Università degli Studi di Trento
Laurea Triennale in Fisica

Programmazione Scientifica - Scritto

Teachers: *Prof. L. Tubiana, Prof. A. Roggero, Guglielmo Grillo*
Data riconsegna: *June 1, 2025*

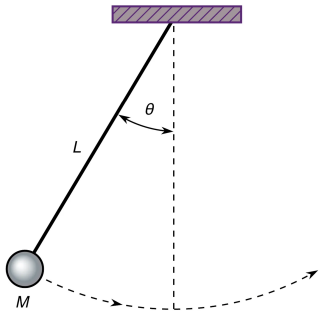
1 Introduzione

Come accennato durante l'introduzione al corso, è possibile superare automaticamente lo scritto svolgendo e consegnando alcune prove intermedie dette *Compitini*.
Come nel caso precedente, per risolvere i compitini potete usare **solo quanto imparato a lezione**.
L'obiettivo del compitino è implementare una simulazione numerica per il moto di un pendolo senza l'ipotesi di piccole oscillazioni e analizzarne i risultati, verificandone anche la correttezza nel caso limite di piccole oscillazioni.

Il compito è diviso in 3 sottoparti per un totale di 33 punti: i) simulazione del moto di un pendolo in C (20 punti + 3 punti), ii) analisi in Python e verifica della correttezza delle analisi (7 punti), iii) Identificazione del passo massimo di integrazione (3 punti).

2 Simulazione in C

Scrivere un programma in C che risolva numericamente l'equazione del pendolo prendendo come variabile cinematica l'angolo θ:



$$\begin{cases} F_{\parallel} = mg\cos(\theta) \\ F_{\perp} = mg\sin(\theta) \\ F_{\perp} = -mL\ddot{\theta} \end{cases} \implies \ddot{\theta} = -\frac{g}{L}\sin(\theta)$$

Per far evolvere numericamente un'equazione differenziale si usa un'*integratore numerico* che date velocità, posizioni, e accelerazioni al tempo t permette di calcolarle al tempo $t + \Delta t$, con Δt un intervallo di tempo piccolo. Come integratore vi chiediamo di utilizzare

il velocity-verlet. Date la posizione e la velocità all'istante temporale discreto t_i, si calcola la posizione e la velocità al passo successivo $t_{i+1} = t_i + \Delta t$ usando la seguente procedura:

1. Calcolate la nuova posizione come $\theta(t_{i+1}) = \theta(t_i) + \omega(t_i)\Delta t + \frac{1}{2}\alpha(t_i)\Delta t^2$
2. Calcolate l'accelerazione associata alla nuova posizione $\alpha(t_{i+1}) = F(x_{i+1})/m$
3. Calcolate la nuova velocità $\omega(t_{i+1}) = \omega(t_i) + \frac{1}{2}\left(\alpha(t_i) + \alpha(t_{i+1})\right)\Delta t$

Dove abbiamo posto $\omega(t) = \dot{\theta}(t)$ (velocità angolare) e $\alpha(t) = -\frac{g}{L}\sin(\theta(t))$ (accelerazione angolare). Per i curiosi, l'integratore è spiegato con qualche dettaglio in più in appendice.

2.1 Programma minimo (18 punti)

Scrivete un codice C in grado di simulare il moto del pendolo prendendo il seguente file di input e producendo gli output come descritto sotto.

2.2 Input

Il vostro programma dovrà essere in grado di leggere le configurazioni di partenza da un file. I file di input devono seguire il seguente formato **in cui l'ordine delle righe può essere cambiato**:

```
# Condizioni iniziali per l'integratore
# Lunghezza della simulazione in passi; float
T 10000
# Incremento temporale; float
dt 0.001
# Angolo iniziale rispetto alla normale in radianti; float
theta0 1.07
# Velocita' angolare iniziale; float
omega0 0
# Lunghezza della fune; float
L 1
# Massa; float
m 1
# Accelerazione di gravita'; float
g 9.81
# Tempo di salvataggio, rappresenta ogni quanti passi salvare l'output su file; intero
ts 100
# Nome file in output
out pendolo0_1.csv
```

Le righe che cominciano per "#" sono commenti e vanno ignorate.

2.2.1 Output

Il vostro programma dovrà essere in grado di salvare l'output su di un file in formato csv. Ogni riga deve riportare (in questo ordine):

1. passo di integrazione i
2. tempo corrispondente t_i
3. valore dell'angolo $\theta(t_i)$
4. valore della velocità $\omega(t_i)$
5. valore dell'accelerazione $\alpha(t_i)$

L'output deve inoltre contenere un header, cioè una serie di righe in cima al file contenenti i dati su come generarli. Queste righe inizieranno con "#HDR" (in modo da essere ignorate da numpy) e dovranno contenere i parametri fisici in input per generare il file di output: T, dt, theta0, etc. Ad esempio:

```
#HDR T 1000
#HDR dt 0.001
#HDR theta0 1.07
#HDR omega0 0
#HDR L 1
#HDR m 1
#HDR g 9.81
#passo tempo   theta     w       alpha
0       0.0     1.07     0      -2.712
10      0.01    ...      ...     ...
20      0.02    ...      ...     ...
```

2.3 Input da argomenti (2 punti)

Scrivere il codice in modo tale che i parametri *theta*0 e *omega*0 possano essere passati come argomenti da linea di comando, e che in tal caso i valori passati come argomenti sovrascrivano quelli letti dal file all'interno del vostro programma.

2.4 Integratore generico (3 punti)

Scrivete la funzione che implementa l'algoritmo di velocity verlet in modo che sia più generica possibile ed in particolare non faccia assunzioni al suo interno su quali sono le forze che determinano le accelerazioni del sistema, prendendo come parametro in input la funzione che le calcola.

3 Test e analisi del codice (6 punti)

Scrivete uno o più file di partenza con angoli piccoli (vedi Fisica I) e simulate almeno dieci oscillazioni complete del pendolo. Utilizzando uno jupyter notebook, caricate il file csv generato e verificate che per piccole oscillazioni la traiettoria sia quella ottenibile analiticamente:

$$\theta(t) = \theta_0 \cos(\Omega t) + \frac{\omega_0}{\Omega} \sin(\Omega t)$$

con $\Omega = \sqrt{\frac{g}{L}}$.
Per verificare che il vostro codice funzioni fate i seguenti test:

- Plottate la funzione $e(t) = \frac{\theta_{\text{teorico}}(t) - \theta_{\text{numerico}}(t)}{\max_t[\theta_{\text{teorico}}(t)]}$, dove al denominatore c'è l'angolo massimo raggiunto secondo la formula delle piccole oscillazioni in un periodo.
- Tracciate il grafico nello spazio delle fasi, fate cioè un grafico mettendo sull'asse delle x l'angolo $\theta(t)$ e sull'asse delle y la corrispondente velocità angolare $\omega(t)$ della simulazione. Sovrapponete ad essa lo stesso grafico ottenuto utilizzando i valori teorici e disegnato con una linea tratteggiata (usate '-' come ultimo argomento di pyplot.plt). I due grafici dovrebbero coincidere.
- Calcolate l'energia totale del vostro sistema (cinetica + potenziale) e verificate che sia costante a meno di fluttuazioni ed uguale a quella teorica entro un'incertezza accettabile.

3.1 Regime non lineare (4 punti)

Simulare il seguente sistema al variare dell'angolo di partenza $\theta_0 \in \left[\frac{\pi}{36}; \frac{\pi}{2}\right]$, ripetere il calcolo dell'energia totale e dell'errore $e(t)$ e relativi plot di cui al punto precedente, e tracciare il grafico delle fasi $\theta(t), \omega(t)$ per almeno 10 valori di θ_0:

```
T 10
dt 0.001
theta0 ???
omega0 0
L 1
m 1
g 1
ts 10
out pendolo_theta???.csv
```

4 Consegna Extra (3 punti)

Simulare il pendolo in regime non lineare per varie condizioni iniziali e passo temporale dt. Identificate il passo temporale dt più grande che fornisce una simulazione stabile e cercate di motivare il perchè della vostra scelta.

5 Consegna del codice

Per portare a termine con successo il compitino andrà consegnato un file compresso nome_gruppo.tar.gz contenente i file sorgente, gli header c, un file di testo contenente una piccola descrizione delle scelte fatte nello strutturare il codice, le immagini generate e tutti i file di input correlati.

6 Appendice

6.1 Valutazioni sul codice

Valuteremo le seguenti caratteristiche. Avete molta libertà su come organizzare le cose, qual è il metodo migliore in base ai principi visti a lezione?

Modularità. Deve essere composto da librerie/moduli e funzioni il più possibile modulari. Questo vale sia per C che per Python.

Robustezza. Il codice deve controllare che i valori forniti in input siano compatibili con le assunzioni del programma (es: come si verifica che il numero di step richiesto non dia overflow?).

Efficienza. Le operazioni costose vanno evitate, incluso un'uso inopportuno della memoria, o il duplicare calcoli. Sapete trovare un sistema efficiente per evitare di calcolare due volte le forze ad ogni passo? Nota: un sistema - non l'unico- può essere usare una variabile `static` nell'integratore (vedi libri di testo suggeriti).

Mantenibilità. Il codice deve essere ben documentato e soprattutto le funzioni, variabili, e strutture devono avere nomi sensati. Le funzioni non devono avere comportamenti impliciti e devono fare una sola cosa.

Allocazione della memoria. Le dimensioni N possono variare, ed il programma deve tenerne conto. Qual è il modo più opportuno di farlo? Non si possono allocare array troppo grandi sullo stack.

Variable length arrays. [1] L'uso dei variable length arrays (es. `double vec[n];` con `n` definito all'interno della stessa funzione) non è permesso.

6.2 Algoritmo velocity-Verlet

Per il caso di un pendolo si può usare l'algoritmo velocity-verlet per integrare le equazioni del moto e calcolare la traiettoria. Questo algoritmo è molto utilizzato per integrare le equazioni del moto di sistemi in cui l'energia si conserva (non funziona ad esempio nel caso di un pendolo forzato e smorzato..).
Data un'equazione differenziale al secondo ordine

$$m\frac{d^2x(t)}{dt^2} = f(x(t)), \tag{1}$$

possiamo riscriverla come un sistema di due equazioni al primo ordine

$$\begin{cases} \frac{dx(t)}{dt} = v(t) \\ \frac{dv(t)}{dt} = \frac{1}{m}f(x(t)) \end{cases} \tag{2}$$

L'algoritmo di velocity-Verlet consiste nell'evolvere avanti nel tempo le due funzioni $x(t)$ e $v(t)$ con la seguente regola

$$\begin{cases} x(t_0 + \Delta t) = x(t_0) + \Delta t v(t_0) + \frac{\Delta t^2}{2m}f(x(t_0)) \\ v(t_0 + \Delta t) = v(t_0) + \frac{\Delta t}{2m}\left(f(x(t_0)) + f(x(t_0 + \Delta t))\right) \end{cases} \tag{3}$$

dove Δt è il passo temporale utilizzato.

[1] `https://en.wikipedia.org/wiki/Variable-length_array`