Relazione sul Biliardo Triangolare

Vincenzo Achille

5 Novembre 2025

1 Fisica del Biliardo e Tecniche utilizzate

Il programma realizzato prevede la simulazione di un biliardo triangolare come quello riportato in Fig.1. La dinamica descrive il lancio di una pallina a partire dalla posizione $(0, y_0)$ dove y_0 è una generica altezza per cui il centro della pallina è compreso tra $[-r1, r1]$. La pallina possiede un angolo θ_0 compreso tra $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ dove per $y_0 = 0$ e $\theta_0 = 0$ ho l'asse di simmetria del biliardo ed il moto in direzione orizzontale. Nell'esecuzione del programma la pallina sarà lanciata sempre verso destra. La fisica del sistema prevede degli urti elastici con i bordi del biliardo: l'angolo di incidenza sarà uguale a quello di riflessione come in Fig.**??**. Il sistema sarà caotico. La pallina inoltre, potrà uscire sia da destra che da sinistra.

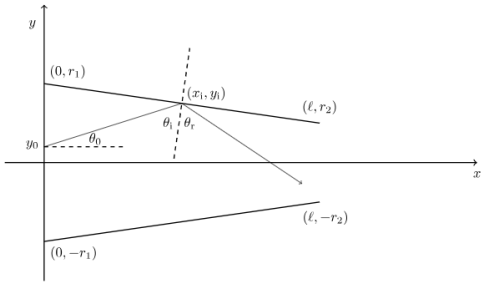

Figura 1: Geometria del sistema fisico realizzato. Dalla figura è possibile vedere cosa sono le grandezze r_1, r_2, l

Per realizzare il moto è stata utilizzata una rappresentazione basata sui coefficienti angolari e sulle coordinate dell'urto (le traiettorie saranno rettilinee).

I parametri che descriveranno una pallina saranno quindi il punto di partenza (che sarà il punto di partenza per il primo lancio e l'ultimo punto di impatto per i successivi) e il coefficiente angolare della retta lungo cui si muove. Da questi 3 parametri sarà possibile ricostruire la traiettoria lungo la quale la pallina si muoverà fino al prossimo urto. Per avere la legge oraria (che differisce per il verso di percorrenza della retta) è stata introdotta la variabile direction che sarà +1 se verso destra, -1 se verso sinistra o 0 per moti verticali. Posti quindi x_i e y_u come posizioni d'urto ed m come coefficiente angolare la traiettoria sarà:

$$y = m(x - x_i) + y_u \tag{1}$$

Il tempo indicato con t aumenta sempre. Se la pallina si muove verso destra avremo x = t. Verso sinistra non è sufficiente sostituire x con -x, ma sarà necessario introdurre un parametro ausiliario detto T che corrisponde alla posizione iniziale e sostuire dentro la eq.1 T-t a t. In questo modo, nel secondo caso, il segno del coefficiente angolare sarà preservato ed il nuovo x diminuirà.
Per realizzare la condizione d'urto vengono calcolati la posizione d'urto (interesezione tra la traiettoria e l'equazione del biliardo) e il punto della pallina che urterà. Tale punto è dato dall'intersezione, presa nel verso di percorrenza, tra il cerchio e la traiettoria. Tale condizione approssima la pallina ad un corpo rigido pur essendo una condizione approssimata, e non esatta. La perdita di precisione si avrà in particolare per palline con coefficiente angolare vicino a quello appartenente all'equazione del biliardo. Abbassando il raggio la pallina tenderà a diventare un punto materiale rendendo trascurabili tali approssimazioni.
Scopo del programma saranno sia una simulazione grafica di un singolo lancio con parametri iniziali y_0 e θ_0 a scelta che la generazione di N tentativi con parametri iniziali distribuiti gaussiani nei loro intervalli di definizione.
A posteriori la rappresentazione basata sui coefficienti angolari presenta una grave criticità che non risulterebbe in una rappresentazione di tipo vettoriale. Per angoli vicini a $\pm\frac{\pi}{2}$ la tangente dovrebbe tendere a ∞ (moti verticali). Poichè tale valore non è rappresentabile, utilizzando la funzione tangente avremo un valore limite con un corrispondente angolo limite. Il tutto risulterà in una perdita di precisione. Palline con tali angoli sono segnalate con un errore nel lancio singolo e scartate negli N lanci. Nonostante ciò non compromettono la statistica perchè presentano una frequenza di occorrenza

bassa rispetto al campione.

2 Implementazione

Il programma è stato implementato tramite 2 file di intestazione e 1 di implementazione.

- file ball.cpp/.hpp contengono la dichiarazione e la definizione della classe ball che realizza tutte le funzionalità dinamiche del biliardo. Questa continene come attributi m_x, m_y, m_m e m_direction che sono i già discussi parametri dell'implementazione dinamica. La classe possiede anche una serie di attributi privati static che contengono le informazioni sulla geometria del biliardo: ossia m_l, m_r1, m_r2 che possono essere visualizzati in figura. Tra i membri static figurano anche m_radius (il raggio della pallina), m_window (le dimesioni della finestra) e m_center (posizione del centro del biliardo) si intende il punto di coordinate (0,0) nella figura Fig.1.
  Infine è presente m_errorTolerance che entra in gioco nella condizione d'urto. Questo è la discrepanza massima tra il valore di impatto calcolato e il punto della pallina. Quando la loro differenza è minore di m_errorTolerance la condizione d'urto viene triggerata. Questo parametro risulta necessario perchè durante l'evoluzione temporale sono presenti dei salti dovuti all'incremento del tempo t citato prima. In tale modo è possibile porre un range entro cui viene eseguita la condizione d'urto. Questo parametro è direttamente legato alla velocità di simulazione, infatti,ll'aumentare di quest'ultima, la distanza tra un incremento e l'altro aumenta portando alla possibilità di non innescare la condizione d'urto. Tale relazione si può esprime analiticamente. Con m_errorTolerance di default si consiglia di non portare animationSpeed non oltre i 5.
  Tale classe contiene il costruttore parametrico che inizializza le palline effettuando dei controlli sugli intervalli dei parametri.. I metodi pubblici della classe sono ballDynamics() e ballDynamicsAnimated(float animationSpeed). Il primo gestisce l'evoluzione dinamica della pallina tramite una serie di metodi privati restituendo i valori finali di y_f

e di θ_f, il secondo esegue le stesse funzionalità con aggiunta della simulazione grafica. ballDynamics() sarà quindi chiamato durante gli N lanci mentre ballDynamicsAnimated() durante il lancio singolo. Tra i metodi privati centrali vi sono i metodi update(float x, float y) e impactPosition(). Il primo si occupa di aggiornare un oggetto ball dopo un urto. impactPosition() calcola la posizione dell'urto date tali condizioni iniziali. I restanti metodi sono volti alla corretta implementazione di update() secondo la logica precedentemente descritta.

- il file statisticsRoot.hpp contiene invece il metodo statistics(int N, float $\bar{y}_0$, float $\bar{\theta}_0$, float σ_{y_0}, float σ_{θ_0}, bool isDiscarded) che genera la pallina N volte e ne esegue l'analisi statistica sovracitata. Tramite isColliding è anche possibile scartare le palline che escono a sinistra.

  I file statisticsRoot contengono un unico metodo, ossia statistics() che esegue N lanci date le medie e le deviazioni standard dei parametri iniziali y_0 e θ_0. Contiene anche una variable bool che, se vera, consente di scartare i lanci che rimbalzano verso sinistra (ossia con m_direction finale = -1).

La compilazione è stata svolta utilizzando CMake, un sistema di build open-source che semplifica la gestione dei progetti software. Tramite file di configurazione dichiarativi, permette di generare automaticamente i file di build specifici creando un unico eseguibile e garantendo portabilità e mantenibilità del codice. Le librerie da installare per un corretto funzionamento sono SFML (che si occupa di gestire l'interfaccia grafica della simulazione) e ROOT (che si occupa, per gli scopi del programma, delle funzioni che implementano i metodi statistici). Il programma presenta alcuni memory leak legati ad SFML.

Una volta avviato il programma dal terminale sarà possibile decidere se scegliere il lancio singolo che utilizzerà come parametri di input la y_0 e la θ_0 o se scegliere gli N lanci che richiedono come parametri le medie $\bar{y}_0$ e $\bar{\theta}_0$, le deviazioni standard σ_{y_0} e σ_{θ_0} di y_0 e θ_0, il numero N di lanci e un bool che se vero scarta le palline che escono a sinistra. Il primo programma da' come output i valori finali y_f e θ_f oltre alla simulazione. Invece il secondo restituisce gli istogrammi delle y_f e delle θ_f insieme ai loro valori medi, le deviazioni standard e i coefficienti di asimmetria e di curtosi.
Per testare correttamente il programma è stato controllato il funzionamento dei costruttori e dei metodi setter verificando l'inizializzazione per alcuni va-

lori scelti casualmente e per valori vicini ai limiti degli intervalli di validità. Per quanto riguarda il moto sono state prese delle condizioni iniziali casuali e delle condizioni vicine ai valori di validità e sono poi state verificate in 3 regimi differenti: ossia per $r_1 > r_2$, $r_1 < r_2$ ed $r_1 = r_2$.
A supporto della realizzazione del programma è presente inoltre una pagina di github al seguente link: https://github.com/VincenzoAchille/Biliardo-Triangolare

3 Risultati

Il lancio singolo conferma la corretta esecuzione del programma (salvo problemi discussi precedentemente riguardanti il corpo rigido che sono solo grafici e dei moti verticali) mostrando la caoticità del moto. Dall'analisi statistica emerge che fissati r_1 ed r_2 con $r1 > r2$ per l dell'ordine di $|r_1 - r_2|$ i y_f ed i θ_f si ottengono delle distribuzioni gaussiane (come nel caso delle distribuzioni dei valori iniziali) All'aumentare di l (ossia per biliardi tendenti ad un biliardo non triangolare) avremo che la distribuzione degli y_f tenderà ad una distribuzione uniforme tra r_2 e $-r_2$ mentre la distribuzione dei θ_f tenderà ad una figura di interferenza con invluppo gaussiano. La prima può essere spiegata osservando che nel limite di $l \to +\infty$ il biliardo sarà piatto. In tale regime tutti i valori escono nel $min(r1, r2)$ (che diventerà l'altezza del biliardo) per cui vengono esclusi i valori tra r_2 ed r_1. Per gli angoli finali la distribuzione sembra manifestare un regime di interferenza. Ed alcuni angoli saranno vietati. Esistono delle zone di interferenza dove il valore non raggiunge lo 0. Si suppone che sia dovuto ad un imperferfezione del codice poichè per tali angoli sono molto vicino alla situazione di moto verticale.