Esame di Laboratorio di Fisica Computazionale

2 maggio 2018, ore 9.30

1 Mathematica

1. **Matrici random.**
   Si generi una matrice 1000x1000 i cui elementi sono numeri complessi random, con parti reale e immaginaria comprese tra $[-1, 1]$. Si calcolino gli autovalori di questa matrice. Si disegnino gli autovalori nel piano complesso $z = (x, y)$, dopo aver eventualmente estratto da ciascun numero le rispettive parti reale e immaginaria.

2. **Insieme di Mandelbrot.**
   Sia data una successione di numeri complessi $z_i = z_{i-1}^2 + c$ con $z_0 = 0$ e $c \in \mathbb{C}$. Si generino le curve di livello che rappresentano, in funzione del parametro c, il numero minimo di iterazioni tale per cui $|z_i| > 2$. Si esegua il calcolo nel piano complesso c con $-0.6 \leq \mathrm{Re}(c) \leq -0.4$ e $0.4 \leq \mathrm{Im}(c) \leq 0.6$, utilizzando nel grafico 100 punti per ciascuna dimensione reale.

3. **Pendolo ordinario e pendolo di Kapitza.**
   Si consideri un pendolo di lunghezza l e massa unitaria, soggetto all'accelerazione di gravità g. Il punto di oscillazione del pendolo non è fisso, ma compie a sua volta uno spostamento lungo l'asse verticale, oscillante, con frequenza ν e ampiezza a. Chiamando $\phi(t)$ l'ampiezza delle oscillazioni del pendolo in funzione del tempo, questa soddisfa l'equazione differenziale

$$\ddot{\phi}(t) = -\frac{g + a\nu^2 \cos(\nu t)}{l} \sin \phi(t) \tag{1}$$

   (a) Si definisca una funzione che permetta di disegnare l'andamento dell'ampiezza ϕ in funzione del tempo $t \in [0, \bar{t}]$, per una data scelta dei parametri (l, a, ν) e per date condizioni iniziali $\phi(0) = x$ e $\dot{\phi}(0) = y$.
   Si aggiunga un argomento che permette di specificare il colore della curva, utilizzando l'opzione `PlotStyle -> RGBColor[r,g,b]` come argomento extra del comando `Plot`.

   (b) Si ponga $l = 1$ e si discuta il caso limite in cui $a = 0$. Si considerino le quattro seguenti condizioni al contorno:
   i) $x = 0$, $y = 0.01$
   ii) $x = \frac{\pi}{2}$, $y = 0.01$
   iii) $x = \pi$, $y = 0.01$
   iv) $x = \pi$, $y = 0$
   Si rappresentino le soluzioni nell'intervallo $t \in [0, 20]$.

(c) Si consideri il limite di piccole oscillazioni del punto precedente e si risolva quindi l'equazione differenziale

$$\ddot{y}(t) = -\frac{g}{l}y(t), \qquad \dot{y}(0) = 0.01, \qquad y(0) = \frac{\pi}{2} \tag{2}$$

Si confrontino nella stessa figura i grafici della soluzione $y(t)$ e della soluzione del caso *ii*) del punto precedente.

(d) Si consideri il potenziale effettivo

$$V_{eff} = -gl\cos(\phi_0) + \frac{a^2\nu^2}{4}\sin^2(\phi_0) \tag{3}$$

che descrive il comportamento del pendolo quando $a \ll l$ e $\nu \gg \omega_0$ in funzione dell'ampiezza di oscillazione ϕ_0, essendo $\omega_0 = \sqrt{g/l}$ la frequenza propria del pendolo semplice non forzato.
Si cerchi la condizione su ν in funzione degli altri parametri per l'esistenza di un minimo non banale del potenziale. Si determini un valore di $\bar{\nu}$ che soddisfa questa condizione, per $l = 1$, $a = 0.5$.

(e) Si consideri il caso di un pendolo di lunghezza $l = 1$, con $\phi(0) = \pi$ e $\dot{\phi}(0) = 0.0001$. Si considerino tre casi in cui $a = 0.01$ e la frequenza ν varia e assume i tre valori $\nu = (0.001, 0.01, 0.1)$. Si rappresentino le tre curve in una sola figura con $t \in [0, 20]$.
Analogamente si considerino tre casi in cui $a = 0.5$ e la frequenza ν varia e assume i quattro valori $\nu = (0.05, 0.5489, 4.907, \bar{\nu})$. Si rappresentino le quattro curve in una sola figura, questa volta con $t \in [0, 200]$.

2 C++: Esercizio

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.

Si vuole scrivere un semplice framework in cui sono rappresentati testi (come elenchi di parole) e regole di sostituzione, che modificano le parole sostituendo i caratteri.

1. Si scriva una classe `Text`, che rappresenterà un testo. Si metta tra i membri privati un `std::vector` di `std::string`, che conterrà le parole del testo. Si scriva un costruttore che prenda come argomento una stringa e inizializzi il testo con quella singola parola.

2. Si scriva il costruttore di copie; si scriva inoltre una *access function* che permetta di accedere alla variabile `std::vector` privata, ma solo in lettura.

3. Tra i membri pubblici, si scriva una funzione `print` che stampi il testo su `std::cout`, con le singole parole separate da spazi (e un `std::endl` alla fine). Si scriva inoltre una funzione `length` che restituisca il numero di parole nel testo.

4. Si scriva un costruttore di `Text` che prenda due iteratori dentro un `std::vector` di `std::string` e inizializzi il testo con gli elementi compresi tra il primo iteratore (compreso) ed il secondo (escluso).

   Bonus: si usi la sintassi della lista di inizializzazione, scrivendo un costruttore con corpo vuoto {}.

5. Si scriva una classe `Rule`, che rappresenterà una regola di sostituzione per i singoli caratteri di una parola, cioè un *vocabolario* che ad un carattere ne associa un altro (eventualmente uguale al primo). [Può essere utile ricordare che una `std::string` è un container di oggetti di tipo `char`.] Tra i membri privati si metta il container della STL che meglio si presta a rappresentare un vocabolario di questo tipo. Tra i membri pubblici, si scriva una funzione `add`, che prenda come argomenti due caratteri x e y e aggiunga nel vocabolario le due regole di sostituzione $x \mapsto y$ e $y \mapsto x$.

   Bonus: si scriva la funziona `add` usando l'idioma *named parameter*, in modo da poter concatenare le chiamate ad essa.

6. Tra i membri pubblici di `Rule`, si scriva una funzione `transform`, che prenda come argomento una stringa e restituisca la stringa modificata carattere per carattere secondo le regole nel vocabolario. Se un carattere non è presente nel vocabolario allora esso dovrà rimanere inalterato. Si vuole che questa funzione abbia comportamento polimorfico.

7. Si scriva una classe `XRule`, derivata da `Rule`, che rappresenterà una regola di sostituzione speciale: l'*override* della funzione `transform` dovrà agire come quella della classe base, ma sulla stringa con le lettere in ordine invertito (ad esempio su "olleh" invece che "hello"). Si abbia cura di *riutilizzare* il codice già scritto nella classe base, tramite chiamata a funzione. [Può essere utile usare l'algoritmo `std::reverse`, che inverte l'ordine degli elementi compresi tra i due iteratori che gli vengono passati come argomenti.]

8. Tra i membri della classe `Text`, si scriva una funzione `apply`, che prenda un oggetto di tipo `Rule` e lo usi per trasformare il testo parola per parola. Si vuole che, nel caso venga passato un oggetto di tipo derivato da `Rule`, il comportamento sia polimorfico.

   Bonus: si scriva il ciclo sulle parole come un *range-based for loop*.

9. Nel `main`, si istanzi un oggetto di tipo `XRule` e si inseriscano nel suo vocabolario le seguenti sostituzioni: `a` $\leftrightarrow$ `o`, `m` $\leftrightarrow$ `n`, `g` $\leftrightarrow$ `p`, `l` $\leftrightarrow$ `r`. Si istanzi poi un oggetto di tipo `Text` inizializzato con un vettore contenente le parole "programmazione", "in", "linguaggio", "multiparadigma". Si trasformi il testo usando la regola appena creata e si stampi a schermo il testo trasformato.

10. Si scriva una funzione globale `longest`, che prenda come argomenti due iteratori dentro `std::list<Text>` e restituisca il testo piú lungo (come numero di parole) compreso tra i due. Si usi il seguente algoritmo della STL:

    `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.

11. Si renda *generico* il codice del punto precedente, in modo che la funzione `longest` possa essere chiamata passandole qualunque tipo di iteratori a oggetti di tipo compatibile con `Text` (non necessariamente appartenenti ad un `std::list`).

3 Python

Il comando Unix:
`find   -size +1M -ls > all_big_files.txt`
genera il file di testo "all_big_files.txt" in cui ad ogni riga sono contenute varie informazioni su ogni file che nelle propria $HOME supera la taglia di 1 Mbyte. Si scriva uno script in Python che esegua questo comando e che dal file all_big_files.txt estragga solo le linee relative a tutti i files che terminano col suffisso .txt e che infine scriva queste linee complete in un file di testo il cui nome passato direttamente da riga di comando.