Esame di Laboratorio di Fisica Computazionale

10 giugno 2016, ore 13.30

shell scripting

Si consideri il file `matrice.dat` e si scriva uno script che sommi tutti i numeri presenti nella tabella.

1. Soluzione esplicita: si usi `awk` per sommare ciascuna delle 4 colonne e poi si combinino i risultati parziali.

2. Consultando la pagina di help (`man awk`) si consideri l'utilizzo del costrutto `for` e si identifichi la variabile che conta il numero di campi (ovvero di colonne) in una riga.

Mathematica

1. Si disegni, con $x \in [0, 8\pi]$, la soluzione dell'equazione

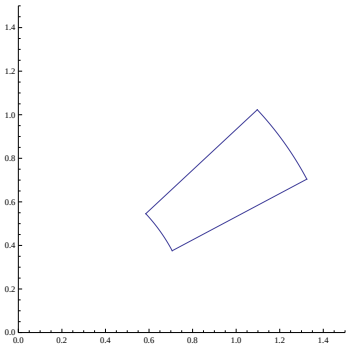
$$y''(x) + 9\,y(x) = 0, \qquad y'(0) = 0, \qquad y(0) = 1 \tag{1}$$

2. Si risolva l'equazione di Bessel

$$x^2 y''(x) + xy'(x) + (x^2 - n^2)y(x) = 0. \tag{2}$$

   Nella soluzione si scelgano, con una sostituzione, le costanti d'integrazione in modo da selezionare solo la funzione di Bessel $J_n(x)$. Si disegnino in un singolo plot, con $x \in [0, 8\pi]$, le prime cinque funzioni di Bessel (ovvero $n = 1, 2, 3, 4, 5$).

3. Si utilizzi il comando `Integrate` per calcolare l'area del arco di corona circolare in figura, delimitato da due circonferenze di raggio $r_1 = 0.8$, $r_2 = 1.5$ e da due raggi che formano con l'asse delle x rispettivamente $\phi = 28^0$ e $\phi = 43^0$.



4. Si calcoli lo stesso integrale del punto precedente utilizzando la tecnica numerica *hit-or-miss*:
   1) si generino 10000 coppie di numeri casuali in modo da ricoprire un quadrato di lato $l = 1.5$;
   2) si verifichi se il punto generato cade all'interno dell'area delimitata dalle circonferenze e dai raggi del punto precedente e quindi si determini la frazione complessiva di punti accettati;
   3) si calcoli l'area, essendo nota l'area del quadrato di lato $l = 1.5$.

5. Scomposizione **LDL**
   Una matrice hermitiana **A** definita positiva può essere scomposta nel prodotto $\mathbf{A} = \mathbf{LDL}^\dagger$, dove **L** è una matrice triangolare inferiore con gil elementi lungo la diagonale pari a 1, mentre **D** è una matrice diagonale.

Il calcolo, ricorsivo, degli elementi delle matrici **L** e **D** si ottiene con le due equazioni seguenti:

$$\begin{aligned} D_j &= A_{jj} - \sum_{k=1}^{j-1} L_{jk} L_{jk}^* D_k \\ L_{ij} &= \frac{1}{D_j} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk}^* D_k \right) \end{aligned} \tag{3}$$

Si scriva, tramite un `Module`, una funzione che prende in input la matrice **A** e restituisce in output le matrici **L** e **D**.
Utilizzando questa funzione, si scomponga la matrice:

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix}$$

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.

Travelling salesman (il commesso viaggiatore)

1. Si scriva una classe `Location` che rappresenterà una destinazione (un luogo) in un mondo bidimensionale quadrato $[0, 1] \times [0, 1]$. Tra i membri private si mettano le due coordinate reali. Si scrivano uno o più costruttori che le assegnino, e si faccia in modo che se le due coordinate non vengono fornite come argomenti al costruttore questo dovrà assegnarle in modo casuale (entrambe con la misura piatta sui reali tra 0 e 1). [Può essere utile usare la funzione `drand48`. Non è richiesto che il costruttore controlli i bound.]

2. Si scrivano il costruttore di copie e l'overloading dell'operatore `<`, che confronta due destinazioni. La relazione d'ordine sarà semplicemente dettata dalla coordinata x: una posizione è minore di un'altra se e solo se lo è la sua ascissa.

3. Si scriva una funzione `distance` (membro pubblico di `Location`), che prenda come argomento un oggetto `l` di tipo `Location` e restituisca la distanza euclidea con `l`.

4. Si scriva una classe `Salesman` che rappresenterà un commesso viaggiatore. Tra i membri private si mettano il luogo iniziale del commesso e un `std::vector` di destinazioni da visitare. Tra i membri public si scriva un costruttore che richieda come parametri il luogo iniziale e due iteratori in un opportuno `std::vector`. Tale costruttore dovrà inizializzare il luogo iniziale e copiare gli elementi compresi tra i due iteratori nel vettore privato.

5. [facoltativo] Si riscriva il costruttore del punto precedente in versione template, in modo che accetti iteratori in altri tipi di container (come `deque` o `set`).

6. Si scriva una funzione `ndest` (membro pubblico di `Salesman`) che restituisca il numero di `Location` da visitare (esclusa quella iniziale).

7. Si implementi una funzione `visit` (anch'essa membro pubblico). Questa dovrà restituire la distanza totale percorsa dal commesso, nel caso in cui percorra le destinazioni nel loro ordine naturale, cioè quello definito dall'operatore `<`. Si usi l'algoritmo `std::sort`, che prende due iteratori random access e ordina (secondo `operator<`) il range compreso tra essi. Per risolvere i punti successivi, questa funzione dovrà avere comportamento polimorfico: la si dichiari in modo tale che questo avvenga.

8. Si commenti brevemente sulla possibilità di usare l'algoritmo `std::sort` con altri container, in particolare `deque`, `list`, `set`.

9. Nel `main`, si verifichi che la distanza totale percorsa per toccare 10 destinazioni random partendo dall'origine è inferiore al semiperimetro del quadrato solo in un caso su 300 circa.

10. Si scriva una classe `LazySalesman`, che erediti pubblicamente da `Salesman`. Questo tipo di commesso ama soffermarsi nei luoghi, e percorre una distanza aggiuntiva d per ognuna delle destinazioni che visita. Si metta la quantità d tra i membri `private` e si scriva un opportuno costruttore che la prenda come argomento. Tale costruttore dovrà, come per la classe base, richiedere anche i due iteratori, ma non il luogo iniziale, che deve essere posto all'origine $(0, 0)$.

11. Si implementi l'overriding della funzione `visit`, che restituisca la distanza percorsa totale tenendo conto delle distanze aggiuntive. Si abbia cura di riutilizzare, tramite chiamata a funzione, il codice già scritto per la classe base.

12. Nel `main`, istanziare un `std::vector` di `Location` e inizializzarlo con tre destinazioni poste in $(0.1, 0.1)$, $(0.5, 0.1)$, $(0.5, 0.5)$. Inoltre si istanzi un `std::vector` di puntatori a `Salesman`. Lo si riempia con i puntatori a tre oggetti allocati dinamicamente: un `Salesman` con locazione iniziale $(1, 1)$, un `LazySalesman` con distanza aggiuntiva 0.1, un `LazySalesman` con distanza aggiuntiva 0.5. Tutti e tre dovranno visitare le tre destinazioni inserite nel primo vettore. Si verifichi che la distanza più breve è percorsa dal secondo commesso. [Si abbia cura di liberare la memoria allocata.]