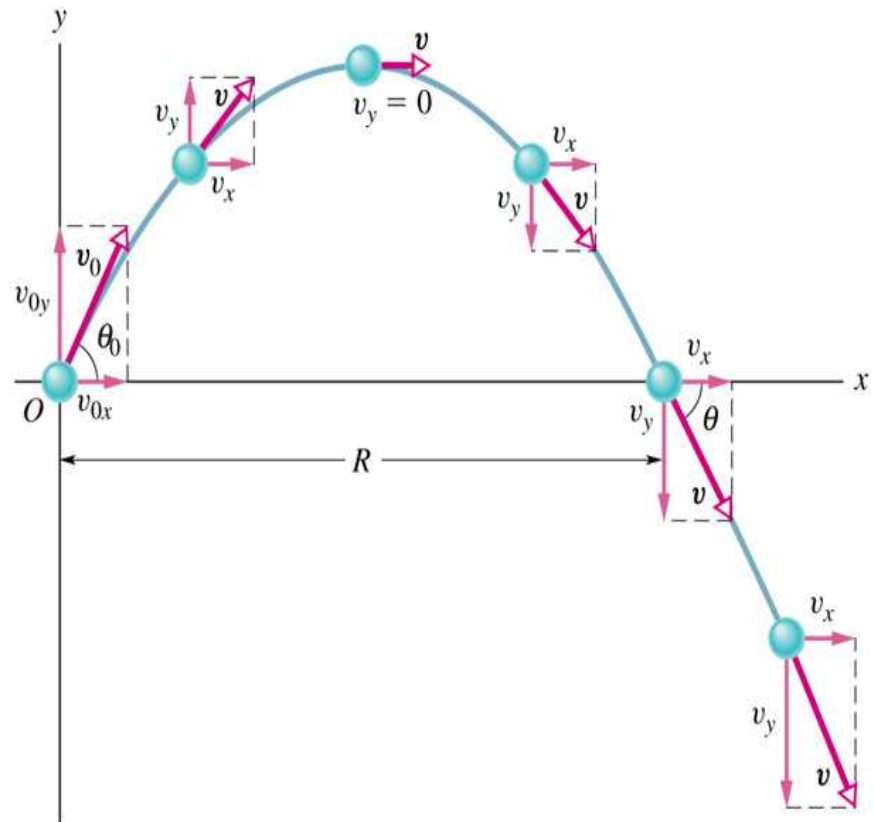
```
esp = 2.71825
```

RISULTATO ESATTO:

e=2.718281828

Problema fisico: moto di un proiettile

Classico problema di **cinematica** (in 2D) del lancio di una massa in presenza di **gravità** e assenza di attrito: sull'asse x la velocità è costante, sull'asse y è uniformemente accelerata ($a = -g$); ad ogni istante t si può calcolare la **posizione** (x, y) della massa; il problema è concettualmente semplice ma **noioso** da risolvere manualmente...



Esercizio: moto di un proiettile

Questo esercizio serve come **ripasso** del *linguaggio* C++ e delle *fasi essenziali* dell'approccio computazionale (*editare files, compilare, produrre grafici, ...*) e non viene valutato ai fini del voto finale; **N.B.** sarà comunque utile per l'esercizio successivo !!!

Scrivere ed **utilizzare** un programma in C++ in modo che chieda i parametri **iniziali** del moto (l'angolo di lancio θ_0 , in *radianti*, e la velocità iniziale v_0) ed il tempo t dall'inizio del moto per cui calcolare la **posizione** che viene stampata come **risultato**.

Esercizio: moto di un proiettile

- **verificare**, in alcuni casi, che il programma dà risultati in accordo con i calcoli **analitici**, ad esempio: $v_0=50 \text{ m/s}$, $t=3 \text{ s}$, $\theta_1=25^\circ$, $\theta_2=45^\circ$;
- **modificare** il programma in modo che sia possibile fornire in input l'angolo in **gradi** e la velocità in **km/h** ;
- **modificare** il programma in modo che sia possibile, fornendo la **massa** del corpo, calcolare l'energia **cinetica**, l'energia **potenziale** e l'energia **meccanica totale** (**verificare** la **conservazione** di quest'ultima !);
- **modificare** il programma in modo che sia possibile (**ciclo**) avere in output la posizione del corpo, in funzione del tempo, in un certo intervallo (*discretizzato*) di tempo (ad es. da **0** a **10 s**, con $\Delta t=0.2 \text{ s}$), così da poter fare un **grafico** (ad es. scrivendo i dati di output su di un file e poi usando **gnuplot**) che illustra la **traiettoria** ($y(x)$) del corpo.