XIII Scuola Estiva di Fisica - Corso di Pyton per la Fisica
Luigi Palatella

1.1 Introduzione generale alle simulazioni numeriche

In questo capitolo tratteremo il moto dei proiettili con e senza resistenza dell'aria. Porremo le basi intuitive su come trasformare le equazioni del moto in formule iterative gestibili da un algoritmo implementabile al computer.

Tutti gli studenti di fisica, con gioia o dolore, prima o poi affrontano questo problema classico. In realtà questo "classico" della didattica della fisica si sceglie per una motivazione molto semplice: l'equazione del moto è sicuramente la più semplice che non sia banalmente unidimensionale. In sostanza abbiamo un moto in due dimensioni in cui l'accelerazione è costante e diretta lungo l'asse verticale (in genere asse y) verso il basso. Scriviamo le equazioni del moto in maniera vettoriale

$$\vec{a} = -g\hat{y} \tag{1.1}$$

dove $\hat{y}$ è il versore verticale lungo la y. Possiamo scrivere l'equazione per componenti, come forse è più comune nei libri scolastici

$$\begin{cases} a_x = 0 \\ \\ a_y = -g \end{cases} \tag{1.2}$$

Ora siamo di fronte ad un passaggio fondamentale necessario per tradurre le equazioni differenziali (ovvero le equazioni del moto) in formule utilizzabili da un computer. Che cosa vuol dire accelerazione istantanea? Dalla matematica sappiamo che si tratta di una

definizione che utlizza il concetto di limite. In formule

$$a_y = \lim_{\Delta t \to 0} \frac{v_y(t + \Delta t) - v_y(t)}{\Delta t}. \tag{1.3}$$

Ovviamente la stessa relazione c'è fra posizione e velocità

$$v_y = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t}. \tag{1.4}$$

Per dare il via ai nostri esperimenti di calcolo numerico dobbiamo fare un passo indietro, dimenticare il concetto di limite e rendere Δt piccolo (e cosa voglia dire "piccolo" lo determineremo di volta in volta) ma non zero. A questo punto le due definizioni diventano automaticamente delle leggi d'evoluzione. Prendiamo la definizione di velocità, risolviamo per $y(t + \Delta t)$ ottenendo

$$y(t + \Delta t) = y(t) + v_y(t)\Delta t. \tag{1.5}$$

Notiamo come, una volta conosciuta la funzione $v_y(t)$, si possa calcolare il nuovo valore di y, $y(t + \Delta t)$, in funzione del vecchio $y(t)$. Il lettore attento però noterà subito un problema. Finché si effettua l'operazione di passaggio al limite la definizione funziona senza ambiguità, tuttavia se Δt non è nullo il problema diventa: in che istante devo calcolare v_y nell'Eq.(1.6)?

Un famoso teorema di analisi, il teorema di Lagrange, dice che esiste almeno un valore $\xi \in (t, t + \Delta t)$, tale che

$$y(t + \Delta t) = y(t) + v_y(\xi)\Delta t. \tag{1.6}$$

Ovviamente però questo non ci aiuta molto, se noi non sappiamo il valore di ξ. Si può dire che l'essenza del calcolo numerico sta proprio nel cercare il modo più veloce (in termini di operazioni) e furbo possibile per ottenere un valore quanto più possibile vicino a $v_y(\xi)$. Per ora partiamo dal livello base, utilizziamo il metodo di Eulero, e poniamo $\xi = t$. Vedremo che non è sicuramente questa la scelta più ottimale, però, almeno a mio avviso, è sicuramente la più didattica per cominciare a capire quello che vogliamo fare.

Ora, applicando lo stesso ragionamento all'equazione per la y e la v (entrambe le componenti) otteniamo

$$\begin{cases} v_x(t+\Delta t) = v_x(t) \\ v_y(t+\Delta t) = v_y(t) - g\Delta t \\ x(t+\Delta t) = x(t) + v_x(t)\Delta t \\ y(t+\Delta t) = y(t) + v_y(t)\Delta t \end{cases} \tag{1.7}$$

Siamo in sostanza pronti a scrivere il nostro primo programma di calcolo numerico che calcola la posizione del proiettile (in assenza di attrito dell'aria) conoscendo la posizione e la velocità iniziale.

1.2 Il primo caso - proiettile senza attrito

La struttura del programma è abbastanza semplice. Per poter poi riutilizzare un po' del codice che scriviamo conviene già da ora definire una funzione che calcola l'accelerazione partendo dai valori di x, y, vx, e vy. Sempre per poter riciclare in seguito definiamo sin da ora i vettori come tridimensionali.

E' ovvio che il caso del proiettile in esame in realtà potrebbe essere trattato in maniera più semplice essendo l'espressione per l'accelerazione del tutto banale. Tuttavia è meglio utilizzare un formalismo che permette di riciclare quanto più possibile intere porzioni di codice a discapito della semplicità dei codici.

Scriviamo quindi la funzione accelerazione:

```
#Definiamo l'accelerazione

def acc(r,v):
    # sia r che v sono degli array 3x1, ricordiamo
    # che in python il primo elemento di un vettore è
    # r[0], il secondo r[1] e il terzo r[2]

    a = [0] * 3 # inizializzo un vettore nulla 3x1
```

```
    a[1] = -9.81

    return a
```

Alcuni chiarimenti sulla sintassi del python sono ora necessari. Prima di tutto in python gli array iniziano dall'elemento 0-esimo e finiscono con l'elemento (n-1)-esimo. Quindi nel vettore posizione $\vec{r}$ $r[0]$ sarà la coordinata x, $r[1]$ la y mentre $r[2]$ la z. Il comando $a = [0] * 3$ è un trucco "pythonico" per dichiarare rapidamente un array tridimensionale inizializzando a zero i suoi elementi. Gli ultimi due comandi sono abbastanza ovvi, pongo uguale a $-g$ la componente y di $\vec{a}$ e restituisco il vettore accelerazione appena calcolato con il comando *return a*.

Il programma vero e proprio dovrà iniziare assegnando le condizioni iniziali al vettore posizione $\vec{r}$ e velocità $\vec{v}$. Dopo di che costruiamo il ciclo sul tempo, che è il nucleo principale di ogni programma di questo tipo.

```
r0 = [0]*3
v0 = [3,4,0]
# un esempio qualunque del vettore velocità inziale
t = 0 #tempo inziale, 0 per convenzione
dt = 0.01 #time step, "piccolo".. vedremo se ok
tfin = 10 # tempo finale della simulazione

# assegno le condizioni iniziali conservandone
# i valori nei vettori r0 e v0

r = r0
v = v0
while t < tfin:

    #calcolo il vettore accelerazione
    a = acc(r,v)

    r[0] = r[0] + v[0] * dt
    r[1] = r[1] + v[1] * dt
    r[2] = r[2] + v[2] * dt

    v[0] = v[0] + a[0] * dt
    v[1] = v[1] + a[1] * dt
```

```
    v[2] = v[2] + a[2] * dt

    t = t + dt #incremento t di un time-step dt

    print(t,r,v)

   # fine ciclo sul tempo
```

Come ultima istruzione del ciclo sul tempo viene chiesto al programma di scrivere il valore attuale del tempo, della posizione e della velocità.

Prima di analizzare l'output del programma è probabilmente necessario spiegare alcuni dettagli della programmazione. Questo paragrafo può essere saltato se il lettore è già pratico di un minimo di concetti di python o di qualunque altro linguaggio di programmazione.

Come prima cosa analizziamo l'istruzione *while*:

```
while t<tfin:
    .......
    t = t + dt
```

L'istruzione *while* vuol dire essenzialmente finché + condizione. Ovvero fino a che la condizione subito dopo l'istruzione *while* è soddisfatta il programma ripete il ciclo di istruzione che è spostato di un tab a destra rispetto al comando *while*. Python utilizza l'indentazione per decidere il flusso delle istruzioni, quindi indentazioni (con il tab) e rientri non vanno assolutamente messi casualmente!

Pur essendo i codici tutti scaricabili al sito *http://sitodellibro* consiglio vivamente di copiare i codici manualmente. Sebbene questa affermazioni sembri quasi una bestemmia in tempi di copia, incolla, condividi ecc. ritengo che copiare i codici in python sia utile, soprattutto per impratichirsi nell'indentazione, nella sintassi e nella distinzione fra maiuscole e minuscole, fra , e ., fra 0 e O.

Ritornando al nostro codice, dopo tutte le varie istruzioni osserviamo la riga

$$t = t + dt.$$

Questa istruzione, se intepretata in senso letterale matematico, sarebbe un'equazione che implicherebbe che $dt = 0$. Va qui però ricordato un principio fondamentale della programmazione, ovvero la differenza fra il simbolo $=$ di assegnazione (come in questo caso) e quello di comparazione in una condizione che in python è $==$. Quando noi scriviamo $=$ non ci stiamo chiedendo se le due grandezze ai lati del simbolo siano uguali, ma *stiamo assegnando alla variabile di sinistra il valore che assume l'espressione a destra del risultato.* Quindi se, per esempio, $t = 2.34$ e $dt = 0.01$ il programma esegue questa istruzione

$$t \leftarrow 2.34 + 0.01$$

dove $\leftarrow$ sta per *assegna.* Quindi il computer calcola prima l'espressione a destra dell'uguale (che in questo caso dà come risultato 2.35) e la assegna alla variabile t che quindi assume il nuovo valore, "dimenticando" il vecchio valore assunto (che infatti viene perso dalla memoria).

Quindi in sostanza il programma incrementa la variabile tempo t (pensatela come una scatolina nella memoria dove è salvato un numero con la virgola) di dt e ogni volta controlla che sia arrivato al tempo finale $tfin$, quando questo succede esce dal ciclo e, in questo caso, termina l'esecuzione del programma.

Dopo questa digressione sul significato del simbolo $=$ diventerà più chiaro anche il significato delle istruzioni

$$r[0] = r[0] + v[0] * dt$$

In effetti anche in questo caso il calcolatore calcola $v[0] * dt$, lo somma al vecchio valore di $r[0]$ e mette il valore calcolato nella stessa posizione di memoria relativa alla variabile $r[0]$ cancellando il vecchio valore.

Credo che ormai al lettore più attento non sfuggirà il significato delle istruzioni del tipo

$$v[0] = v[0] + a[0] * dt$$

Vediamo ora l'ultima istruzione del ciclo *while*, l'istruzione *print* che scrive in ogni riga il tempo t, la posizione r e la velocità v.

Rispetto a molti casi, che tratteremo nel seguito del libro, il moto del proiettile in assenza di resistenza dell'aria possiede una soluzione analitica. Per i meno addentro al calcolo numerico conviene dare una definizione precisa di questo termine. Le equazioni del moto in fisica, ovvero le formule che legano l'accelerazione alla posizione e alla velocità del sistema, sono in generale molto difficili se non impossibili da risolvere (si dice spesso integrare) analiticamente, ovvero procedendo con dei calcoli simbolici esatti. Molto spesso bisogna appunto ricorrere, anche in ambito accademico o ingegneristico, a soluzioni numeriche, anche se effettuate con algoritmi sicuramente più raffinati di quelli che presenteremo in questo corso didattico.

Ora in questo caso la soluzione analitica esiste, gli studenti dei primi anni di Liceo Scientifico la imparano a memoria, quelli di quinto che conoscono i rudimenti dell'analisi la possono anche ricavare o verificare che essa soddisfa l'equazione del moto (1.1). Scriviamola ora per promemoria

$$\begin{cases} x = v_{0x}t \\ y = v_{0y}t - \dfrac{1}{2}gt^2 \\ v_x = v_{0x} \\ v_y = v_{0y} - gt \end{cases}$$

Il grande vantaggio per noi di avere una soluzione analitica è che possiamo confrontare la soluzione *esatta* con quella numerica per vedere se stiamo procedendo correttamente.

Come si nota in fig.1.1, l'accordo almeno qualitativo è ottimo. Lasciamo al lettore il compito di scorrere i valori numerici forniti dal pro-

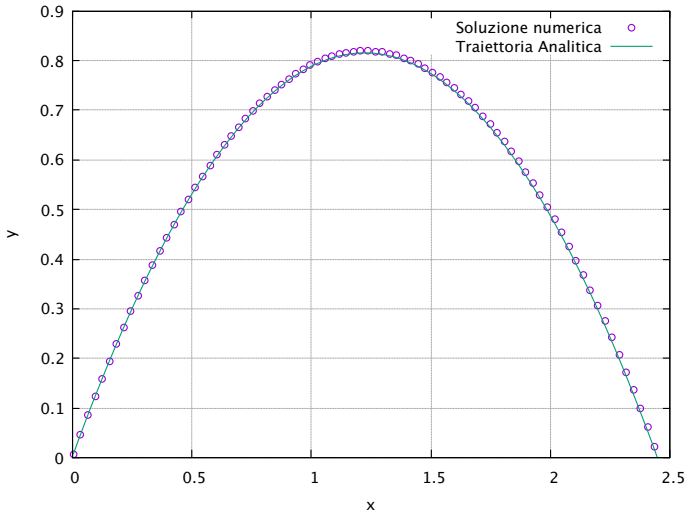

Figura 1.1: Confronto fra soluzione numerica e analitica per la traiettoria di un proiettile lanciato da quota zero con $v_{0x} = 3$ m/s e $v_{0y} = 4$ m/s.

gramma per confrontare altezza massima, gittata e altre proprietà note analiticamente con i valori ottenibili dal calcolo numerico.

In questo caso abbiamo una soluzione analitica con cui confrontare il risultato, ma nella realtà questo accade ovviamente molto raramente. Come si fa a capire se la nostra simulazione numerica funziona bene? In questo caso l'unico parametro numericamente importante è il *time-step* dt. Un modo molto semplice e pratico per verificare i nostri risultati è modificare dt e osservare se i risultati sono indipendenti dalla scelta fatta. Ovviamente ci saranno sempre delle fluttuazioni ma se queste saranno minori di una certa tolleranza possiamo ragionevolmente affermare che la nostra simulazione è corretta.

In tabella riportiamo i risultati ottenuti (controllateli per esercizio per impratichirvi con l'uso del codice python):

Come si nota il valore diventa abbastanza stabile in funzione di dt, il problema è però che differisce abbastanza dal valore analitico che in questo caso sarebbe $y(0.3) = 0.75855$ m. Questo non è dovuto

dt	$y(0.3)$
0.001	0.7600215
0.002	0.7614930
0.005	0.7659075
0.01	0.77326500
0.02	0.78798000

Tabella 1.1: Valore di $y(0.3)$, ovvero la coordinata y calcolata al tempo $t = 0.3$ s al variare del *time-step* dt.

ad un nostro errore ma ad un problema ben noto di questo schema di soluzione, del metodo di Eulero.

Come ripeteremo spesso, questo libro *non* è un testo sul calcolo numerico, e non proporremo sempre gli schemi risolutivi più corretti, ma cercheremo sempre di perseguire un approccio più semplice e didattico possibile. Avendo ottenuto un errore tutto sommato ragionevole (intorno al 2%) per ora ci riteniamo soddisfatti e possiamo andare avanti nell'aggiungere ingredienti al nostro sistema fisico.

In appendice **??** spiegheremo come correggere questo ed altri codici per ottenere una maggiore precisione numerica.

1.3 Proiettile con attrito

Nella realtà il proiettile non è solamente soggetto alla forza di gravità ma anche alla resistenza dell'aria. L'effetto dell'aria su un corpo che viaggia ad una velocità data è alquanto complesso. In generale l'effetto dell'aria ha una componente perpendicolare alla velocità (detta portanza in genere) ed una antiparallela alla velocità detta attrito.

Se uno vuole simulare la dinamica di un aereo con le ali ovviamente dovrà considerare l'effetto di portanza, ma un proiettile si può considerare in prima approssimazione come soggetto solamente alla forza di attrito. Per velocità molto piccole il flusso dell'aria è laminare, ovvero stazionario nel tempo, ma nella realtà un proiettile (ma anche una pallina da tennis o un pallone da calcio) viaggiano a velocità tali da avere moto turbolento.

In questo caso la formula per la forza di attrito F_d è data da:

$$F_d = \frac{1}{2}\rho c_d A|\vec{v}|^2, \tag{1.8}$$

dove ρ è la densità dell'aria (possiamo assumere $\rho = 1.3\ kg/m^3$), A è l'area frontale del proiettile, mentre c_d è il coefficiente di attrito che dipende dalla forma dell'oggetto ed è tanto più piccolo quanto maggiore è la cosiddetta aereodinamicità del proiettile.

Non è facile stimare c_d teoricamente, ma può essere ricavato da misure di laboratorio o (mando a dirlo) da simulazioni numeriche ben più complesse di quelle che stiamo mostrando in questo testo. Possiamo assumere per un proiettile di artiglieria di diametro 155 mm un $c_d = 0.12$.

Siamo ora di fronte al tipico problema che affronta chi voglia scrivere un programma di simulazione partendo dalle formule dei libri di testo. In realtà passare dalla formula (1.8) al codice in python non dovrebbe essere difficile se si ha un buona padronanza con i vettori. Questo aspetto però viene spesso trascurato nella didattica del Liceo Scientifico provocando un blocco pressoché totale quando uno studente tipico si trova in questa situazione.

La prima domanda da porsi è: cosa è F_d nell'equazione (1.8)? La risposta corretta è il modulo della forza di attrito. La convenzione di utilizzare la lettera senza il simbolo di vettore come sinonimo del modulo è sicuramente utile ma spesso fuorviante, come vedremo

nel capitolo sul moto dei pianeti. Avendo accertato che $F_d = |\vec{F}_d|$, siamo però fermi di fronte alla domanda principale: come ottengo le *componenti* di $\vec{F}_d$? Infatti solo con le componenti potrò aggiungere i termini corretti alle equazioni del moto scritte precedentemente.

Per mia esperienza didattica, davanti a questa domanda la maggior parte degli studenti iniziano a calcolare l'angolo fra il vettore e l'asse x, sperando poi di poter calcolare le componenti. Questo approccio ha due problemi principali: il primo è che in tre dimensioni non è affatto facile ragionare con gli angoli, la seconda è che le funzioni goniometriche non hanno funzione inversa. Ne hanno una ristretta, ma non effettivamente una funziona inversa. Prima di proporre la via più semplice e rigorosa con un esempio spiego l'importanza di quello che ho appena affermato.

Tutti gli studenti si saranno trovati davanti alla domanda: dato il vettore $\vec{A} = (A_x, A_y, 0)$, qual è l'angolo Θ fra l'asse x e il vettore $\vec{A}$? Gli studenti più brillandi risponderanno subito

$$\Theta = \mathrm{atan}\left(\frac{A_y}{A_x}\right).$$

Ora, questa risposta in realtà non è corretta. Se A_x è negativa il risultato non può essere giusto, poichè la funzione $\mathrm{atan}(x)$ dà come risultato un angolo compreso in $[-\pi/2, \pi/2]$ mentre se $A_x < 0$ ovviamente l'angolo appartiente al terzo o quarto quadrante.

In effetti la formula giusta dovrebbe essere

$$\Theta = \begin{cases} \mathrm{atan}\left(\dfrac{A_y}{A_x}\right) & \text{se } A_x > 0 \\ \mathrm{atan}\left(\dfrac{A_y}{A_x}\right) + \pi & \text{se } A_x < 0 \\ \pi/2 & \text{se } A_x = 0, A_y > 0 \\ -\pi/2 & \text{se } A_x = 0, A_y < 0 \end{cases}$$

Ovviamente se sia $A_x = 0$ e $A_y = 0$ l'angolo non è definibile (tutte le condizioni appena descritte sono già implementate in *python* nella funzione *math.atan2(y, x)*).

Anche se in questo modo tutto sommato, almeno nel caso bidimensionale, il problema potrebbe essere affrontato, spiegherò ora come risolvere il caso generale usando uno dei concetti più bistrattati del calcolo vettoriale al Liceo, il versore. Molti studenti imparano la definizion di versore chiedendosi intimamente a cosa serva. Effettivamente non è intuitivo capire come utilizzare questo concetto senza aver visto un esempio. Pensiamo intanto a cosa sia il versore di un particolare vettore. Prendiamo il vettore velocità $\vec{v}$ e chiediamoci come sia fatto un vettore che ha la stessa direzione, lo stesso verso, ma modulo unitario. Ovviamente questo nuovo vettore sarà un versore, sarà in particolare *il versore* del vettore $\vec{v}$. Indiciamolo con $\hat{v}$. Ma come si può calcolare questo versore? Basta usare la formula

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|}$$

ovvero il vettore $\vec{v}$ diviso per il suo modulo in modo che dopo la divisione il modulo sarà ovviamente unitario. Annotiamo subito un'altra formula ovvia che però sarà utile in futuro

$$\vec{v} = |\vec{v}|\hat{v}$$

ovvero, se noi di un vettore sappiamo modulo e versore, basta moltiplicare il modulo (che è uno scalare) per il versore (che è un vettore) per ottenere $\vec{v}$. Torniamo ora alla forza di attrito. Il modulo è dato dalla (1.8), ma il suo versore? La risposta non è molto difficile, la forza di attrito avrà la stessa direzione della velocità ma verso opposto. Quindi per ottenere il vettore forza di attrito $\vec{F_d}$ dovremo moltiplicare il modulo (che è sempre positivo!) per l'opposto del versore $\hat{v}$. Avremo quindi

$$\vec{F_d} = -\frac{1}{2}c_d\rho A|\vec{v}|^2\hat{v} \tag{1.9}$$

ma ricordando che $|\vec{v}|\hat{v} = \vec{v}$ otteniamo

$$\vec{F_d} = -\frac{1}{2}c_d\rho A|\vec{v}|\vec{v}. \tag{1.10}$$

Siamo ad un passo dalla soluzione, dobbiamo solo ricordare come si moltiplica un vettore per uno scalare [1], ovvero

$$a \cdot \vec{v} = a \cdot (v_x, x_y, v_z) = (av_x, av_y, av_z)$$

con a uno scalare. Applicando questa formula alla (1.10) otteniamo

$$\vec{F_d} = \left(-\frac{1}{2} c_d \rho A |\vec{v}| v_x, -\frac{1}{2} c_d \rho A |\vec{v}| v_y, -\frac{1}{2} c_d \rho A |\vec{v}| v_z, \right) \tag{1.11}$$

dove ovviamente vale

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

Queste formule sono generali, proviamo ora a calcolare i parametri per un proiettile reale. Supponiamo che la forma sia essenzialmente cilinrica con diametro di 155 mm. Quindi

$$A = \pi r^2 = \pi \, 0.155^2 = 0.075 \, m^2$$

quindi si ha che

$$\frac{1}{2} \rho c_d A = 0.00589 \, \mathrm{kg/m}$$

per calcolare l'accelerazione dobbiamo anche dividere per la massa del proiettile, che sul web viene riportato come $m = 43.5 \; kg$. Siamo dunque pronti per scrivere la formula per l'accelerazione complessiva

$$\begin{aligned} \vec{a} &= \left(-\frac{1}{2} c_d \rho A/m |\vec{v}| v_x, -\frac{1}{2} c_d \rho A/m |\vec{v}| v_y - g, -\frac{1}{2} c_d \rho A/m |\vec{v}| v_z, \right) \\ &= (-k|\vec{v}|v_x, -k|\vec{v}|v_y - g, -k|\vec{v}|v_z,) \end{aligned} \tag{1.12}$$

con $k = 1.35 \cdot 10^{-3} \, m^{-1}$. Siamo ora pronti a modificare la funzione che calcola l'accelerazione nel nostro programma *python*

```
#Definiamo l'accelerazione
def acc(r,v):
    # sia r che v sono degli array 3x1, ricordiamo
    # che in python il primo elemento di un vettore è
```

[1] operazione da non confondere col prodotto scalare fra due vettori

```
    # r[0], il secondo r[1] e il terzo r[2]

    k = 1.35e-4

    vmod = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2])

    a = [0] * 3 # inizializzo un vettore nulla 3x1

    a[0] = -k * vmod * v[0]
    a[1] = -k * vmod * v[1] - 9.81
    a[2] = -k * vmod * v[2]

    return a
```

per utilizzare la funzione radice quadrata *sqrt* dobbiamo importarla dal module math inserendo all'inizio del codice l'istruzione

```
# bisogna importare la funzione radice quadrata
# dal module  math
from math import sqrt as sqrt
```

Siamo ora pronti a fare qualche "esperimento". In fig.2.1 mostriamo il confronto fra due traiettorie con e senza attrito con l'aria per un proiettile lanciato con velocità iniziale $V_0 = 200$ m/s e angolo $\theta = 45°$

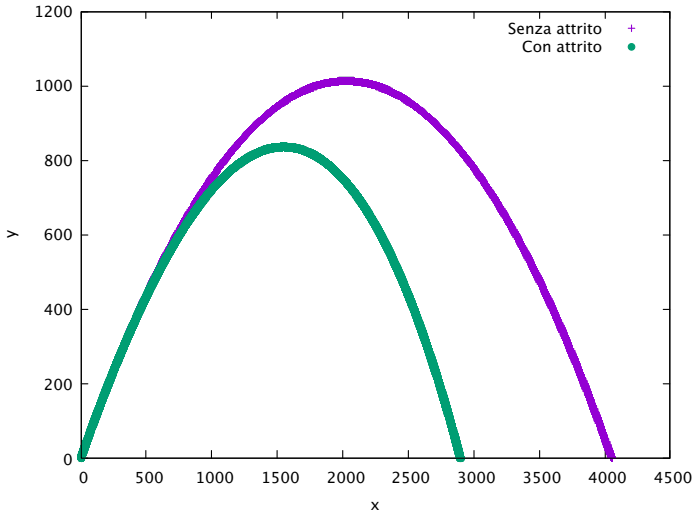

Figura 1.2: Confronto fra la traiettoria del proiettile lanciato con $v_{0x} = v_{0y} = 141$ m/s con e senza attrito con l'aria.

Come si vede l'effetto della resistenza dell'aria è duplice: la gittata ovviamente diminuisce ed anche la forma della traiettoria che non è più esattamente simmetrica rispetto al punto più in alto. Questo avviene poiché la resistenza dell'aria ha un effetto maggiore nella prima fase quando la velocità del proiettile è più grande.

Possibili tanti altri esperimenti.....