# Esame di Laboratorio di Fisica Computazionale

**19 aprile 2017, ore 9.30**

# 1 Mathematica

1. Si calcoli la soluzione del seguente problema differenziale

$$y'(x) = 3y(x) + 1, \qquad y(0) = 2 \quad . \tag{1}$$

2. Si espanda la soluzione del punto precedente in serie di potenze di $x$ attorno al punto $x_0 = 0$, fino all'ordine $n$ con $n = 0, 1, 2, 3$. Si disegni in un solo grafico, per ciascuna approssimazione, lo scarto percentuale dalla soluzione esatta.

3. Si definiscano $\phi_n$ le approssimazioni successive di un problema di Cauchy

$$y'(x) = f(x, y), \qquad y(x_0) = y_0 \tag{2}$$

con

$$\phi_0(x_0) = y_0, \qquad \phi_n(x) = y_0 + \int_{x_0}^{x} dt\, f(t, \phi_{n-1}(t)) \tag{3}$$

Si consideri il problema del punto 1. e si confrontino con la sua soluzione esatta le soluzioni $\phi_n$ ottenute ponendo $n = 0, 1, 2, 3$ e identificando $f(x, y)$ con il secondo membro dell'equazione .

4. Si considerino le due superfici $s_1$ e $s_2$

$$s_1 : \quad x^2 + 2x\, y = z - y^2 \tag{4}$$
$$s_2 : \quad \lambda - 3x^2 - z = 6y^2 \tag{5}$$

   - Si mostrino le due superfici nello stesso grafico, disegnate con $\lambda = 5$ e per $x \in [-3, 3]$ e per $y \in [-3, 3]$.
   - Si calcoli il valore di $\lambda$ per cui le due superfici hanno un solo punto in comune

5. L'insieme dei numeri complessi diversi da zero forma un gruppo rispetto alla moltiplicazione tra numeri complessi, definita come $(a + i\, b)(x + i\, y) = [(ax - by) + i(ay + bx)]$, con $a, b, x, y \in \mathbb{R}$.
   È possibile associare a ogni numero complesso una matrice 2x2, secondo la mappa $\phi$

$$\phi(x + i\, y) = \begin{pmatrix} x & -y \\ y & x \end{pmatrix} \tag{6}$$

   Si introducano delle funzioni che permettono di verificare a livello numerico, sfruttando la rappresentazione matriciale, le proprietà gruppali: chiusura, esistenza dell'identità, esistenza dell'inverso, associatività.

6. Si scrivano, **senza** utilizzare cicli `for` o il comando `Sum`, tre funzioni che calcolino *i*) la deviazione standard di una lista di valori, *ii*) la covarianza e *iii*) la correlazione tra due liste di valori.
   *Suggerimento: data una lista A, si calcoli la somma dei suoi elementi con il comando* `Apply[Plus, A]`.
   Si calcoli la correlazione tra 10000 valori random reali compresi tra 0 e 1 e 10000 valori random interi compresi tra -1 e 1.

7. Si consideri il sistema di equazioni

$$\begin{cases} x'(t) &= 10\,(y(t) - x(t)) \\ y'(t) &= 28\,x(t) - x(t)\,z(t) - y(t) \\ z'(t) &= x(t)\,y(t) - 8/3\,z(t) \\ x(0) &= x_0, \qquad y(0) = y_0, \qquad z(0) = z_0 \end{cases}$$

   - Si definisca una funzione dei tre parametri $(x_0, y_0, z_0)$ che risolva **numericamente**, per $t \in [0, 30]$, il sistema di equazioni differenziali
   - Si definisca una funzione che disegna, in funzione dei tre parametri $(x_0, y_0, z_0)$, la curva parametrica descritta per $t \in [0, 30]$ dalle tre funzioni $(x(t), y(t), z(t))$.
   - Si scelgano a piacere due insiemi di condizioni al contorno e si disegnino i risultati corrispondenti.
     *Avvertenza: potrebbe essere necessario incrementare il parametro* `MaxSteps` *all'interno di* `NDSolve`.

# 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 per gestire operazioni di trasformazione (*hashing*) tra stringhe di caratteri.

1. Si scriva una classe `Task`, che rappresenterà una stringa da trasformare. Si metta tra i membri *private* la stringa stessa (una variabile di tipo `std::string`). Si scriva un opportuno costruttore che la inizializzi, con la stringa vuota "" come *valore di default*.

2. Si scriva il costruttore di copie di `Task` (usando la sintassi della lista di inizializzazione) e una *access function* per leggere la stringa. Si scriva inoltre un particolare costruttore che prenda due `Task` e ne costruisca uno nuovo la cui stringa sia la concatenazione delle due (l'operatore "+" fra stringhe fa esattamente questo).

3. Nel main, si provi la funzionalità di questo costruttore, stampando le stringhe dei due oggetti singoli e poi quella dell'oggetto concatenato.

4. Si scriva una classe `Hasher`, che si occuperà di trasformare una stringa in un'altra, attraverso un certo numero di iterazioni di una operazione detta *hash*. Tra i membri *private* si metta il numero di iterazione da eseguire e si scriva il costruttore, con *valore di default* 2.

5. Si scriva la *helper function* `hash`, membro privato di `Hasher`, che trasformi singolarmente ogni carattere $c$ della stringa nel seguente modo:

$$\begin{aligned} c &\mapsto 5c/4 - 25 \quad \text{se } c \text{ divisibile per } 4 \\ c &\mapsto c - 1 \qquad\quad \text{altrimenti} \end{aligned}$$

   [Nota: `std::string` è un container di oggetti di tipo `char`, e si comporta come gli altri container della libreria standard; in particolare, supporta i *range-based for loops* e gli iteratori.]

6. Si scriva l'operatore "()", in modo che un oggetto di classe `Hasher` possa essere chiamato come se fosse una funzione, passando come parametro un oggetto di tipo `Task`, e restituisca la stringa modificata come descritto al punto 5. Si vuole che questa funzione abbia comportamento polimorfico.

7. Si scriva una classe `ShiftHasher`, derivata di `Hasher`, il cui numero di iterazioni sarà fissato a 1. Si scriva l'opportuno costruttore e l'`override` dell'operatore "()". Il comportamento di quest'ultimo sarà quello di eseguire l'operazione implementata in `Hasher`, solo dopo aver incrementato ogni carattere della stringa di un valore uguale alla sua posizione all'interno della stessa (il primo carattere rimane uguale, il secondo incrementa di 1, il terzo di 2, etc.) [Si riutilizzi il codice già scritto per la classe base, attraverso una chiamata a funzione.]

8. Per controllare il comportamento polimorfico, nel *main* si istanzi un `std::vector` che contenga un `Hasher` e uno `ShiftHasher` allocati dinamicamente [si consideri se usare valori, puntatori o referenze]. Si applichino quindi questi oggetti ad un medesimo `Task` per controllare che le stringhe generate siano diverse. [Ci si ricordi di deallocare gli oggetti allocati dinamicamente.]

9. Si scriva un *class template* `Queue`, che rappresenterà una coda di oggetti su cui eseguire delle operazioni, parametrico nel tipo dell'oggetto contenuto nella coda (che poi sarà `Task`), e nel tipo dell'oggetto che esegue l'operazione (che poi sarà `Hasher`). Il template dovrà avere, tra i membri *private*, un `std::set` contenente gli oggetti in coda (per valore). [Non è necessario scrivere il costruttore.]

10. Si vorrà che i `Task` contenuti nella coda siano ordinati lessicograficamente (è ciò che fa l'operatore "<" tra oggetti di tipo `std::string`). Implementare ciò che manca alla classe `Task` per avere questa funzionalità.

11. Implementare, tra i membri *public* di `Queue`, due funzioni:

    - `add`, che prenda come parametro un oggetto, e lo aggiunga alla coda [può essere utile usare la funzione `insert` di `std::set`, che inserisce il suo argomento nell'insieme]
    - `execute`, che esegua il primo task nella coda ordinata lessicograficamente, lo elimini dalla coda e infine restituisca la stringa modificata dall'oggetto che si occupa della trasformazione [può essere utile la funzione `erase` di `std::set`, che vuole come argomento un iteratore ed elimina il corrispondente oggetto]

    Provare, con qualche riga nel *main*, la funzionalità di questo template, istanziandolo con `Task` e `Hasher` come parametri.

# 3 Shell scripting

Con uno script si cancellino le linee dispari dal file allegato `dati.dat`.