Sebastiano Zagatti

Relazione Esame Laboratorio di Calcolo

PROBLEMA:

Si chiede di calcolare  la traiettoria di una particella che si muove in 2 dimensioni in una scatola di forma ellittica  a parete perfettamente riflettente (urti elastici) supponendo cha all' istante iniziale parta da un punto (x\_0,y\_0) nella scatola e che le pareti siano individuate dal profilo di equazione (x/a)\*\*2+(y/b)\*\*2=1.

Il compito puo' essere svolto con un programma che, per una data posizione e velocita', determini dove e quando avverra' la prossima collisione con la parete (tra le collisioni la particella si muove di moto rettilineo uniforme, con modulo della velocita' pari a 1, stante la condizione di urti elastici).

In pratica, immaginiamo di avere le coordinate (x,y) e l' angolo alpha che il vettore velocita' (v\_x=cos(alpha),v\_y=sin(alpha)) forma con l' asse x positivo ad un dato istante.

La condizione che la particella sia nella scatola e' che (x/a)\*\*2+(y/b)\*\*2<1.

La prima cosa da fare e' di determinare l' istante della prossima collisione risolvendo per t l' equazione di secondo grado

(x(t)/a)\*\*2+(y(t)/b)\*\*2=1

con

x(t) = x + v\_x\*t   [1]

y(t) = y + v\_y\*t

e prendendo la maggiore delle due radici (questo serve ad evitare possibili problemi di arrotondamento). Sia tale valore tc.

La posizione della prossima collisione (xn,yn) sara' allora data da [1] con t=tc.

Il vettore  velocita' dopo l'urto (v') sara' invece dato da

v' = v - 2 (v.n)n

dove v e' il vettore prima dell' urto, (v.n) rappresenta il prodotto scalare tra v e il vettore (versore) n, normale all' ellisse nel punto di impatto (xn,yn), di coordinate cartesiane:

        nx=xn/(a\*a)/sqrt( (xn/(a\*a))\*\*2+ (yn/(b\*b))\*\*2 )

        ny=yn/(b\*b)/sqrt( (xn/(a\*a))\*\*2+ (yn/(b\*b))\*\*2 )

Basta iterare l' algortmo per il numero prescelto di collisioni e scrivere le coppie di coordinate dei punti di collisione su un file.

Per graficarlo con gnuplot(supponendo che le coordinate siano le prime 2

colonne di un file fort.1), si puo' dare il comando

set sample 10000

p b\*sqrt(1-(x/a)\*\*2), -b\*sqrt(1-(x/a)\*\*2),'fort.1'

Gioca qualche ruolo importante la precisione con cui si rappresentano i dati ?

RISOLUZIONE:

Il sorgente Fortran che ho scritto utilizza due moduli:

* Il modulo “prec” che permette di fissare il kind RK delle variabili reali e di modificarlo in modo semplice;
* Il modulo “exam” che contiene due funzioni che permettono di calcolare le componenti della posizione della particella ad un determinato istante di tempo.

Per semplicità ho scelto di dichiarare come array di rango 1 ed estensione 2 la velocità e il versore n, così da poter utilizzare la funzione intrinseca dot\_product, mentre ho preferito mantenere separate le componenti della posizione per facilitarmi la scrittura del sorgente. La variabile intera “i” costituisce un contatore per un ciclo do, mentre la variabile “K” è il numero di collisioni che vogliamo considerare.

Il significato delle altre variabili reali è spiegato nella descrizione del programma.

Il programma riceve come input i valori dei semiassi dell’ellissi (a, b), la posizione iniziale della particella (xo, yo), il valore dell’angolo α (alpha) che individua la direzione della velocità (il cui modulo è sempre uno) e il numero di collisioni (K) che vogliamo considerare; esegue quindi un primo controllo sui dati riguardanti la posizione, segnalando un errore nel caso in cui la posizione iniziale della particella sia esterna all’ellissi o appartenente al suo bordo (la condizione di appartenenza alla scatola ellittica è ).

L’angolo alpha è richiesto in gradi, poiché tale valore è più semplice da dare come input dalla tastiera; esso viene poi trasformato in radianti (rad), così da poter essere utilizzato dalle funzioni intrinseche seno e coseno che permettono il calcolo delle due componenti del vettore velocità, per fare questa operazione ho utilizzato la proporzione che permette di trasformare in angoli in radianti gli angoli in gradi e ho dovuto definire il parametro reale pi, che esprime il valore di π tramite la funzione intrinseca acos.

Segue poi l’algoritmo che il problema chiede di implementare:

* Si calcolano i termini: “primo”, “secondo e “terzo”, che corrispondono ai coefficienti A,B,C dell’equazione di secondo grado: , necessaria per trovare gli istanti a cui avvengono le collisioni, tali termini sono stati trovati da me per via analitica e sono poi stati utilizzati nel programma;
* Utilizzando la classica formula di risoluzione delle equazioni di secondo grado:, si trovano i due valori t1, t2, poi tramite due istruzioni IF, viene selezionata la maggiore delle due soluzioni il cui valore viene assegnato alla variabile t;
* Utilizzando la variabile t, vengono calcolate, tramite le funzioni del modulo exam, le coordinate del punto d’impatto della particella con l’ellissi, tali coordinate dovranno costituire il punto di partenza per la successiva iterazione del ciclo, quindi, evitando di definire ulteriori variabili, vengono ridefiniti i valori di xo e yo;
* Si calcolano ora le due componenti del versore n, normale all’ellissi nel punto di collisione, implementando le formule fornite nel testo del problema;
* Sempre utilizzando la formula fornita nel testo e sfruttando la funzione intrinseca dot\_product si calcola la nuova velocità v’; come per la posizione, anche la nuova velocità costituirà la velocità di partenza dello step successivo del ciclo, per cui viene ridefinito l’array v;
* I valori della posizione d’impatto vengono scritti su un file “fort.1”.

Tale algoritmo viene infine iterato utilizzando un ciclo do che si ripete per il numero di collisioni che si vuole considerare.

Utilizzando Gnuplot e l’istruzione:

set sample 10000

p b\*sqrt(1-(x/a)\*\*2), -b\*sqrt(1-(x/a)\*\*2),'fort.1'

si ottiene il grafico dell’ellissi e i punti in cui avvengono le collisioni. Utilizzando una rappresentazione a 32 bit si può osservare come in prima approssimazione i punti sembrino appartenere all’ellissi ma, zoomando in loro prossimità, si vede che essi non vi appartengono perfettamente; utilizzando una rappresentazione a 64 bit o superiore, la situazione migliora notevolmente e, anche zoomando al massimo, i punti appartengono sempre al grafico dell’ellissi.

Per quanto riguarda l’algoritmo, in alternativa alla soluzione classica delle equazioni di secondo grado, si potrebbe pensare di implementare il metodo di bisezione o il metodo di Newton, si tratta, infatti, di cercare gli zeri di una parabola di equazione: .

Il problema dell’implementazione di questi metodi è evidente: per quel che riguarda il metodo di bisezione, esso ha bisogno di un determinato intervallo ai cui estremi la funzione abbia segni discordi, la richiesta è quindi di due punti con caratteristiche particolari. Alle complicazioni si aggiunge il fatto che la soluzione che stiamo cercando deve essere necessariamente la maggiore delle due e che l’equazione della parabola vari ad ogni step del ciclo do. La soluzione potrebbe essere trovata calcolando (ad ogni step) la derivata prima della funzione e individuando il punto di minimo, esso costituirebbe l’estremo sinistro dell’intervallo che vogliamo considerare; per quanto riguarda l’estremo destro esso deve necessariamente trovarsi “a destra” sia dello zero della funzione che del punto di minimo, così da garantire che il segno della funzione in tale punto sia opposto al segno della funzione nel punto di minimo: solo così si può essere sicuri di trovare la maggiore delle soluzioni (ovviamente nel caso particolare della parabola). La questione del segno si può risolvere con l’utilizzo di istruzioni IF per distinguere i vari casi; per trovare l’effettiva ascissa del secondo punto si potrebbe implementare un ciclo do infinito che ,ad ogni step, aumenti di un’unità l’ascissa del punto di minimo e la cui condizione di uscita sia che la funzione abbia segno > o < di zero, a seconda dei casi, in tale punto. In alternativa il valore del secondo punto potrebbe essere immesso “per tentativi” da tastiera, anche se questo aumenterebbe di molto i tempi di esecuzione del programma e sarebbe impensabile nel caso si voglia considerare un elevato numero di collisioni.

Per quel che riguarda il metodo di Newton i problemi sono simili e similmente risolvibili, ma in questo caso l’algoritmo necessita di un solo punto di partenza che può essere individuato come un qualsiasi punto con ascissa maggiore del punto di minimo, così da assicurare la ricerca della maggiore delle soluzioni. Nel nostro caso basterà dunque fornire all’algoritmo del metodo di Newton il valore di minimo della funzione cui sommiamo 1.

Entrambi gli algoritmi sono piuttosto complicati da implementare rispetto alla prima soluzione che ho proposto, per semplicità ho implementato l’algoritmo di Newton e, utilizzando i computer dell’aula informatica al piano terra del Dipartimento di Fisica, i risultati corrispondevano esattamente con quelli del primo algoritmo che ho proposto, con qualsiasi tipo di rappresentazione.