Assignment 1: Meccanica Orbitale

Matteo Raffaele Vacca

22 Maggio 2023

Abstract

Nel documento corrente vengono riportati differenti programmi realizzati in python in cui ci si propone di definire:
a) Il moto stabile di un pianeta attorno ad un sistema binario di stelle
b) Il moto di due corpi celesti che si influenzano a vicenda, in moto attorno ad una stessa stella, tale per cui uno destabilizzi l'orbita dell'altro e un pianeta leggero sia intrappolato da uno massivo e vi orbiti intorno.

1 Introduzione

Per comprendere quali siano state le procedure seguite al fine di raggiungere gli scopi preposti bisogna effettuare una introduzione in merito alla costruzione del problema:

- ▷ ***Equazioni Differenziali Ordinarie***: anche dette ODE, come si comprende dal nome sono equazioni che coinvolgono una funzione e le sue derivate, generalmente esse descrivono sistemi dinamici come quello che ci si propone di computare

- ▷ ***Velocity Verlet***: trattasi di un algoritmo ideato per risolvere equazioni differenziali ordinarie numericamente.
  Per poter funzionare il Verlet (abbreviato per praticità) necessità di operare con intervalli finiti di tempo a partire da condizioni iniziali definite

$$t = t_0 \tag{1}$$

  che evolverà all'istante successivo in

$$t = \tau + t_0 \tag{2}$$

  Nello specifico il Verlet, facendo uso delle equazioni del moto, calcola ad ogni istante la posizione x e la velocità v utilizzando un *time-step* che non è altri se non il t qui sopra citato. Bisogna fare attenzione perchè ciò non implica necessariamente che

$$\tau = t_0 \tag{3}$$

  anche se questa è una possibilità. Difatti per costruire il primo programma in origine si è utilizzata questa convenzione, che però è stata poi sostituita con il *time-step*

adattivo ricavato dalle equazioni del moto (analizzato nel dettaglio in *'Esecuzione'*). L'algoritmo è così strutturato:

$$\mathbf{r}_{i+1} = \mathbf{r}_i + \mathbf{v}_i\tau + \frac{1}{2}\mathbf{a}_i\tau^2 \tag{4}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \frac{1}{2}(\mathbf{a}_i + \mathbf{a}_{i+1})\tau \tag{5}$$

dove il tempo totale di simulazione dipende dalla scelta che viene fatta nel criterio della selezione del *time-step*. In caso di time step fisso, detto N il numero di passi temporali, il periodo, ovvero il tempo di simulazione, sarà

$$T = Nt_0 \tag{6}$$

mentre nel caso di *time-step adattivo*

$$T = \sum_{n=1}^{i+1} \tau_i \tag{7}$$

Vi è sempre il problema che un *time-step* troppo elevato generi instabilità nel sistema ma al contempo si possono effettuare simulazioni relativamente lunghe. L'erroe generato utilizzando il Verlet sarà nell'ordine di $\sigma(\tau^4)$ che lo rende più preciso di altri algoritmi.
Ulteriore vantaggio nell'utilizzo del Verlet è che esso è un algoritmo simplettico, che quindi conserva l'energia, criterio fondamentale al fine di poter determinare se un'orbita sia stabile o meno.

▷ ***Meccanica Orbitale*** : Dalle equazioni della Gravitazione Universale per interazione tra due corpi si ricava facilmente

$$F_1 = G\frac{m_1 m_p}{r_1^2} \qquad\qquad F_2 = G\frac{m_2 m_p}{r_2^2} \tag{8}$$

dove con m_1 ed m_2 sono identificate le masse delle due stelle mentre con m_p quella del pianeta. La forza risultante sul pianeta sarà la somma delle due.
Al fine di eseguire il programma bisognava comprendere per quali condizioni un'orbita fosse stabile, i modi per poterlo comprendere erano:
a) *Random*: posto N e tempo inziale si poteva eseguire il programma variando la distanza finchè non si trovava stabilità
b) *Condizioni M.O.* : Dalle leggi di meccanica orbitale si possono impostare condizioni di velocità o distanza tale per cui l'orbita sia stabile
Si è optato per seguire la seconda possibilità e le leggi che descrivono i fenomeni sono rispettivamente per *Orbita circolare*

$$v_p = \sqrt{\frac{GM}{r}} \tag{9}$$

e per *Orbita ellittica*

$$v_p = \sqrt{GM\left(\frac{2}{r} - \frac{1}{a}\right)} \tag{10}$$

con a semiasse maggiore dell'ellisse.
Questo metodo ha un difetto: le condizioni *o.c.* (orbita circolare) pongono sin dal principio il corpo a velocità ideale per qualunque distanza affinchè l'orbita sia circolare, le condizioni *o.e.* impongono intrinsecamente i semiasse dell'ellisse che si sta costruendo. Come si giustificano queste scelte? E' presto detto: se l'algoritmo è ben costruito l'orbita sarà stabile, circolare o ellittica, a prescindere dalla distanza a cui viene messo il pianeta.
Lo stratagemma per identificare la stabilità dell'orbità, è verificare il bilancio energetico, in altri termini: se la somma dell'energia potenziale e cinetica del pianeta è negativa nel tempo l'orbita è considerata stabile a prescindere dalla sua forma. Quanto detto coincide col dire che il volume dello spazio delle fasi si deve conservare. Dunque dalla meccanica Hamiltoniana, l'integrale primo del moto sarà

$$H = T + V = \frac{1}{2}mv_p^2 - GMm_p\left(\frac{1}{r_1} + \frac{1}{r_2}\right) \tag{11}$$

dove si sono poste le stelle a massa pari tra loro. Non è dunque necessario che l'energia totale del pianeta sia costante nel tempo, la sola negatività dell'energia totale assicura che esso si trova in uno stato legato.

2 Esecuzione

Al fine di utilizzare il Verlet bisogna ricavarsi le accelerazioni a_i e per fare ciò si costruisce un sistema di unità di misura basato sulle grandezze astronomiche tale per cui, poste le masse delle stelle $m_1 = m_2 = 1 = M$, posta la distanza del pianeta rispetto all'origine x_0 , posta la distanza in modulo di ciascuna stella dal centro delle coordinate $d/2$, si definisce la distanza del pianeta dalle stelle e si usa questo parametro per ricavare l'accelerazione su ogni componente per ciascuna stella all'istante iniziale su un piano bidimensionale cartesiano come segue

$$F_{x1} = GM\frac{x_0 - d/2}{r_1^2} \qquad\qquad F_{x2} = GM\frac{x_0 + d/2}{r_2^2} \tag{12}$$

mentre per le componenti y

$$F_{y1} = GM\frac{y_0}{r_1^2} \qquad\qquad F_{y2} = GM\frac{y_0}{r_2^2} \tag{13}$$

Dove ci si sarà resi conto dell'assenza della massa del pianeta, in realtà in tale approssimazione, come detto, si stanno calcolando le accelerazioni e non le forze, pertanto $a = F/m$ ma si è lasciata la dicitura F per preservare la sintassi usata nel programma.
Per la realizzazione del Verlet si pone la costante di gravitazione universale pari a

$$G = 4\pi^2 AU^3 yr^{-2} M^{-1} \tag{14}$$

in modo da evitare scale astronomiche nella computazione.
Di seguito vengono riportati i notebook di ciascun programma:

Problema A:

▷ ***Orbita Circolare*** :

```
#velocity-Verlet sistema binario di stelle t step adattivo
import numpy as np
from numpy import cos, pi, sin
from math import sqrt, asin, atan2
import matplotlib.pyplot as plt

# imposto le grandezze iniziali
d = 1 #Distanza tra Stelle
m = 3E-6  #massa Terra
GM = 4*pi**2   #cost. Sole
N = 10000
t_0 = 0.01
A = 0.001 #coefficiente da introdurre nel t step


x_0 = 5 # x_0 posizione iniziale pianete rispetto all'origine
x_s_0 = x_0 - d/2 #posizione rispetto stella vicina
x_a_0 = x_0 + d/2 #posizione stella lontana

#Definisco gli elementi array di posto zero
y_0 = 0.0

v_y_0 = sqrt(2*(GM)/x_0) #velocit iniziale per orbita circolare
v_x_0 = 0

r_s = sqrt((x_s_0)**2 + (y_0)**2) #distanza stella lontana
r_a = sqrt((x_a_0)**2 + (y_0)**2) #distanza stella vicina

F_x_s = -GM*x_0/(r_s**3) #forza stella lontana
F_y_s = -GM*y_0/(r_s**3)

F_x_a = -GM*x_0/(r_a**3) #forza stella vicina
F_y_a = -GM*y_0/(r_a**3)

F_x_t = F_x_s + F_x_a #forza totale su pianeta
F_y_t = F_y_s + F_y_a

#Preparo i vettori in cui andr a salvare i risultati dei cicli e li
   inizializzo
X = np.zeros(N+1) #posizione rispetto al centro
X[0] = x_0
Y = np.zeros(N+1)
Y[0] = y_0

F_x_1 = np.zeros(N+1) #forza stella 1
```

```
F_x_1[0] = F_x_s
F_y_1 = np.zeros(N+1)
F_y_1[0] = F_y_s

F_x_2 = np.zeros(N+1) #forza stella 2
F_x_2[0] = F_x_a
F_y_2 = np.zeros(N+1)
F_y_2[0] = F_y_a

F_x = np.zeros(N+1) #forza totale
F_x[0] = F_x_t
F_y = np.zeros(N+1)
F_y[0] = F_y_t
F = np.zeros(N+1)
F[0] = (F_x[0]**2 + F_y[0]**2)**0.5

v_x = np.zeros(N+1)
v_x[0] = v_x_0
v_y = np.zeros(N+1)
v_y[0] = v_y_0
v = np.zeros(N+1) # modulo velocit
v[0] = (v_x[0]**2 + v_y[0]**2)**0.5

r_1 = np.zeros(N+1) #distanza stella 1
r_1[0] = r_s

r_2 = np.zeros(N+1) #distanza stella 2
r_2[0] = r_a

r =np. zeros(N+1) #distanza dal centro
r[0] = x_0

E_pot = np.zeros(N+1) #potenziale su pianeta
E_pot[0] = - (m*GM*(1/r_1[0] + 1/ r_2[0]))

E_k = np.zeros(N+1) #cinetica
E_k[0] =  0.5*m*(v[0]**2)

E_tot =np.zeros(N+1) #totale
E_tot[0] = E_pot[0] + E_k[0]

t_1 = np.zeros(N+1) #time step per stella 1
t_1[0] = t_0

t_2 = np.zeros(N+1) #time step per stella 2
t_2[0] = t_0

t = np.zeros(N+1) #time step totale
```

```
t[0] = t_0

T_t = t_0 #tempo totale
```

Si noti che si è usata una forma semplificata per le velocità perchè ovviamente i termini $d/2$ presenti nelle definizioni delle posizioni si annullavano vicendevolmente.

Come si può notare sono stati realizzati degli array per ogni grandezza che viene aggiornata nel ciclo, in questo modo ciascun nuovo elemento viene salvato in memoria evitando problemi di riscrittura. In seguito all'inizializzazione si può scrivere il Verlet con *time step adattivo* così definito:

$$\tau_{i+1} = \frac{-\mathbf{v}_{i+1} + \sqrt{\mathbf{v}_{i+1}^2 - 2A\mathbf{F}_{i+1}|\mathbf{r}_{i+1}|}}{\mathbf{F}_{i+1}} \tag{15}$$

in questo modo, grazie all'introduzione del coefficiente A, la somma al numeratore non va a zero, e grazie al valore assoluto della distanza il termine sotto radice è non negativo. Questo metodo discerne dalla soluzione non immaginaria dell'equazione di secondo grado del moto uniformemente accelerato, che altri non è se non l'equazione 4. Il *time step adattivo* consente di avere tempi di calcolo più brevi grazie alla concatenazione delle equazioni, per costruzione ci si attende un alto numero di punti lungo le zone con curve più accentuate.

```
#eseguo a tutti gli effetti il velocity-Verlet
for n in range(N):
  X[n+1] = X[n] + t[n]*v_x[n] + 0.5*F_x[n]*t[n]**2
  Y[n+1] = Y[n] + t[n]*v_y[n] + 0.5*F_y[n]*t[n]**2
  r[n+1] = sqrt( X[n+1]**2 + Y[n+1]**2)

  r_1[n+1] = sqrt( (X[n+1]-d/2)**2 + Y[n+1]**2) #nuova distanza stella 1
  r_2[n+1] = sqrt( (X[n+1]+d/2)**2 + Y[n+1]**2) #nuova distanza stella 2

  F_x_1[n+1] = -(GM)*(X[n+1]- d/2)/r_1[n+1]**3 #Forza stella 1 su pianeta
  F_y_1[n+1] = -(GM)*(Y[n+1])/r_1[n+1]**3

  F_x_2[n+1] = -(GM)*(X[n+1]+d/2)/r_2[n+1]**3 #Forza stella 2 su pianeta
  F_y_2[n+1] = -(GM)*(Y[n+1])/r_2[n+1]**3

  F_x[n+1] = F_x_1[n+1] + F_x_2[n+1] #Forza totale su pianeta
  F_y[n+1] = F_y_1[n+1] + F_y_2[n+1]
  F[n+1] = (F_x[n+1]**2 + F_y[n+1]**2)**0.5

  v_x[n+1] = v_x[n] + 0.5*t[n]*(F_x[n]+F_x[n+1]) #Aggiornamento velocit
  v_y[n+1] = v_y[n] + 0.5*t[n]*(F_y[n]+F_y[n+1])
  v[n+1] = (v_x[n+1]**2 + v_y[n+1]**2)**0.5

  t_1[n+1] = (-v[n+1] + ((v[n+1]**2 -
   2*A*F[n+1]*abs(r_1[n+1])))**(1/2))/F[n+1] #t step rispetto a stella 1
  t_2[n+1] = (-v[n+1] + ((v[n+1]**2 -
   2*A*F[n+1]*abs(r_2[n+1])))**(1/2))/F[n+1] #t step rispetto a stella 2
  t[n+1] = abs(min( t_1[n+1] , t_2[n+1]))
```

```
  E_pot[n+1] = - (m*GM*(1/r_1[n+1] +1/r_2[n+1])) #Energia potenziale
  E_k[n+1] =  0.5*m*(v[n+1]**2) #Energia cinetica
  E_tot[n+1] = E_pot[n+1] + E_k[n+1] #Energia totale

  T_t += t[n+1]
  T = np.linspace(0 , T_t, N+1)
```

A questo punto si introducono i grafici

```
#costruisco i grafici
plt.rcParams['font.family'] = 'serif'

f1 = plt.title('Orbite planetarie' , fontsize = 14)

plt.scatter(X , Y , s = 0.1 ,color ="green", label = "orbita")
plt.scatter(d/2 , y_0, color="red", label = "star 1")
plt.scatter(-d/2 , y_0, color="orange", label = "star 2")
plt.xlabel('X (UA)' , fontsize = 12)
plt.ylabel('Y (UA)' , fontsize = 12)
plt.axis('equal')
plt.legend(fontsize = 10)
plt.show()

f2 = plt.title('Conservazione Energia' , fontsize = 14)
plt.plot(T , E_pot, label = 'E Cinetica')
plt.plot(T , E_k, label = 'E Potenziale')
plt.plot(T , E_tot, label = 'E Totale')
plt.xlabel('Tempo' , fontsize = 12)
plt.ylabel('Energia' , fontsize = 12)
plt.legend(fontsize = 10)
plt.show()

f3 = plt.title('Variazione Temporale' , fontsize = 14)
plt.plot(T , t )
plt.xlabel('Periodo', fontsize = 12)
plt.ylabel('t step' , fontsize = 12)
plt.show()
```

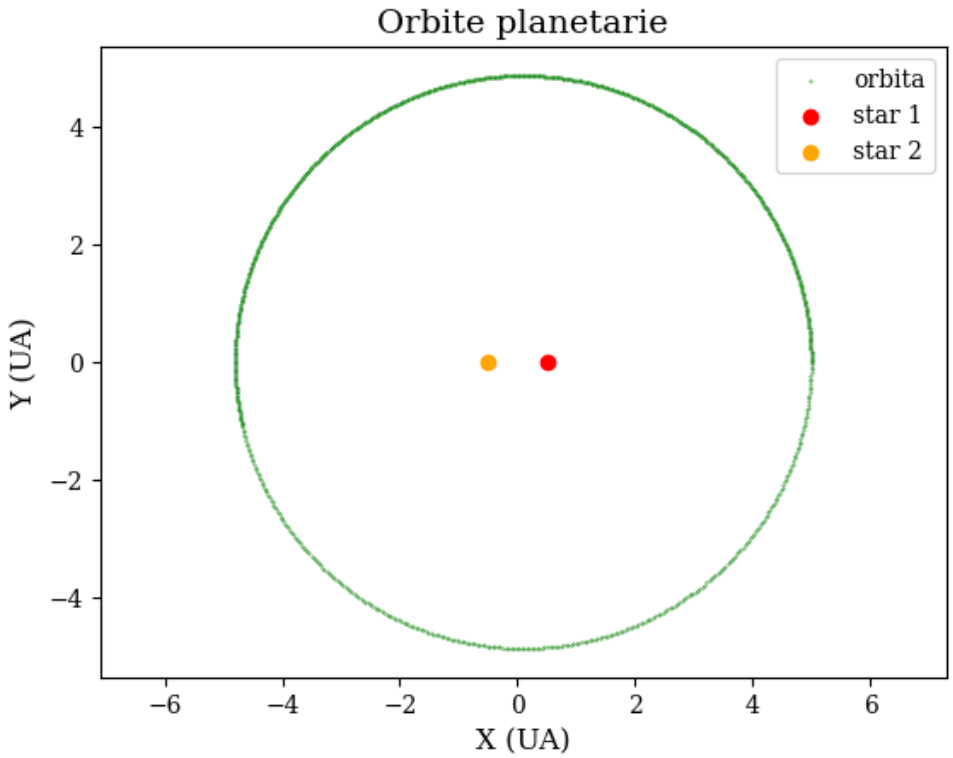

Figure 1: Orbita circolare sistema binario di stelle, pianeta singolo

Come si nota dal notebook python e dalla Figura 1, è stato eseguito uno *scatter plot* per vedere la concentrazione di punti che si aspettano dal *time step adattivo*, tuttavia si presenta una maggiore densità di punti per la mezza orbita superiore. Partendo da un *time step* iniziale maggiore o minore, non si riscontra una differenza nella distribuzione di punti. O meglio, se N è molto grande e τ è molto piccolo, si ottengono orbite molto definite con grandi oscillazioni dell'energia e un grafico molto fitto. Al contrario, con N piccolo e τ grande, il programma non completa il grafico. Il programma sta eseguendo circa un'orbita e mezzo, per evitare questo fatto e stimare la distribuzione dei punti si interrompe il ciclo prossimo alla chiusura dell'orbita con una condizione *if*

```
if T_t >= 7.7:
    break
```

Da cui si ricava il grafico (Figura 2). Come è palese, non è distinguibile una densità differente di punti lungo l'orbita. Questo fatto è attribuibile all'impostazione della velocità iniziale, per orbita circolare, anche con *time step adattivo*, non vi saranno variazioni significative di velocità, ergo per definizione non vi saranno grandi variazioni di τ.

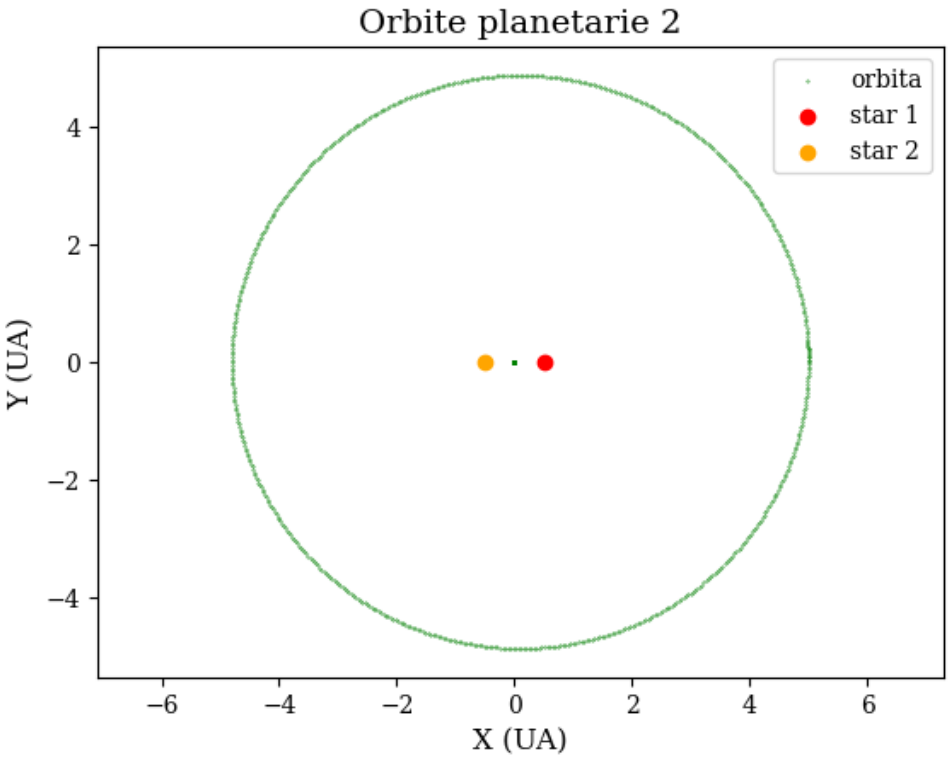

Figure 2: Orbita circolare sistema binario di stelle, pianeta singolo, periodo ridotto

Si introducono i grafici relativi a energia nel tempo e a *time-step adattivo* nel tempo

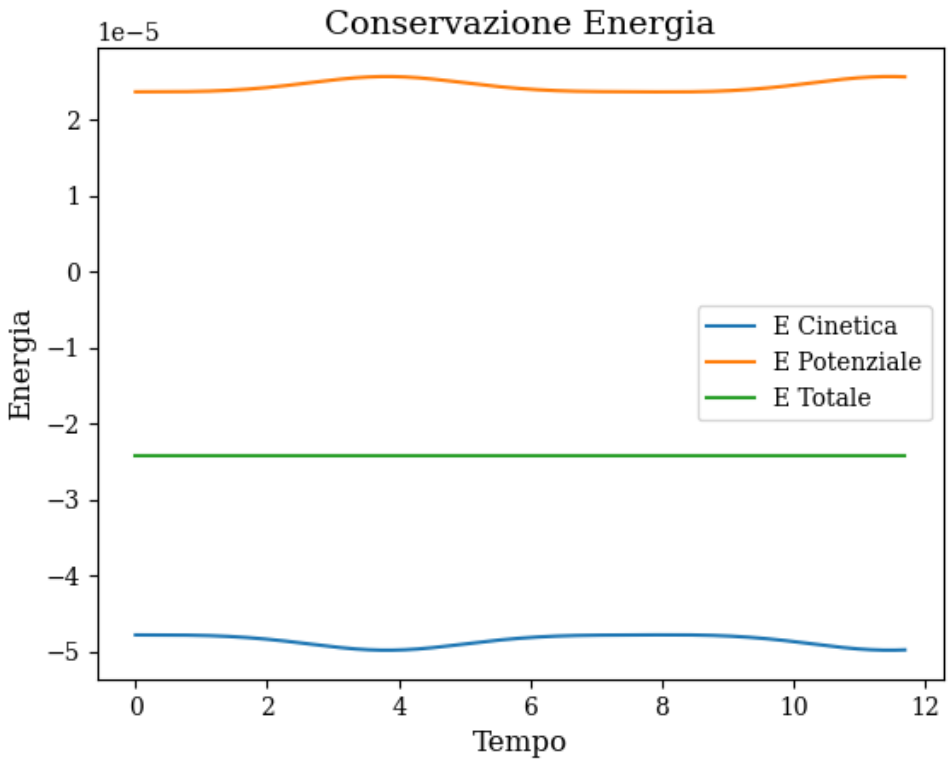

Figure 3: Bilancio Energetico per N = 1000 , $t_0 = 0.01$

Che ovviamente varia nel caso in cui si cambino i parametri N e t_0, ciò non di meno

si noti che la scala è nell'ordine di 10^{-5}, quindi una variazione minuscola che comunque segue l'andamento dell'orbita

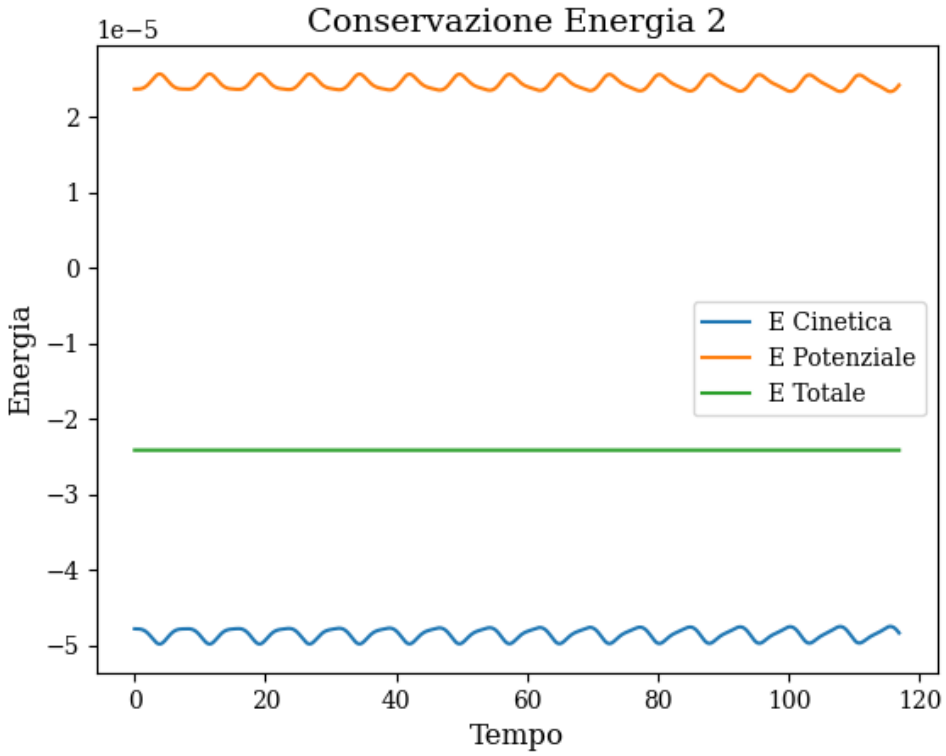

Figure 4: Bilancio Energetico per N = 10000 , $t_0 = 0.01$

Aumentando N aumenta il tempo di un fattore proporzionale ma l'oscillazione non ha variazioni nell'intensità. Come ci si aspetta il programma non genera variazioni energetiche in funzione di N per t_0 fissato.
Bisogna infine controllare come varia la generazione di time step adattivo lungo il periodo, per farlo si procede analogamente a quanto appena mostrato

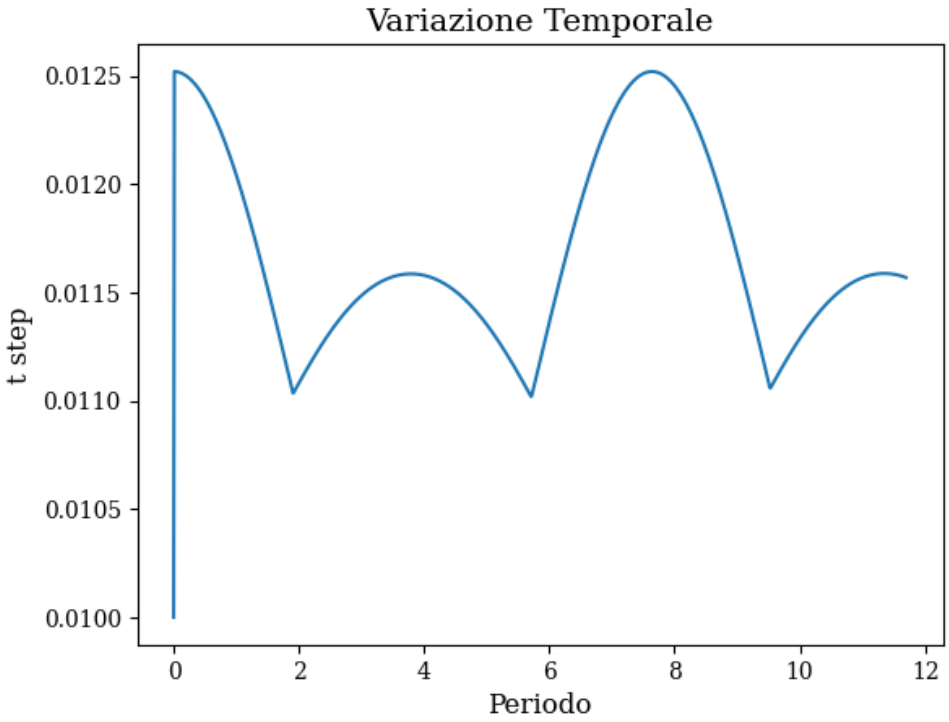

Figure 5: Time step generato dall'equazione adattiva e periodo per N = 1000 , $t_0 = 0.01$

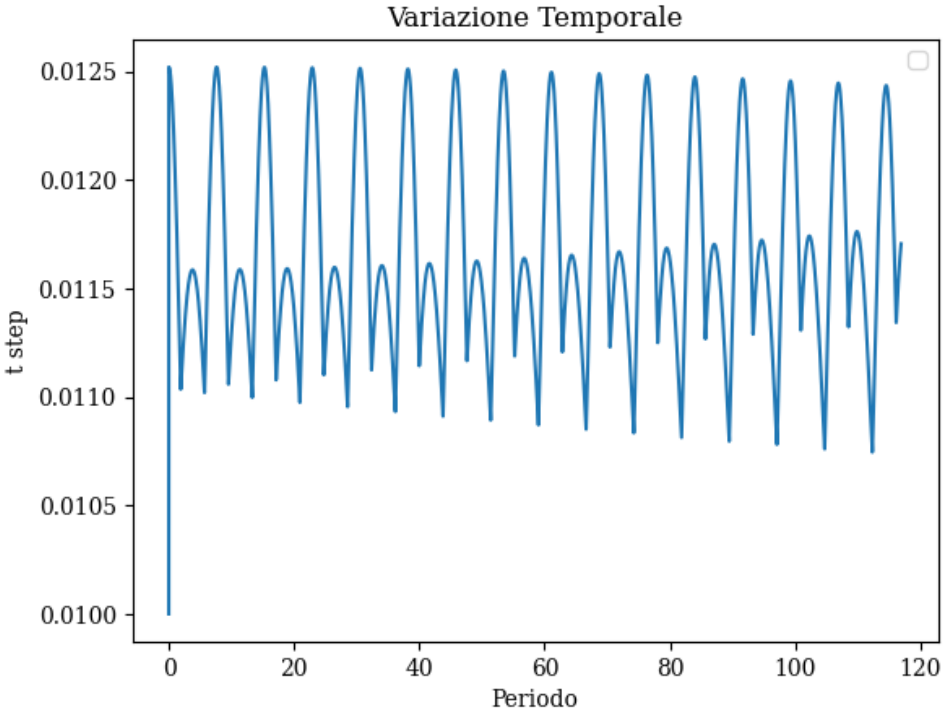

Figure 6: Time step generato dall'equazione adattiva e periodo per N = 10000 , $t_0 = 0.01$

Analizzando il primo grafico dei due (Figure 5) si nota un iniziale drop temporale, che è causato dalla scelta di $t_0 = 0.01$ e nel momento in cui il ciclo ha inizio avviene lo "spostamento" per tempo più consono in base all'orbita. La distribuzione del time step sta entro 15 millesimi, ovviamente, visto anche il grafico dell'energia per $N = 1000$ non si potevano

osservare variazioni nella densità di punti nell'orbita.
Come si nota vi è una certa variazione tra i due grafici e nel secondo (Figure 6) in particolare c'è un difetto di time step sul periodo totale, tuttavia si parla di una variazione nell'ordine del decimo di millesimo (10^{-4}), in un intervallo temporale in cui vengono compiute $120/7,7 = 15,6 orbite$, l'insieme di questa informazione, comparata con quelle ricavate da (Figure 4) fanno comprendere immediatamente perchè non si sia vista una variazione significativa nell'orbita che è rimasta sostanzialmente circolare anche per $N = 10000$

▷ ***Orbita Ellittica*** :

Ora si sostituisce semplicemente le condizioni di orbita ellittica

```
v_y_0 = sqrt((2*GM)*(2/x_0 - 1/(2*(x_0))))#velocita' iniziale per
orbita ellittica
v_x_0 = 0
```

in cui implicitamente si sta imponendo il centro di massa delle stelle in uno dei due fuochi dell'ellisse. Ci si chiederà se è legittima tale scelta, nel momento in cui si deve decidere come applicare la condizione però ci sono solo due possibilita: mettere entrambe le stelle in un fuoco o mettere ciascuna in un fuoco, si è optato per la prima giacchè le orbite sembravano più stabili. Di seguito viene riportato il grafico con semiasse maggiore $a = 2x_0 = 10UA$, $N = 1000$, τ variabile con valore iniziale $t_0 = 0,01$ e $A = 0.009$

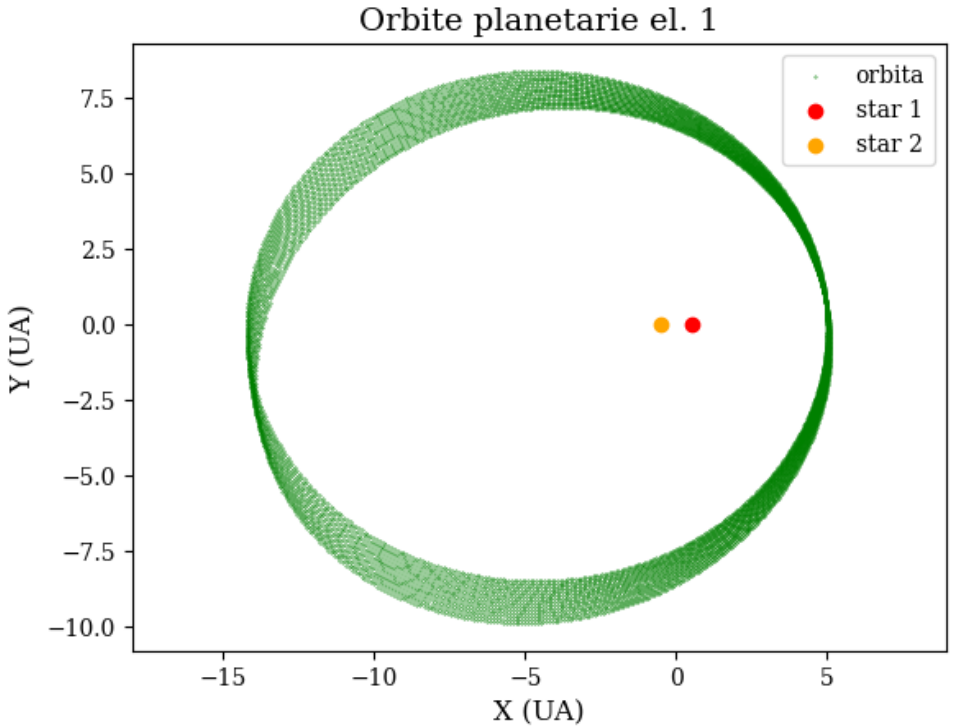

Figure 7: Orbita ellittica sistema binario di stelle, pianeta singolo

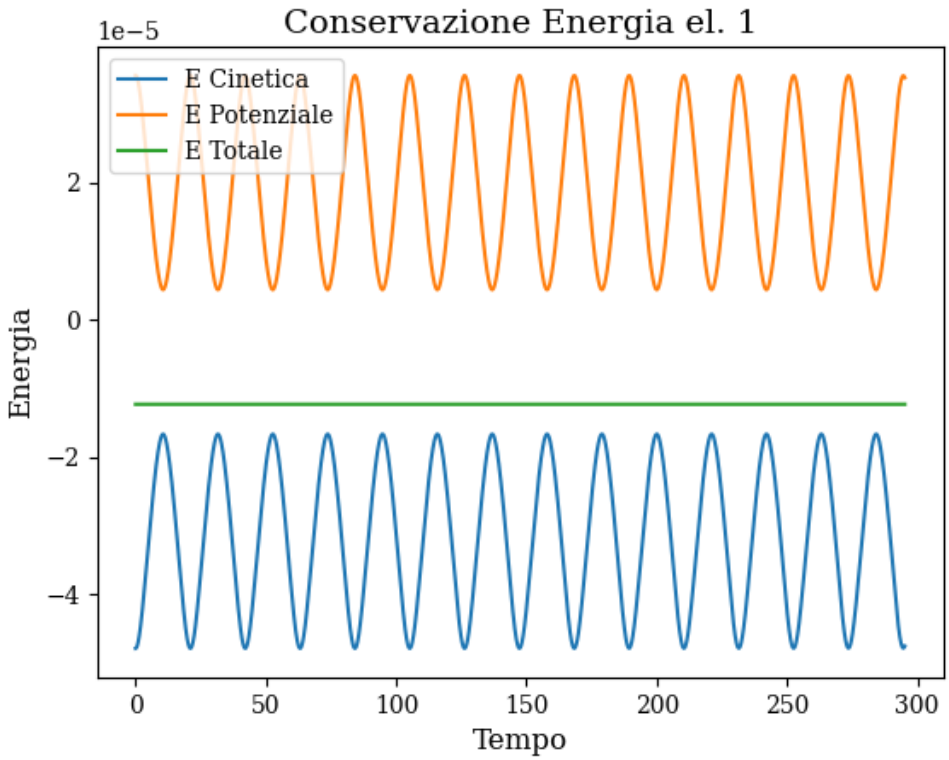

Figure 8: Bilancio Energetico orbita ellittica per N = 10000 , $t_0 = 0.01$, $A = 0.009$

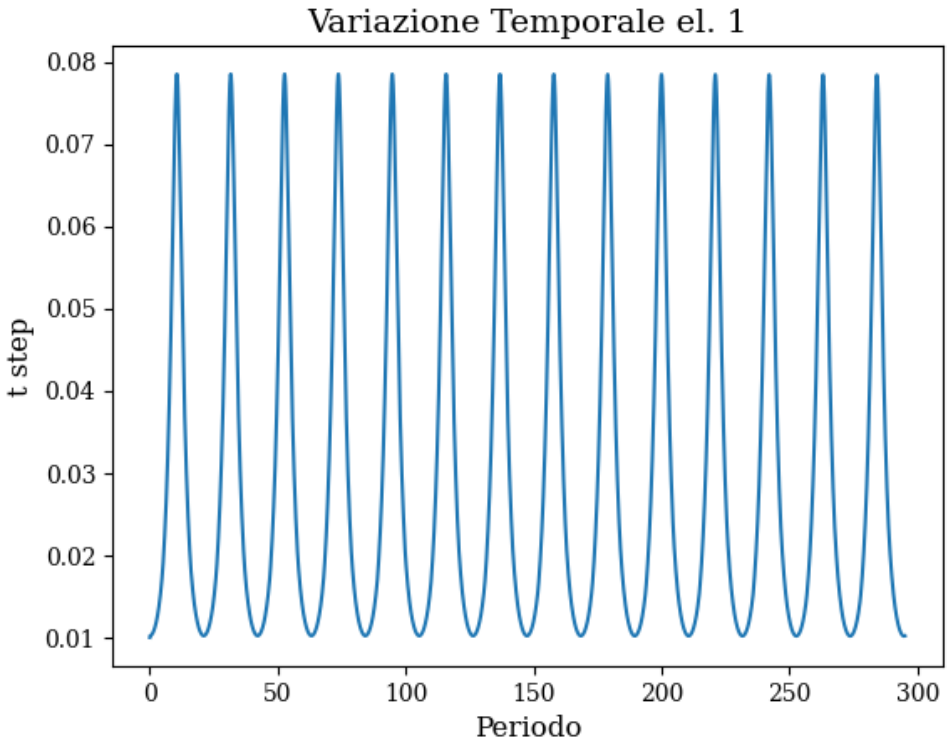

Figure 9: Time step orbite ellittiche generato dall'equazione adattiva e periodo per N = 10000 , $t_0 = 0.01$, $A = 0.009$

Le conclusioni stimate in precedenza continuano a valere in buona approssimazione, il caso di orbita circolare è un caso particolare dell'orbita ellittica anche se ciò non implica

un viceversa intrinseco non pare ci siano osservazioni da fare in merito a grafici di tempo ed energia. Diverso è il caso del grafico dell'orbita, che si presenta con un accumulo di punti in prossimità degli apsidi per restringimento della traiettoria, mentre si vede una densità minore ed un aumento del moto di precessione nel resto dell'orbita. Questo comportamento è normale quando si pesenta un minimo di instabilità nell'orbita, anche Mercurio presenta un andamento simile. Il grafico è composto da 300 periodi orbitali (300 anni), quindi viene considerata stabile per lunghi periodi di tempo.
Per verificare il comportamento di pianeti di pari massa a diversa distanza si è eseguito un plot a mezzo di una funzione che racchiudesse il verlet con condizione di ellitticità:

```
def orbita(x_0 = 5 , N = 10000 , d = 1):
.
.#resto del programma
.
    T_a = np.linspace(0 , 1, N+1) #Tempo arbitrario per evitare che il
   plot presenti le energie dipendenti dai tempi di ciascun pianeta

  state = [ X , Y , E_pot , E_k , E_tot , t , T_a ]
    return state

#costruisco i grafici
plt.rcParams['font.family'] = 'serif'
x_in = np.linspace(2, 20, 5)
for i in x_in:
  state = orbita(d = 1, x_0 = i, N = 10**3)
  #plt.plot(state[0], state[1], label= 'x0 = ' + str(i))

  #plt.figure()
  plt.title('Orbite planetarie ellittiche' , fontsize = 14)
  plt.plot(state[0], state[1], label= 'x0 = ' + str(i))
  plt.xlabel('X (UA)' , fontsize = 12)
  plt.ylabel('Y (UA)' , fontsize = 12)
  plt.axis('equal')
plt.scatter(d/2 , y_0, color="red", label = "star 1")
plt.scatter(-d/2 , y_0, color="orange", label = "star 2")
plt.legend(fontsize = 10)
```

Questo codice genera la famiglia di orbite ellittiche aventi il centro di massa delle due stelle in uno dei due fuochi

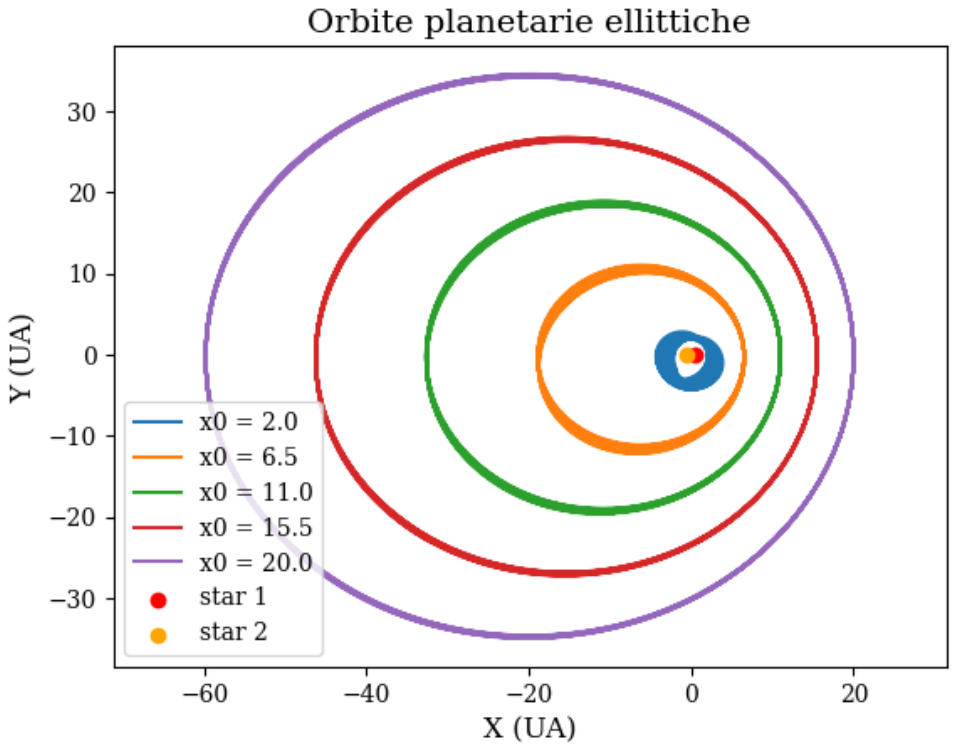

Figure 10: Orbite ellittiche sistema binario di stelle, cinque pianeti, $d_p = 4.5UA$

Come si nota la distanza iniziale tra un pianeta e l'altro è 4,5 UA, i pianeti più vicini presentano profilo deviato rispetto all'orbita ellittica ma è attribuibile alla vicinanza del pianeta con la stella.
Si esegue dunque una stima dell'energia cinetica in un grafico omnicomprensivo, con periodo in unità temporali arbitrarie per vederne il profilo senza che sia dipendente dal periodo di ogni pianeta (ciascuno presenterà periodo proprio differente da un altro a parità di tempo). Stampando opportunamente, dividendo il codice per evitare collisioni con gli altri grafici si ottiene:

```
    plt.title('Energia Potenziale Sistema Stellare' , fontsize = 14)
    plt.plot(state[6], state[2], label = 'E.P.p. x0 = ' + str(i))

    #differente finestra di codice
    plt.title('Energia Cinetica Sistema Stellare' , fontsize = 14)
    plt.plot(state[6] , state[3], label = 'E.C.p. x0 = ' + str(i))

    #differente finestra di codice
    plt.title('Energia Totale Sistema Stellare' , fontsize = 14)
    plt.plot(state[6] , state[4], label = 'E Totale')

    #per ciascun plot fuori dal ciclo si scrive
plt.xlabel('Tempo (unit arbitrarie)' , fontsize = 12)
plt.ylabel('Energia 'inserie nome energia' ' , fontsize = 12)
plt.legend(fontsize = 10)
```

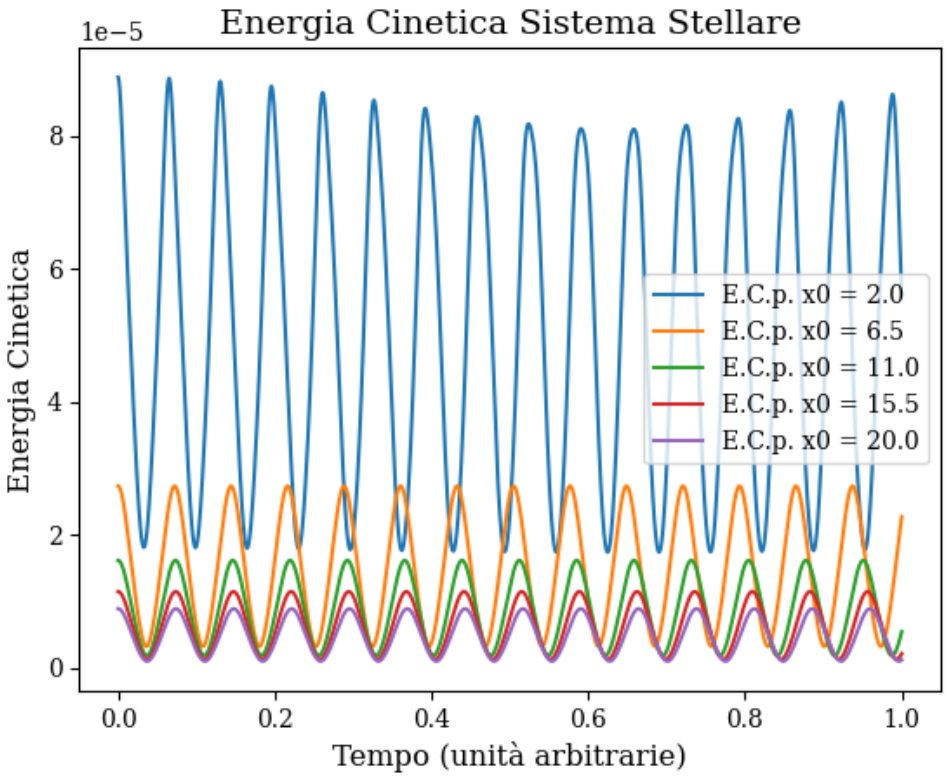

Figure 11: Energia cinetica sistema stellare binario con unità temporali arbitrarie in cui il Tempo rappresenta il periodo, $d_p = 4.5UA$

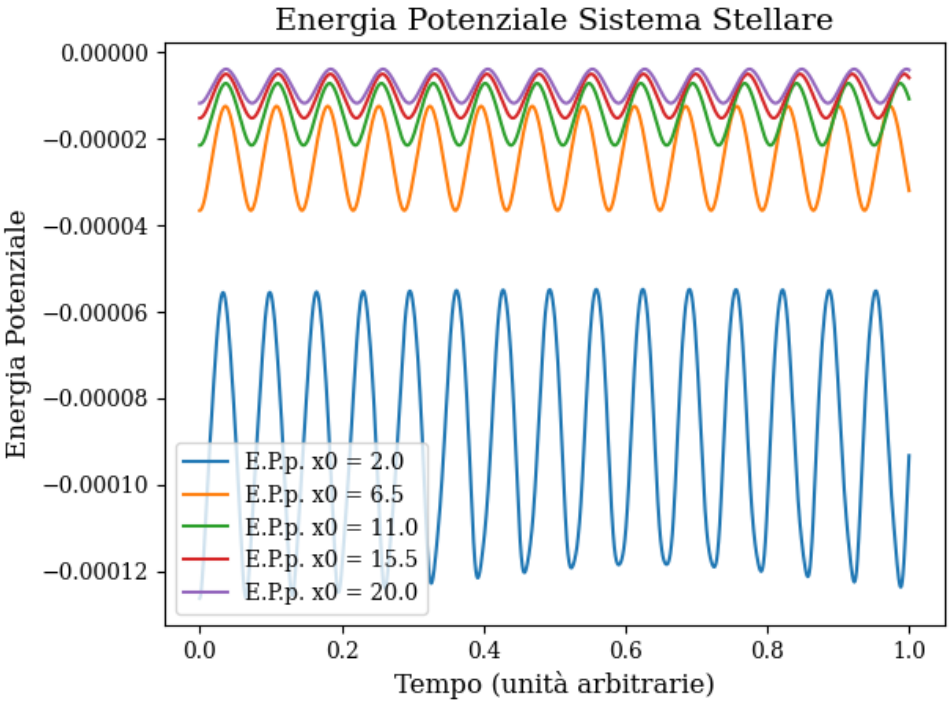

Figure 12: Energia potenziale sistema stellare binario con unità temporali arbitrarie in cui il Tempo rappresenta il periodo, $d_p = 4.5UA$

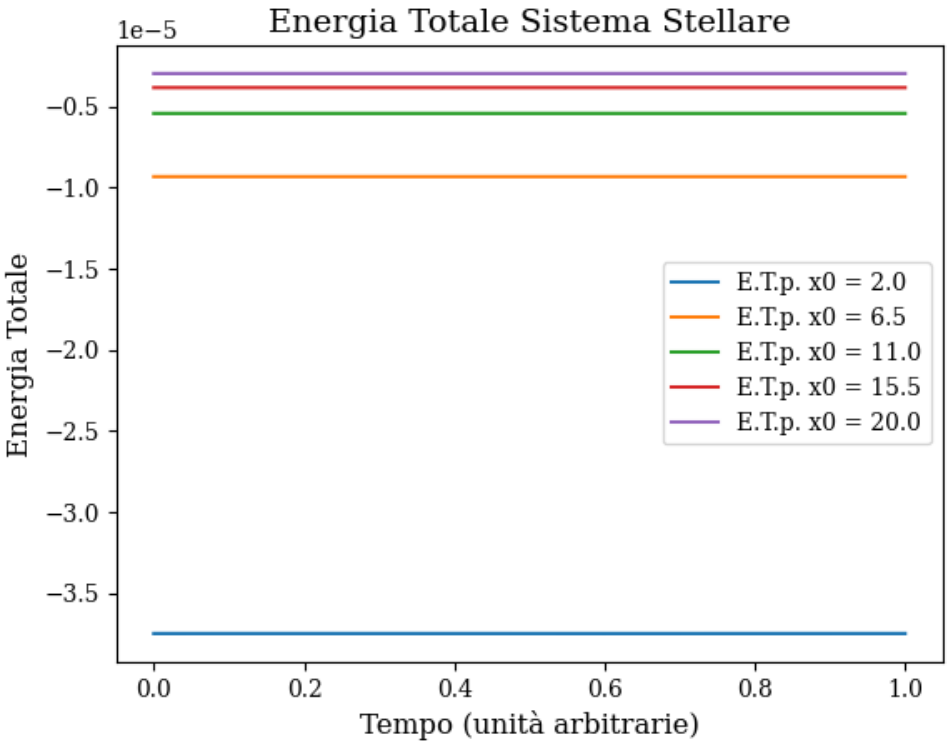

Figure 13: Energia totale sistema stellare binario con unità temporali arbitrarie in cui il Tempo rappresenta il periodo, $d_p = 4.5UA$

E come è evidente la somma dell'energia totale per i corrispettivi pianeti è costante

Problema B:

▷ ***Interazione con orbita stabile*** :

Bisognava determinare una condizione in cui un pianeta massivo interagiva con un secondo più leggero che si comportava da satellite. La scelta che si è fatto è stata quella di porre le masse di Terra e Luna in unità astronomiche semplificate, come nel caso precedente le distanze sono frazioni della distanza astronomica Terra-Sole. Le unità di tempo sono archi di anni.
In prima approssimazione si pone il problema con soli due corpi interagenti come di seguito:

```
    #velocity-Verlet pianeta con interazione Luna
import numpy as np
from numpy import cos, pi, sin
from math import sqrt, asin, atan2
import matplotlib.pyplot as plt

# imposto le grandezze iniziali e funzione
d = 0.002 #distanza Terra Luna
m = 3E-6
m_2 = 0.012*m
GM = 4*pi**2
GM_l = GM*m_2
GM_t = GM*m

N = 10000
t = 0.001
T = np.linspace(0, N*t, N+1)

x_0 = 1  #posizione Terra rispetto all'origine
x_l_0 = x_0 + d #posizione Luna

#definisco gli elementi array di posto zero
y_0 = 0.0

r_t = sqrt((x_0)**2 + (y_0)**2) #distanza Sole_Terra
r_l = sqrt((x_l_0)**2 + (y_0)**2) #distanza Sole-Luna
r_lt = sqrt((x_l_0 - x_0)**2 + (y_0)**2) #distanza Luna-Terra

v_y_0 = sqrt((GM)/r_t) + sqrt((GM_l)/r_lt) #velocit Terra
v_x_0 = 0

v_y_l = sqrt((GM)/r_l) + sqrt((GM_t)/r_lt)#velocit Luna
v_x_l = 0

#forza Sole su Luna ist. 0
F_x_l = -GM*(x_l_0)/(r_l**3)
F_y_l = -GM*y_0/(r_l**3)
```

```
#forza Sole su Terra ist. 0
F_x_t = -GM*x_0/(r_t**3)
F_y_t = -GM*y_0/(r_t**3)

#Forza Luna su Terra ist. 0
F_x_lt = GM_l*(x_l_0 - x_0)/(r_lt**3)
F_y_lt = GM_l*y_0/(r_lt**3)

#Forza Terra su Luna ist. 0
F_x_tl = -GM_t*(x_l_0 - x_0)/(r_lt**3)
F_y_tl = -GM_t*y_0/(r_lt**3)

#Forza totale sulla Terra ist. 0
F_x_tot = F_x_lt + F_x_t
F_y_tot = F_y_lt + F_y_t

#preparo i vettori in cui andr a salvare i risultati dei cicli e li
   inizializzo

X1 = np.zeros(N+1) #Distanza Terra
X1[0] = x_0
Y1 = np.zeros(N+1)
Y1[0] = y_0
r_1 = np.zeros(N+1)
r_1[0] = r_t

X2 = np.zeros(N+1) #Distanza Luna
X2[0] = x_l_0
Y2 = np.zeros(N+1)
Y2[0] = y_0
r_2 = np.zeros(N+1)
r_2[0] = r_l

r =np. zeros(N+1) #Distanza T_L
r[0] = r_lt

F_x_1 = np.zeros(N+1) #Forza S-Terra
F_x_1[0] = F_x_t
F_y_1 = np.zeros(N+1)
F_y_1[0] = F_y_t

F_x_2 = np.zeros(N+1) #Forza S-Luna
F_x_2[0] = F_x_l
F_y_2 = np.zeros(N+1)
F_y_2[0] = F_y_l

F_x_3 = np.zeros(N+1) #Forza Luna-Terra
F_x_3[0] = F_x_lt
```

```
F_y_3 = np.zeros(N+1)
F_y_3[0] = F_y_lt

F_x_4 = np.zeros(N+1) #Forza Terra - Luna
F_x_4[0] = F_x_tl
F_y_4 = np.zeros(N+1)
F_y_4[0] = F_y_tl

F_x = np.zeros(N+1) #Forza totale su Terra
F_x[0] = F_x_1[0] + F_x_3[0]
F_y = np.zeros(N+1)
F_y[0] = F_y_1[0] + F_y_3[0]

F_xl = np.zeros(N+1) #Forza totale su Luna
F_xl[0] = F_x_2[0] + F_x_4[0]
F_yl = np.zeros(N+1)
F_yl[0] = F_y_2[0] + F_y_4[0]

v_x1 = np.zeros(N+1) #Velocit Terra
v_x1[0] = v_x_0
v_y1 = np.zeros(N+1)
v_y1[0] = v_y_0

v_x2 = np.zeros(N+1) #Velocit Luna
v_x2[0] = v_x_l
v_y2 = np.zeros(N+1)
v_y2[0] = v_y_l

E_p_t = np.zeros(N+1) #Potenziale Terra
E_p_t[0] = - (m*GM*(1/r_1[0])) - (m*GM_l*(1/r[0]))

E_p_l = np.zeros(N+1) #Potenziale Luna
E_p_l[0] = - (m_2*GM*(1/r_2[0])) - (m_2*GM_t*(1/r[0]))

E_k_t = np.zeros(N+1) #Cinetica Terra
E_k_t[0] =  0.5*m*(v_x1[0]**2 + v_y1[0]**2)

E_k_l = np.zeros(N+1) #Cinetica Luna
E_k_l[0] =  0.5*m_2*(v_x2[0]**2 + v_y2[0]**2)

E_tot_t =np.zeros(N+1) #Energia Totale Terra
E_tot_t[0] = E_p_t[0] + E_k_t[0]

E_tot_l =np.zeros(N+1) #Energia Totale Luna
E_tot_l[0] = E_p_l[0] + E_k_l[0]

#eseguo a tutti gli effetti il velocity-Verlet
for n in range(N):
```

```
X1[n+1] = X1[n] + t*v_x1[n] + 0.5*F_x[n]*t**2 #Posizione aggiornata
 Terra
Y1[n+1] = Y1[n] + t*v_y1[n] + 0.5*F_y[n]*t**2
r_1[n+1] = sqrt( X1[n+1]**2 + Y1[n+1]**2)

X2[n+1] = X2[n] + t*v_x2[n] + 0.5*F_xl[n]*t**2 #Posizione aggiornata
 Luna
Y2[n+1] = Y2[n] + t*v_y2[n] + 0.5*F_yl[n]*t**2
r_2[n+1] = sqrt( X2[n+1]**2 + Y2[n+1]**2)

r[n+1] = sqrt( (X1[n+1]-X2[n+1])**2 + (Y1[n+1] - Y2[n+1])**2)
 #Posizione relativa T-

F_x_1[n+1] = -(GM)*(X1[n+1])/r_1[n+1]**3 #Forza Sole su Terra
F_y_1[n+1] = -(GM)*(Y1[n+1])/r_1[n+1]**3

F_x_2[n+1] = -(GM)*(X2[n+1])/r_2[n+1]**3 #Forza Sole su Luna
F_y_2[n+1] = -(GM)*(Y2[n+1])/r_2[n+1]**3

F_x_3[n+1] = -GM_l*((X1[n+1] - X2[n+1]))/(r[n+1]**3) #Forza Luna su
 Terra
F_y_3[n+1] = -GM_l*((Y1[n+1] - Y2[n+1]))/(r[n+1]**3)

F_x_4[n+1] = -GM_t*((X2[n+1] - X1[n+1]))/(r[n+1]**3) #Forza Terra su
 Luna
F_y_4[n+1] = -GM_t*((Y2[n+1] - Y1[n+1]))/(r[n+1]**3)

F_x[n+1] = F_x_1[n+1] + F_x_3[n+1] #Forza totale su Terra
F_y[n+1] = F_y_1[n+1] + F_y_3[n+1]

F_xl[n+1] = F_x_2[n+1] + F_x_4[n+1] #Forza totale su Luna
F_yl[n+1] = F_y_2[n+1] + F_y_4[n+1]

v_x1[n+1] = v_x1[n] + 0.5*t*(F_x[n]+F_x[n+1]) #velocit Terra
v_y1[n+1] = v_y1[n] + 0.5*t*(F_y[n]+F_y[n+1])

v_x2[n+1] = v_x2[n] + 0.5*t*(F_xl[n]+F_xl[n+1]) #velocit Luna
v_y2[n+1] = v_y2[n] + 0.5*t*(F_yl[n]+F_yl[n+1])

E_p_t[n+1] = - (m*GM*(1/r_1[n+1])) - (m*GM_l*(1/r[n+1])) #Potenziale
 su Terra
E_p_l[n+1] = - (m_2*GM*(1/r_2[n+1])) - (m_2*GM_t*(1/r[n+1]))
 #Potenziale Luna

E_k_t[n+1] =  0.5*m*(v_x1[n+1]**2 + v_y1[n+1]**2) #Cinetica Terra
E_k_l[n+1] =  0.5*m_2*(v_x2[n+1]**2 + v_y2[n+1]**2) #Cinetica Luna

E_tot_t[n+1] = E_p_t[n+1] + E_k_t[n+1]  #Energia Totale Terra
```

```
  E_tot_l[n+1] = E_p_l[n+1] + E_k_l[n+1] #Energia Totale Luna


plt.rcParams['font.family'] = 'serif'
#costruisco i grafici
f1 = plt.title('Orbita Interazione Planetaria' , fontsize = 14)
plt.plot(X1 , Y1 ,  label = "Orbita Terra" )
plt.plot(X2 , Y2 ,  label = "Orbita Luna" )
plt.scatter(0 , y_0, color="orange", label = "Sole")
plt.xlabel('X (UA)' , fontsize = 12)
plt.ylabel('Y (UA)', fontsize = 12)
plt.axis('equal')
plt.legend( loc=(0.75, 0.82) , fontsize = 9)
plt.show()

f5 = plt.title('Ingrandimento Interazione' , fontsize = 14)
plt.scatter(X1 , Y1 ,  label = "Orbita Terra" , s = 1 , linewidths=5 )
plt.scatter(X2 , Y2 ,  label = "Orbita Luna" , s = 1 , linewidths=5 )
plt.axis([0.95, 1.02, 0, 0.5])
plt.xlabel('X (UA)' , fontsize = 12)
plt.ylabel('Y (UA)', fontsize = 12)
plt.legend( fontsize = 9)
plt.show()

f2 = plt.title('Conservazione Energia Terra' , fontsize = 14)
plt.plot(T , E_p_t, label = 'Potenziale Terra')
plt.plot(T , E_k_t, label = 'Cinetica Terra')
plt.plot(T , E_tot_t , label = 'Energia Tot Terra')
plt.xlabel('Tempo' , fontsize = 12)
plt.ylabel('Energia' , fontsize = 12)
plt.legend(fontsize = 9)
plt.show()

f3 = plt.title('Conservazione Energia Luna' , fontsize = 14)
plt.plot(T , E_p_l, label = 'Potenziale Luna')
plt.plot(T , E_k_l, label = 'Cinetica Luna')
plt.plot(T , E_tot_l , label = 'Energia Totale')
plt.xlabel('Tempo' , fontsize = 12)
plt.ylabel('Energia' , fontsize = 12)
plt.legend( fontsize = 9)
plt.show()

f4 = plt.title('Orbita Relativa', fontsize = 14)
plt.plot(X1 - X2 , Y1 - Y2 , color ="green", label = "Orbita Luna su
    Terra" )
plt.scatter(0 , 0, color="blue", label = "Terra")
plt.xlabel('X (UA)', fontsize = 12)
plt.ylabel('Y (UA)', fontsize = 12)
```

```
plt.axis('equal')
plt.legend(loc=(0.75, 0.82) , fontsize = 9)
plt.show()
```

Da cui si ottengono i rispettivi grafici:

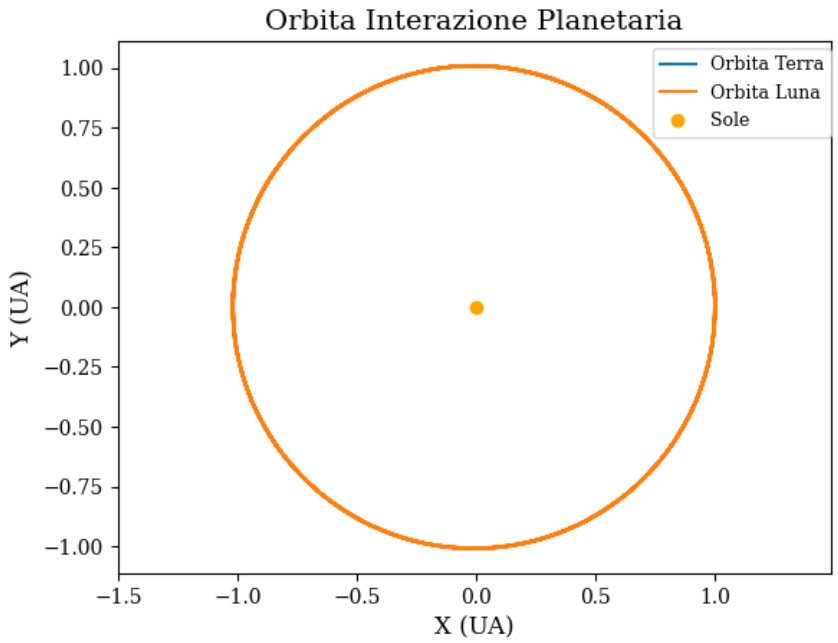

Figure 14: Orbita Terra e Luna per $d = 0.002UA$ con orbite sovrapposte, $N = 10000$e, $x_0 t = 1UA$

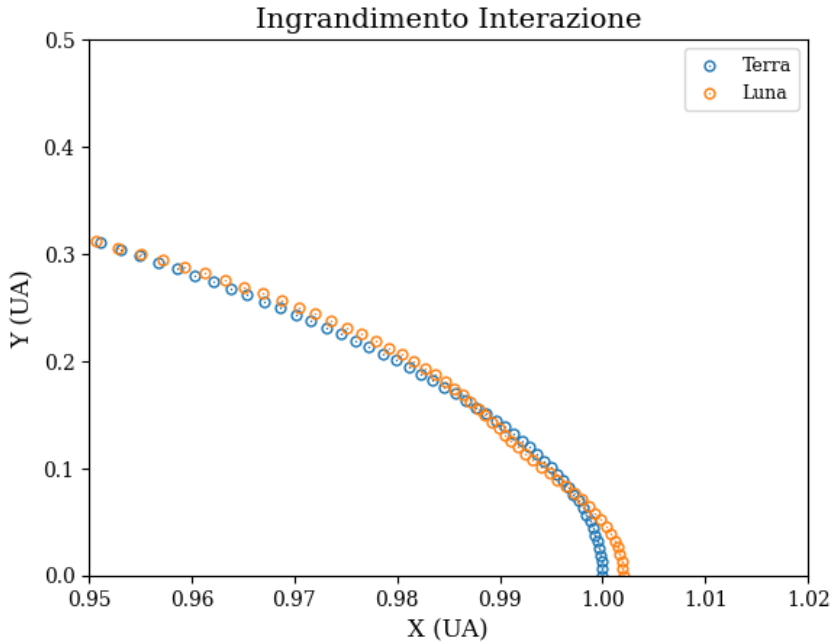

Figure 15: Ingrandimento Orbita Terra e Luna per $d = 0.002UA$ con orbite sovrapposte, $N = 10$e, $x_0 = 1UA$

Come si nota dai grafici precedenti vi è un problema in merito alla sovrapposizione delle orbite che non consente di verificare l'andamento della Luna attorno alla Terra. Per

risolvere questo problema si è realizzato un ingrandimento della Figura 14
Ulteriormente si introduce l'orbita relativa della Luna attorno alla Terra

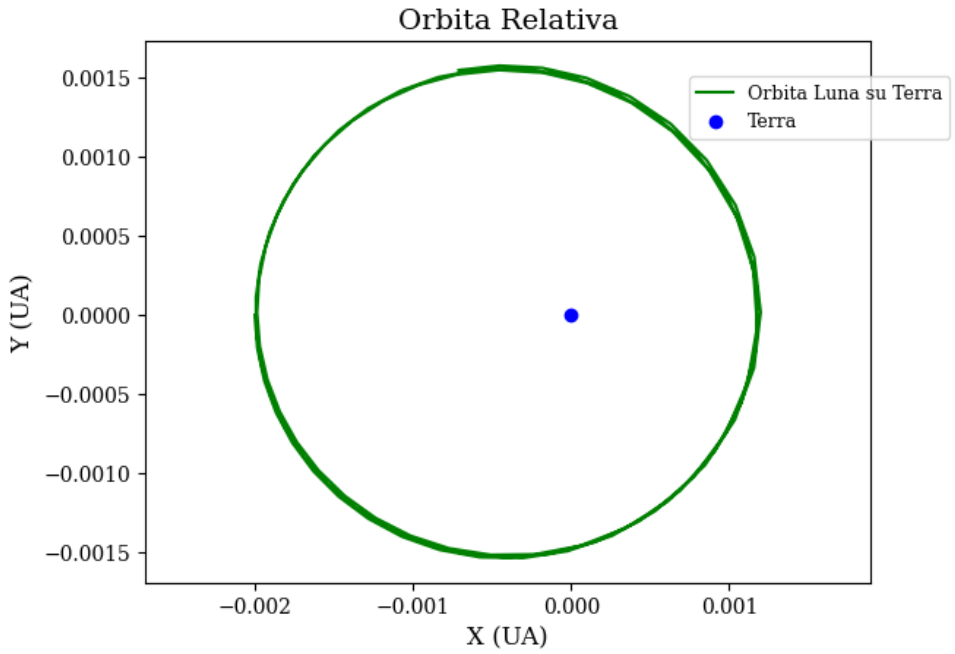

Figure 16: Orbita relativa Luna attorno alla Terra per $d = 0.002UA$, $N = 10000$e, $x_0 = 1UA$

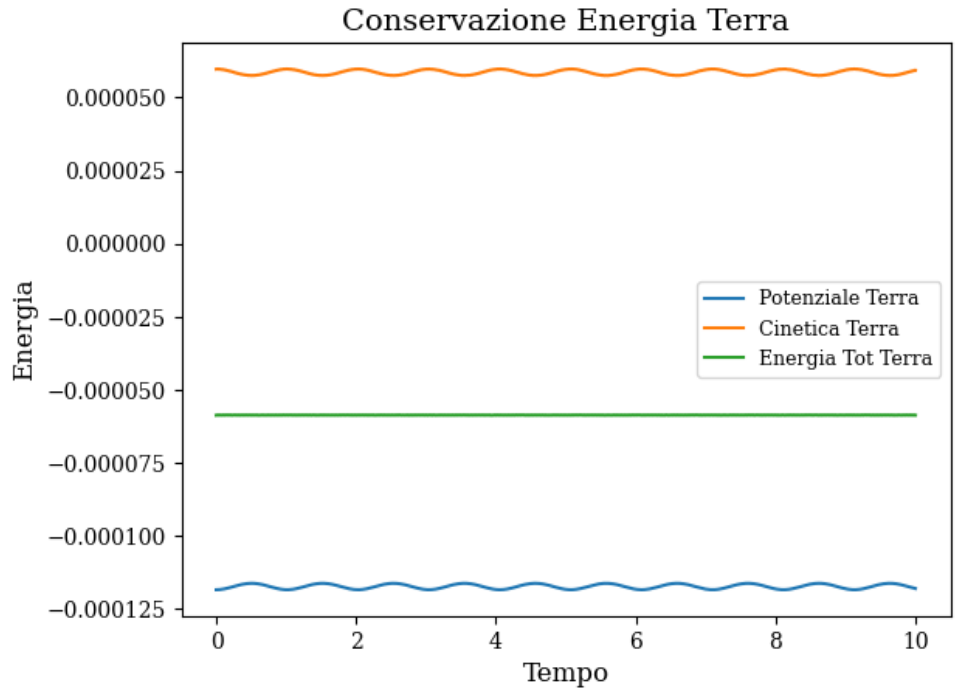

Figure 17: Energia Terra per $d = 0.002UA$, $N = 10000$e, $x_0 = 1UA$

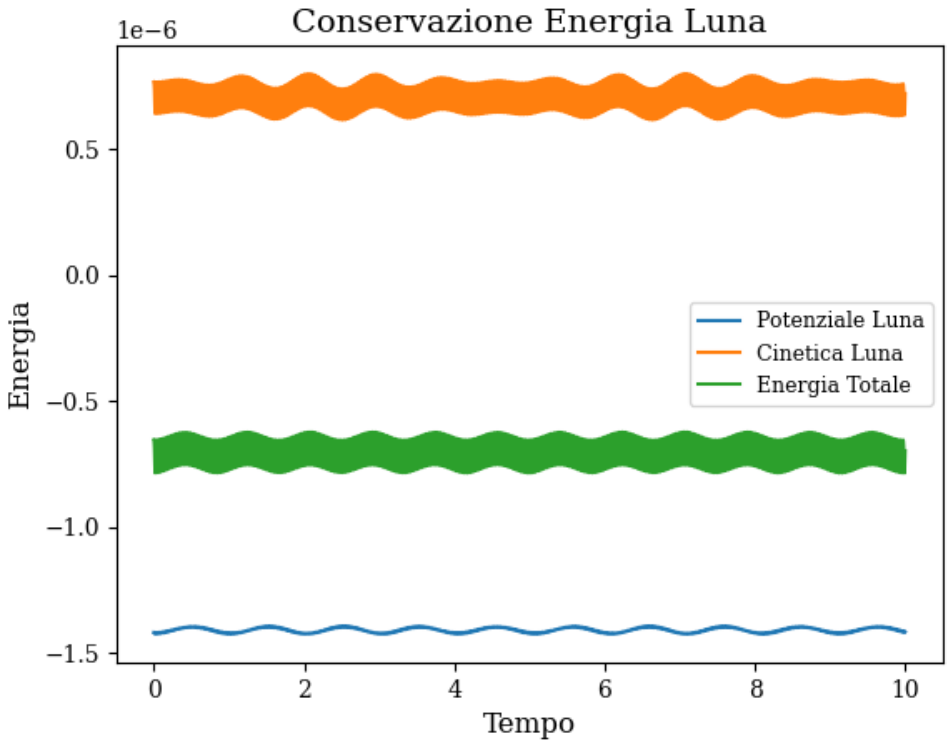

Figure 18: Energia Luna per $d = 0.002UA$, $N = 10000$, problema energia totale

L'energia totale della Luna, per quanto visto in Figura 17, non è costante. Per cercare di comprendere questo fenomeno si è rianalizzato il codice e si è deciso di realizzarne uno ulteriore per comprendere se vi fossero problemi relativi alla distanza.
Si fa ausilio di una funzione

```
def inter_orbit(x_0 = 1 , N = 10000 , d = 0.0009):
.
.
. #resto del programma
.#prima di chiudere il ciclo introduco

     state = [ X1 , Y1 , X2 , Y2 , E_p_t , E_k_t , E_p_l , E_k_l ,
   E_tot_t , E_tot_l ]
  #state_l =[X2 , Y2 , E_p_l , E_k_l , E_tot_l]
  return state

plt.rcParams['font.family'] = 'serif'
d_in = np.linspace(0.001, 0.007, 4)
for i in d_in:
  state = inter_orbit(d = i, x_0 = 1, N = 10**4)

#costruisco i grafici
  f1 = plt.title('Orbita Sistema Terra-Luna' , fontsize = 14)
  plt.plot(state[2] , state[3] ,  label = "d L_T = " +str(i))
plt.plot(state[0] , state[1] , color ="green", label = "Orbita Terra")
plt.scatter(0 , 0, color="orange", label = "Sole")
plt.xlabel('X (UA)' , fontsize = 12)
```

```
plt.ylabel('Y (UA)' , fontsize = 12)
plt.axis('equal')
plt.legend()
```

da cui si sono ricavati i grafici:

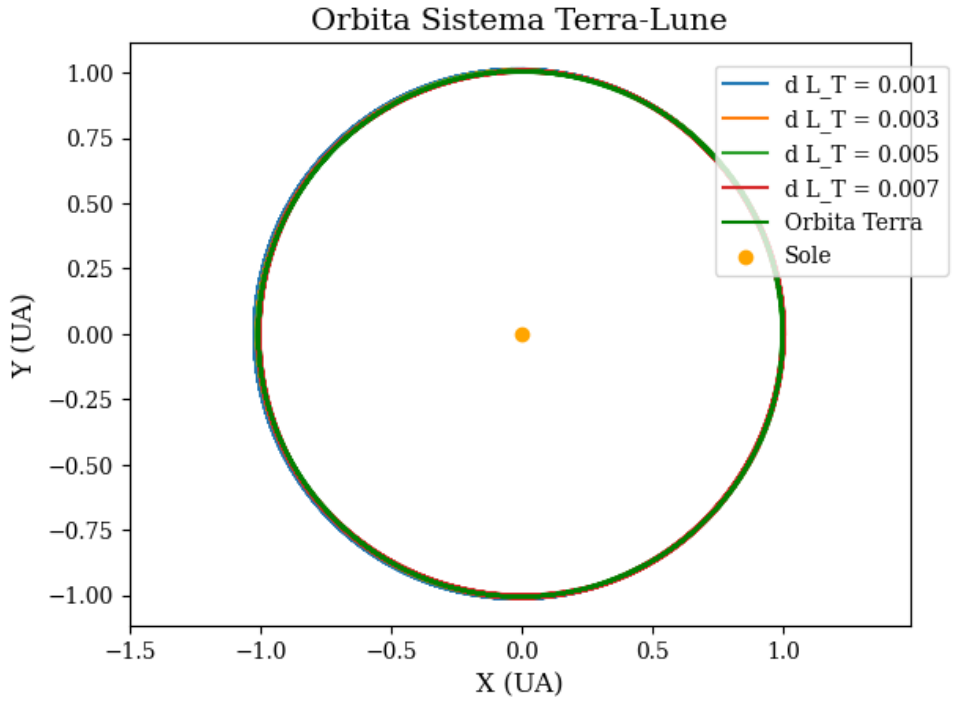

Figure 19: Orbita Terra e Luna per d variabile con orbite sovrapposte, $N = 10000$, $x_0 = 1UA$

```
    f4 = plt.title('Orbite Relative lune')
    plt.plot(state[2]  - state[0] , state[3] - state[1] , label = "d = "
   + str(i))
plt.scatter(0 , 0, color="blue", label = "Terra")
plt.xlabel('X (UA)')
plt.ylabel('Y (UA)')
plt.axis('equal')
plt.legend()
```

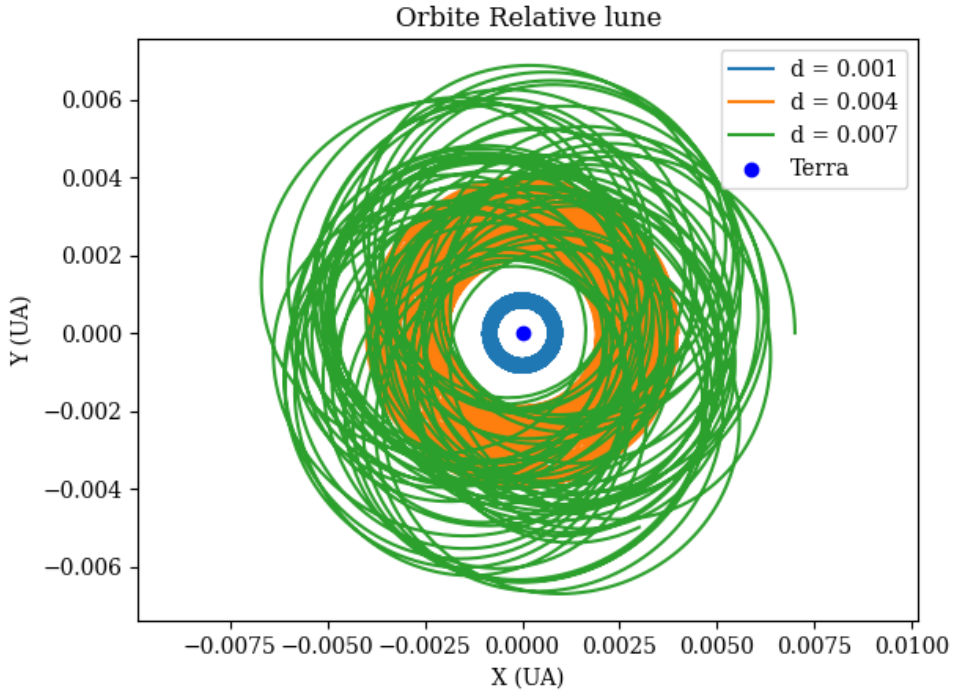

Figure 20: Orbita relativa Luna attorno alla Terra per d variabile , $N = 10000$, $x_0 = 1UA$

Come possiamo notare per $d = 0,007UA$ si perde la stabilità dell'orbita, o meglio l'orbita è stabile ma con grandi oscillazioni. Per i successivi grafici:

```
    f21 = plt.title('Conservazione Energia Terra')
    plt.plot(state[10] , state[4], label = 'Potenziale Terra' +str(i))
    plt.plot(state[10] , state[5], label = 'Cinetica Terra')
    plt.plot(state[10] , state[8] , label = 'Energia Totale')
plt.xlabel('Tempo')
plt.ylabel('Energia')
plt.legend()
```

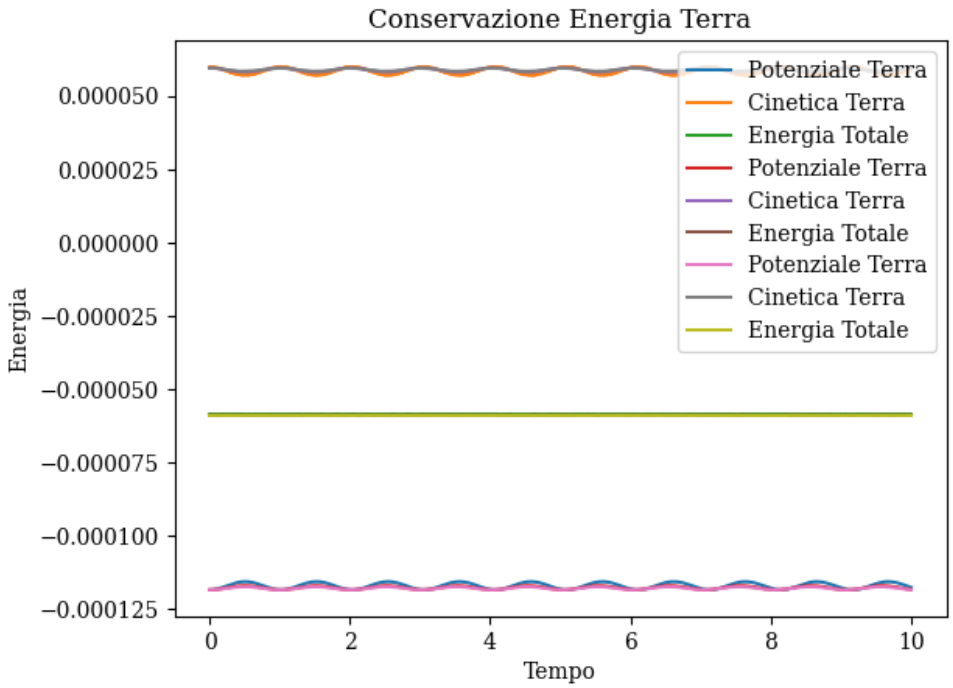

Figure 21: Bilancio energetico Terra per d variabile , $N = 10000$, $x_0 = 1UA$

Si fa altrettanto con la Luna:

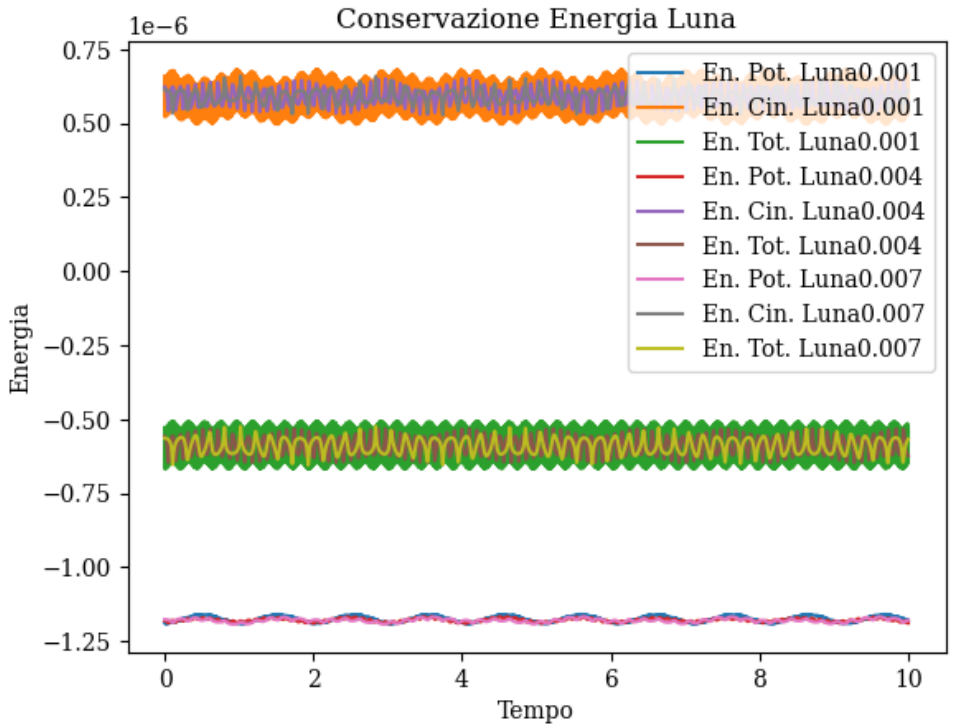

Figure 22: Bilancio energetico Luna per d variabile , $N = 10000$, $x_0 = 1UA$

Si noti che, per quanto difficile possa essere visualizzare a causa dei colori, l'energia della Luna in questi casi risulta sempre negativa (come ci si aspetta per orbite stabili), minore dell'energia cinetica e in modo del tutto analogo a figura 18 non pare costante, anzi: nonostante un generale andamento ordinato sono presenti troppe fluttuazioni. Per quanto

è vero che le scale di energia sono molto piccole comunque la dimensione dell'oscillazione dell'energia totale nel tempo è comparabile con l'energia cinetica. Questo fenomeno non è problematico se considerato quanto detto in *Introduzione*, tuttavia per quanto visto in precedenza si potrebbe ritenere che esista un errore nel programma che ne sia la causa che però non è stato rilevato.
Per quanto etto si deve assumere che il programma funzioni e semplicemente le interazioni coi pianeti comportino una grande fluttuazione dell'energia cinetica se comparata con quella gravitazionale. Si è realizzato un nuovo programma che però presentasse i dati di massa di Terra e Saturno, disponendo il pianeta più massivo più lontano dal Sole per comparare qualitativamente i risultati dell'energia. I dati introdotti sono: $x_0 = 5UA, d = 0.05UA, N = 12000, t = 0.001$ anche in questo caso il pianeta massivo presenta bilancio energetico perfetto mentre quello leggero no. Se i pianeti stanno su orbite stabili intorno al Sole il loro bilancio energetico, calcolato attraverso il Verlet, non presenta problemi, quindi questo fa supporre, per quanto detto prima, che il bilancio energetico sull'orbita del pianeta satellite resta minore di zero ma non costante, esattamente come la Luna per la Terra nel precedente programma. Di seguito i grafici

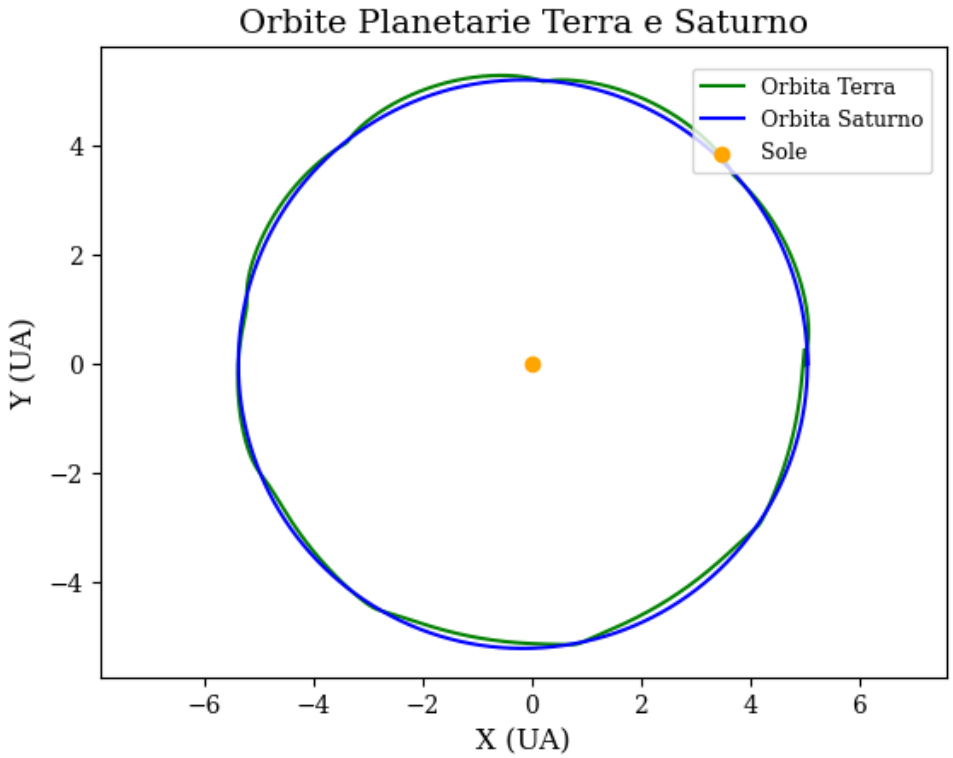

Figure 23: Orbita Terra e Saturno per $d = 0.05UA$, $N = 12000$, $x_0 = 5UA$

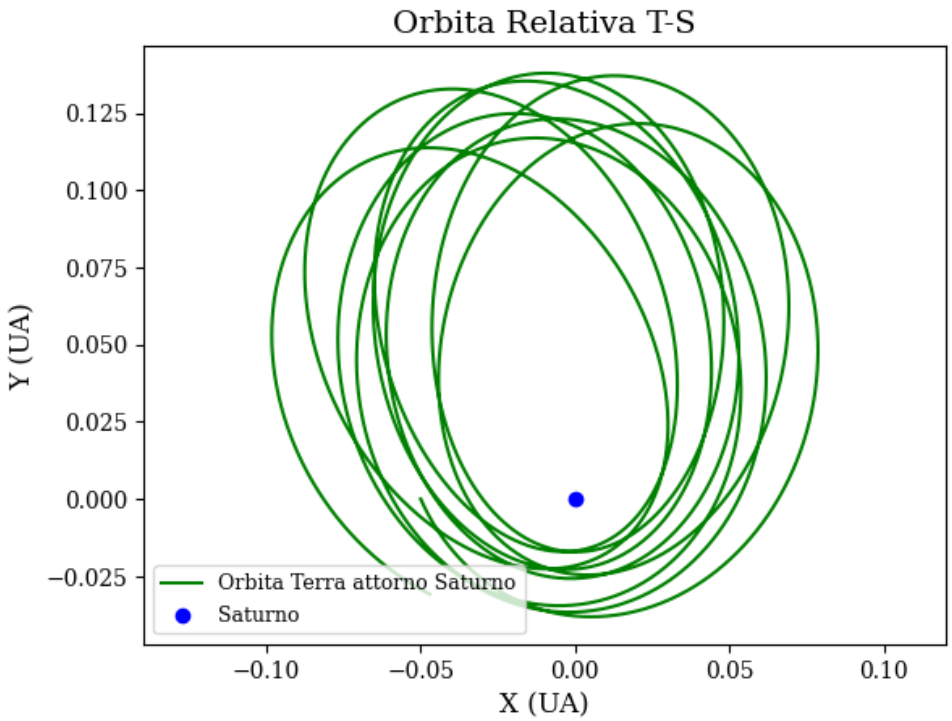

Figure 24: Orbita relativa Terra attorno Saturno per $d = 0.05UA$, $N = 12000$, $x_0 = 5UA$

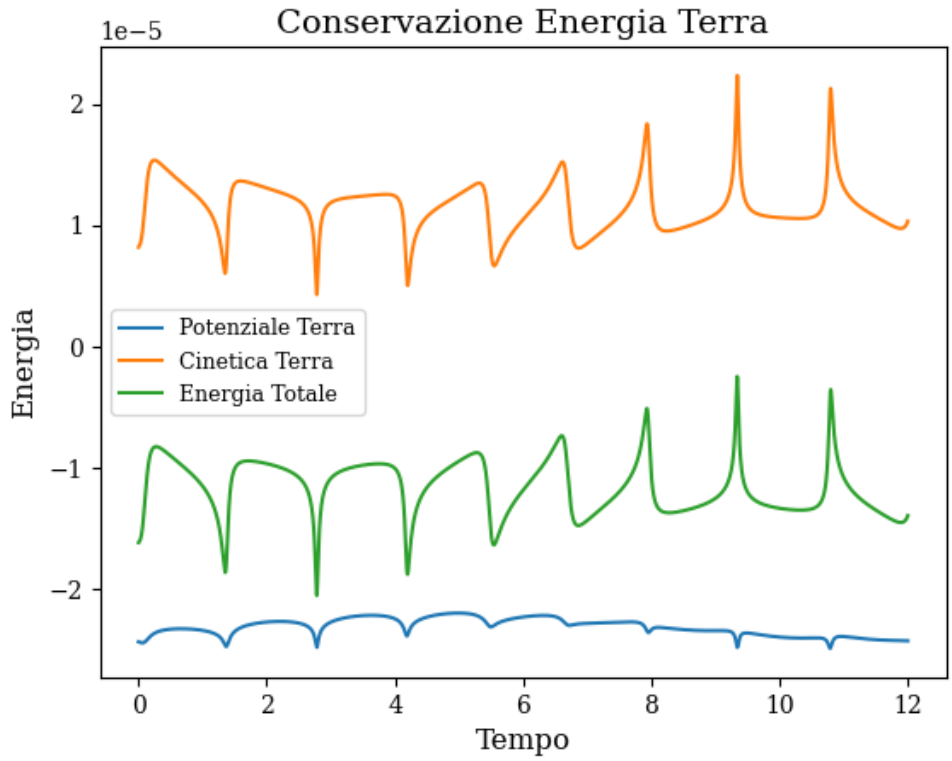

Figure 25: Energia Terra attorno Saturno per $d = 0.05UA$, $N = 12000$, $x_0 = 5UA$

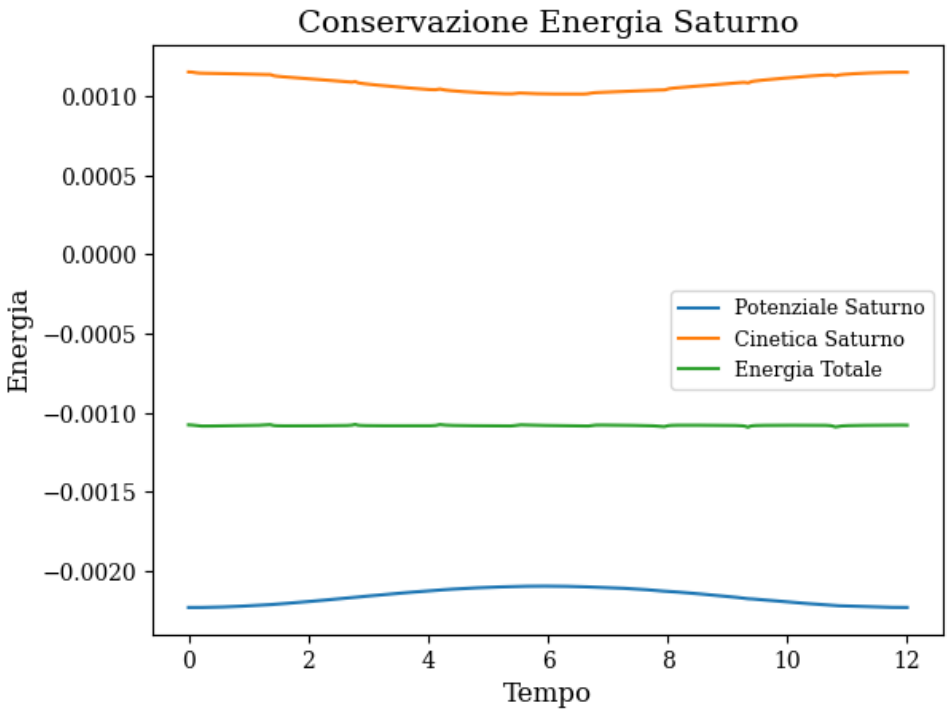

Figure 26: Energia Saturno per $d = 0.05UA$, $N = 12000$, $x_0 = 5UA$

▷ ***Interazione con orbita instabile*** :

Determinare un caso in cui un pianeta destabilizzasse l'orbita di un altro. Per determinare per quali parametri un'orbita sia considerabile instabile si sono introdotti 3 valori nella funzione di dimensione inferiore al millesimo di Unità astronomica:

```
d_in = np.linspace(0.0001, 0.00075, 3)
```

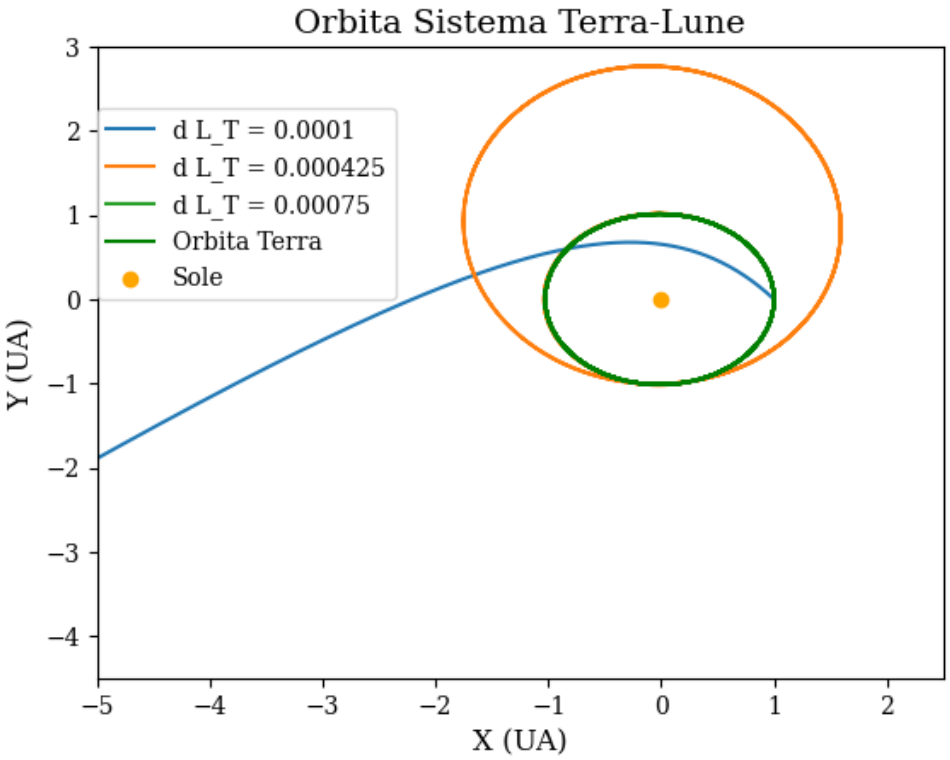

Figure 27: Orbita Terra e Luna per d variabile, $N = 10000$, $x_0 = 1UA$

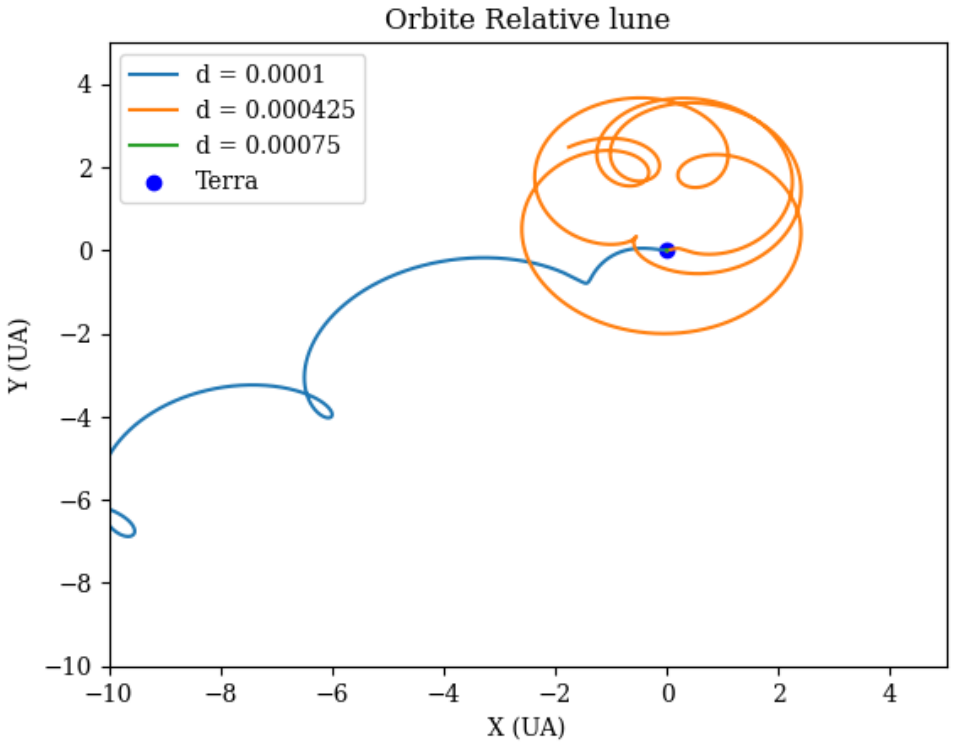

Figure 28: Orbita relativa Luna attorno alla Terra per d variabile , $N = 10000$, $x_0 = 1UA$

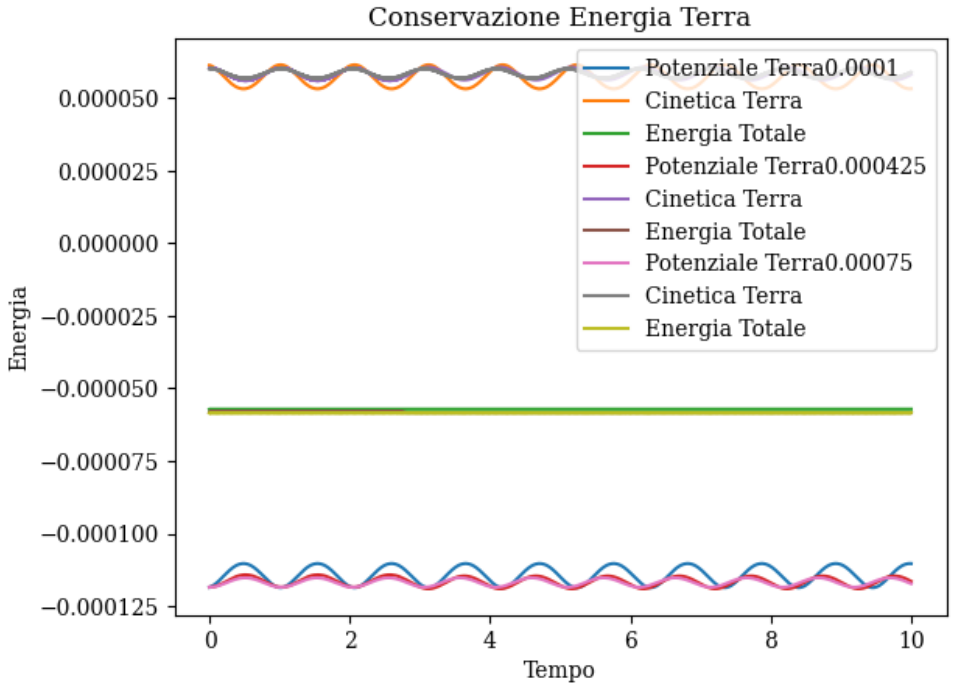

Figure 29: Bilancio energetico Terra per d variabile , $N = 10000$, $x_0 = 1UA$

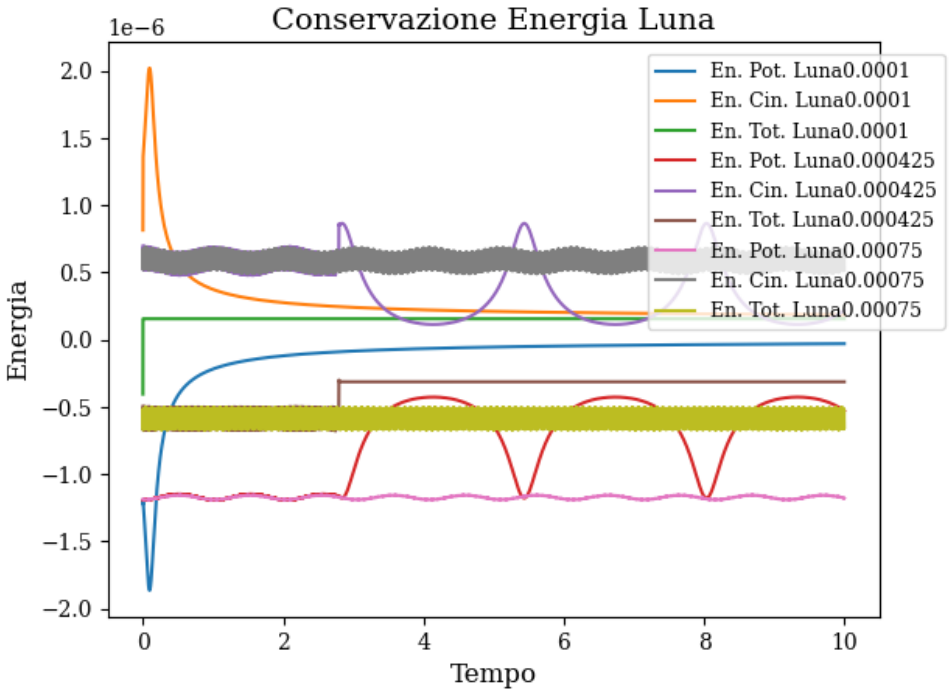

Figure 30: Bilancio energetico Luna per d variabile , $N = 10000$, $x_0 = 1UA$

Ancora una volta si nota come l'energia totale della Terra si conserva mentre quella delle Lune si comporta in modo anomalo, il chè non ci dà problemi visto che dovevamo determinare una instabilità data dall'interazione dei pianeti-satelliti. Difatti per orbite legate l'energia totale risulta minore di zero mentre per quelle slegate sarà maggiore.

3 Conclusioni

Sembrano essere stati raggiunti gli scopi preposti dalla consegna iniziale, resta un problema fondamentale in merito all'analisi degli errori che non è stata portata a termine per motivi tempistici. Difatti vi sono stati vari problemi con i programmi che hanno comportato ritardo sulla stesura della relazione.
Buona prassi sarebbe stata quella di confrontare rispettivamente le eccenticità delle orbite tra quelle calcolate e quelle reali, l'andamento dell'energia nel tempo in funzione dei valori reali, il periodo di rivoluzione reale comparato con quello calcolato, l'andamento dell'angolo in funzione del tempo. Coi soli dati disponibili si può dire che è stato possibile calcolare ogni caso richiesto ma senza sapere con quali margini il programma stia sbagliando sul calcolo.