Esame di Laboratorio di Fisica Computazionale

15 giugno 2015, ore 9.30

shell scripting

1. Si aggiunga al file proposto `elenco.txt` una colonna, che preceda quella dei nomi; in questa colonna si segni il numero progressivo di riga.

Mathematica

1. **Oscillatore armonico forzato**
   Si risolva l'equazione differenziale

$$x''(t) + \omega_N^2 x(t) = k\cos(\omega_F t)$$
$$x'(0) = 0, \qquad x(0) = 1$$

   Si disegnino i grafici tridimensionali che rappresentano la soluzione valutata con:
   1) $t \in [0, 50]$ e $\omega_F \in [0, 2]$, essendo fissati $k = 1$ e $\omega_N = 1$
   2) $t \in [0, 50]$ e $\omega_N \in [0, 2]$, essendo fissati $k = 1$ e $\omega_F = 1$
   3) $t \in [0, 50]$ e $k \in [0, 2]$, essendo fissati $\omega_F = 0.5$ e $\omega_N = 1$.
   Aumentare, se possibile, la qualità dei grafici.

2. **Matrici hermitiane**

   (a) Si generi una matrice 4x4 hermitiana utilizzando il comando per generare numeri casuali.

   (b) Si verifichi che gli autovalori di questa matrice sono reali.

   (c) Si calcoli il polinomio caratteristico di questa matrice e si risolva l'equazione secolare corrispondente, verificando che le radici coincidono con gli autovalori determinati al punto precedente.

3. **Matrici del gruppo SU(2)**

   (a) Si verifichi che la seguente matrice A appartiene al gruppo SU(2), ovvero al gruppo delle matrici unitarie 2x2 con determinante +1.

$$A = \begin{pmatrix} \cos(y1)\exp(iy2) & \sin(y1)\exp(iy3) \\ -\sin(y1)\exp(-iy3) & \cos(y1)\exp(-iy2) \end{pmatrix}$$

   suggerimento: si utilizzi la sostituzione `Conjugate[x_]->x` *per sfruttare il fatto che i parametri y_i sono reali.*

   (b) Si valuti la matrice A per ($y_1 = 0.01$, $y_2 = 0.3$, $y_3 = 0.5$). Si calcolino i coefficienti della scomposizione di A in termini della matrice identità e delle tre matrici di Pauli

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \ \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

4. **Campo di induzione magnetica generato da un dipolo**
   Si consideri un dipolo magnetico, di forma cilindrica, di lunghezza L e di raggio a.

(a) Le componenti del campo di induzione magnetica lungo l'asse del cilindro z e lungo la direzione radiale ρ si ottengono calcolando i seguenti integrali:

$$\begin{aligned} B_z(a,L,z,\rho) &= \int_0^a dR + \frac{R(L/2-z)}{(R^2+(L/2-z)^2+\rho^2-2R\rho)^{3/2}} + \\ &\quad + \frac{R(L/2+z)}{(R^2+(L/2+z)^2+\rho^2-2R\rho)^{3/2}} \\ B_\rho(a,L,z,\rho) &= \int_0^a dR - \frac{R(\rho-R)}{(R^2+(L/2-z)^2+\rho^2-2R\rho)^{3/2}} + \\ &\quad + \frac{R(\rho-R)}{(R^2+(L/2+z)^2+\rho^2-2R\rho)^{3/2}} \end{aligned}$$

(b) Si disegnino le linee del campo B, nel piano (z,ρ), nell'intervallo $z \in [-2,2]$ e $\rho \in [-2,2]$, per un dipolo di lunghezza $L=2$ e di raggio $a=1$.

C++

Si risolva l'esercizio proposto. Per facilitare la correzione, se possibile includere tutto in un unico file sorgente. La sufficienza è raggiunta risolvendo correttamente i primi tre punti.

Random walk in una dimensione

1. Si scriva una classe `Walker` che rappresenterà un random walk in una dimensione. Tra i membri private si metta la posizione del walker sull'asse reale. Tra i membri public si scriva un costruttore che richieda come parametro la posizione iniziale e la assegni. Si specifichi come valore di default la posizione 0.

2. Si scriva una funzione `position` (membro pubblico di `Walker`) che restituisca la posizione corrente.

3. Si implementi l'overloading dell'operatore `+=` (anch'esso membro) in modo che la sintassi `w += δ` incrementi la posizione corrente di `w` di un passo δ (col suo segno). Il comportamento al bordo (posizione 0) dovrà essere riflettente, cioè se dopo un update la posizione è negativa, essa va invertita. Questo operatore dovrà essere dichiarato virtuale per risolvere i punti successivi.

4. Si implementi anche l'operatore `-=`, che decrementa la posizione di un passo δ (col suo segno). Per fare questo si riutilizzi interamente il codice scritto al punto precedente.

5. Nel `main` si istanzi un oggetto di tipo `Walker`; gli si facciano compiere due balzi, il primo di lunghezza 2 verso destra, il secondo di lunghezza 2.5 verso sinistra, e si controlli la posizione finale, stampandola su `cout`.

6. Si scriva una classe `LazyWalker`, che erediti pubblicamente da `Walker`. Il suo comportamento è quello di eseguire i passi solo ogni tanto; più precisamente, ogni passo viene eseguito con una probabilità p. Si ponga p come membro private e si scriva un opportuno costruttore che prenda un parametro e inizializzi p (e la posizione iniziale a 0).

7. Si scriva l'operatore `+=` (override di quello ereditato da `Walk`) in modo che realizzi il passo con probabilità p. Si riutilizzi (tramite una chiamata a funzione) il codice già scritto per l'operatore della classe base. (Può essere utile usare `drand48()` come generatore di numeri pseudo-casuali compresi tra 0 e 1.)

8. Eseguire un test nel `main`, usando un `LazyWalker` con parametro $p = 0$ (che quindi non si muove mai), per verificare se anche l'operatore `-=` abbia bisogno di un override. Nel caso, implementarlo.

9. Scrivere una funzione globale `race`, che prenda come parametri due oggetti di tipo `Walker` (si valuti se passarli per copia o per referenza, considerando il punto successivo) e che

faccia eseguire ai due walker 1000 passi di lunghezza 1 diretti in modo random a destra o a sinistra. I passi fatti dai due dovranno essere esattamente gli stessi e nello stesso ordine. Il valore di ritorno della funzione indicherà quale dei due walker ha raggiunto, al termine del cammino, la posizione più distante dall'origine: 1 se è il primo, 2 se è il secondo, 0 in caso di pareggio.

10. Nel `main`, eseguire 10 000 gare, usando la funzione `race`, tra un `Walker` e un `LazyWalker` con parametro $p = 0.7$, e verificare che il secondo vince in circa un terzo dei casi. (Entrambi partono dall'origine in ogni gara.)

11. Nel `main`, si istanzi un `vector` di 100 oggetti `LazyWalker` allocati staticamente con valori di p generati in modo random tra 0 e 1. Si stampi il massimo valore di p in questo vettore, usando l'algoritmo

    ```
    std::max_element(it1, it2, pred),
    ```

    che compara a due a due gli elementi compresi tra gli iteratori `it1` e `it2` usando il predicato binario `pred`, e restituisce un iteratore all'elemento che realizza il massimo. Il predicato andrà implementato a tale scopo.