

Indice

Introduzione	1
1 Richiami di teoria e definizioni	2
2 Teoria dei metodi numerici	5
2.1 Metodo di Eulero	10
2.2 Metodo di Verlet	11
2.3 Metodi Runge-Kutta	12
3 Modelli fisico-matematici	14
3.1 Moto di un corpo in un campo gravitazionale	14
3.1.1 Moto di un satellite soggetto ad attrito	16
3.2 Pendolo composto	18
4 Codice e risultati computazionali	19
4.1 Simulazioni di moto nel campo gravitazionale terrestre	25
4.1.1 Test di caduta	25
4.1.2 Orbite planetarie	27
4.1.3 Satellite in un'atmosfera	33
4.2 Simulazioni del pendolo composto	39
4.2.1 Panoramica di evoluzioni	39
4.2.2 Caos e diagramma delle biforazioni	41
5 Conclusioni	48
Bibliografia	50

Introduzione

Le equazioni differenziali sono strumenti molto potenti che svolgono un ruolo fondamentale nel descrivere una moltitudine di fenomeni di diversa natura e sono pertanto tra le equazioni più usate in Fisica, Matematica e Ingegneria. In ambiti applicativi è estremamente complesso risolvere un'equazione differenziale. Infatti, tranne in rari casi, la soluzione esplicita non si riesce a determinare analiticamente. In questa relazione di tirocinio andremo a descrivere alcune tecniche ed algoritmi basilari che si usano per approssimare la soluzione di sistemi di equazioni differenziali ordinarie. In particolare, verranno confrontate le diverse prestazioni degli algoritmi numerici in due casi di dinamica non lineare: 1) il moto di un satellite in un'atmosfera e 2) l'evoluzione temporale di un pendolo composto sottoposto ad una forzante esterna. Verrà discussa l'importanza degli algoritmi stessi nello studio e nella simulazione dei sistemi dinamici.

Per una maggiore comprensione dei metodi di simulazione e per un maggior confronto tra i vari algoritmi, abbiamo implementato alcuni metodi *ex novo*, creando un integratore per i due sistemi fisici considerati. Il linguaggio utilizzato è il Python.

Capitolo 1

Richiami di Teoria e Definizioni

Di seguito forniamo alcuni richiami sulla teoria delle equazioni differenziali ordinarie.

Definizione 1: Un'equazione differenziale ordinaria è una relazione della forma :

$$F \left(t, y(t), y'(t), \dots, y^{(n)}(t) \right) = 0,$$

dove $F : D \subseteq \mathbb{R}^{n+2} \rightarrow \mathbb{R}$ è una funzione di $n+2$ variabili.

Definizione 2: Un'equazione differenziale è detta in forma normale quando è espressa come:

$$y^{(n)}(t) = G \left(t, y(t), y'(t), y''(t), \dots, y^{(n-1)}(t) \right)$$

dove $G : S \subseteq \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ è una funzione di $n + 1$ variabili. Non è sempre possibile esprimere un'equazione differenziale in tale forma.

Definizione 3: Chiamiamo problema di Cauchy di un'equazione differenziale il seguente problema :

$$\left\{ \begin{array}{l} y^{(n)}(t) = G \left(t, y(t), y'(t), \dots, y^{(n-1)}(t) \right) \\ y(t_0) = y_0 \\ y'(t_0) = y_1 \\ \vdots \\ y^{(n-1)}(t_0) = y_{n-1} \end{array} \right.$$

dove:

- t_0 indica il riferimento della variabile indipendente
- y_0, y_1, \dots, y_{n-1} sono i valori della variabile dipendente e delle sue derivate in t_0 .

Si può dimostrare che la soluzione di un problema di Cauchy esiste ed è unica sotto poche semplici ipotesi nella funzione G [5]. Si noti che un problema alle condizioni iniziali è esprimibile tramite un problema di Cauchy.

Definizione 4: Una soluzione di un problema di Cauchy è una funzione $U(t)$ tale che verifichi l'equazione differenziale e rispetti le n condizioni iniziali.

Definizione 5: Un sistema di equazioni differenziali ordinarie del primo ordine è un sistema della forma:

$$\left\{ \begin{array}{l} y'_1(t) = G_1(t, y_1(t), y_2(t), \dots, y_n(t)) \\ y'_2(t) = G_2(t, y_1(t), y_2(t), \dots, y_n(t)) \\ \vdots \\ y'_n(t) = G_n(t, y_1(t), y_2(t), \dots, y_n(t)) \end{array} \right.$$

a cui sono associati n vincoli per i valori iniziali.

Anche per i sistemi si può dimostrare l'esistenza della soluzione e della sua unicità, sotto semplici ipotesi riguardo le funzioni G_i [5].

In forma vettoriale, il sistema sopra descritto può essere espresso in maniera compatta come segue:

$$\vec{Y}'(t) = \vec{G}(t, \vec{Y})$$

con \vec{Y}' , \vec{G} e \vec{Y} vettori n dimensionali.

Una soluzione del sistema sarà una funzione $\vec{Y} = \{ y_i \}$ a valori vettoriali tale che le sue componenti soddisfino il sistema.

Possiamo notare che, data un'equazione di ordine n ,

$$y^{(n)}(t) = f(t, y(t), y'(t), \dots, y^{(n-1)}(t))$$

essa può essere scritta come un sistema equivalente di equazioni dell'primo ordine con l'introduzione di n variabili :

$$\begin{aligned} y_1 &= y(t) \\ y_2 &= y'(t) \\ y_3 &= y''(t) \\ &\vdots \\ y_n &= y^{(n-1)}(t) \end{aligned}$$

$$y_n' = f(t, y_1, y_2, y_3, \dots, y_n)$$

Pertanto, d'ora in poi considereremo solo sistemi di equazioni differenziali ordinarie dell'primo ordine, poiché un'equazione di ordine n sarà sempre riconducibile a un insieme di equazioni di ordine 1.

Capitolo 2

Teoria dei metodi numerici

Esistono varie tecniche per ricavare degli algoritmi per approssimare la soluzione di un'equazione differenziale (o di un sistema di equazioni differenziali). È possibile:

- 1) utilizzare la formula di Taylor per una approssimazione locale;
- 2) sfruttare il teorema fondamentale del calcolo integrale e le formule di quadratura;
- 3) usare il rapporto incrementale per approssimare le derivate.

Tutti i metodi si basano sull'idea di discretizzare l'intervallo di integrazione mediante un insieme di punti:

$$x_n = x_0 + hn \quad n = 0, 1, \dots, N$$

dove h rappresenta il passo di integrazione della variabile indipendente, cioè la distanza tra due punti consecutivi (in questa relazione non affronteremo il caso dei metodi che usano un passo non costante).

La soluzione del problema, cioè la funzione che soddisfa l'equazione, viene approssimata anch'essa da un insieme di punti:

$$\{ y_n \mid n = 0, 1, \dots, N \}.$$

Tali punti sono generati dai metodi numerici di integrazione e sono delle approssimazioni dei valori esatti $y(x_i)$.

I metodi numerici differiscono per il modo in cui approssimano la funzione soluzione, quindi per come si generano i punti $\{y_n\}$.

In generale, dato il problema di Cauchy:

$$\begin{cases} y'(x) = f(x, y(x)) \\ y(x_0) = y_0 \end{cases}$$

e introdotta una *discretizzazione* dell'intervallo I , mediante i punti

$$x_i = x_0 + hi \quad i = 0, 1, \dots, N \text{ con } h \text{ costante,}$$

possiamo integrare l'equazione differenziale tra due punti consecutivi ottenendo la seguente relazione:

$$\int_{x_i}^{x_{i+1}} y'(x) dx = \int_{x_i}^{x_{i+1}} f(x, y(x)) dx.$$

Applicando ora il teorema fondamentale del calcolo integrale otteniamo la formula generale:

$$y(x_{i+1}) = y(x_i) + \varphi(f(x_i, y(x_i)), f(x_{i+1}, y(x_{i+1})), h, f) + R(x_i, y(x_i), h, f)$$

dove:

- φ è una formula di quadratura per la funzione $f(x, y(x))$.
- $R(x_i, y(x_i), h, f)$ è l'errore della formula di quadratura φ .

In forma più compatta possiamo scrivere, introducendo la funzione H , la seguente relazione:

$$y(x_{i+1}) = H(y(x_i), f(x_i, y(x_i)), f(x_{i+1}, y(x_{i+1})), h, f) + R(x_i, y(x_i), h, f)$$

In altri termini, $R(x_i, y(x_i), h, f)$ è chiamato errore locale di troncamento ed esso rappresenta la differenza tra il valore esatto della soluzione $y(x_{i+1})$ e il valore fornito dal metodo numerico, quando questo operi su valori esatti $y(x_j)$ e non con i valori approssimati y_j [2].

In generale si distinguono 2 tipi di metodi :

1) Metodi One-Step : che per generare il successivo punto y_{n+1} necessitano solo la conoscenza del punto y_n , descritti nella pagina precedente.

2) Metodi Multi-Step : che per generare il successivo punto y_{n+1} necessitano la conoscenza di m punti $y_n, y_{n-1}, \dots, y_{m+1}$ (che non tratteremo in questa relazione).

La formula generale esplicita dei metodi One-step è ottenibile, alternativamente, dalla seguente relazione esatta (applicando la formula di Taylor):

$$y(x + h) = y(x) + h \phi(x, y(x); h; f) + R(x, y(x); h; f)$$

dove:

- h è il passo di integrazione.
- ϕ è una funzione.
- $R(x, y(x); h; f)$ è l'errore locale.

Discretizzando la formula e trascurando l'errore locale si ottiene l'equazione alle differenze per i metodi One-step

$$y_{i+1} = y_i + h \phi(x_i, y_i; h; f)$$

Per ogni algoritmo numerico è necessario conoscere il suo errore di approssimazione e quindi stimare l'errore commesso dal metodo. L'errore di questi algoritmi è dato da 2 contributi [2] :

$$\text{Errore} = \text{Errore di Troncamento} + \text{Errore di Propagazione}$$

- *L'Errore di Troncamento* è dovuto al fatto che stiamo discretizzando la funzione con un insieme finito di punti.
- *L'Errore di Propagazione* è dovuto al fatto che nei calcolatori stiamo lavorando con un'aritmetica finita.

Noi tratteremo solo l'errore di troncamento e possiamo subito notare che in esso influisce sia l'omissione ad ogni passo dell'errore locale che l'errore dovuto all'utilizzo dei valori approssimati y_i , invece che dei $y(x_i)$, per $i > 1$.
Si introduce quindi l'errore globale di troncamento.

Definizione 7: Si definisce errore globale di troncamento il valore:

$$e_{i+1} = y(x_{i+1}) - y_{i+1}$$

Definizione 8: Un metodo numerico si dice convergente se vale che [2]:

$$\lim_{h \rightarrow \infty} y_i = y(x), \quad i = \frac{x-x_0}{h}$$

Definizione 9: Un metodo numerico si dice consistente se vale che [2] :

$$\lim_{h \rightarrow 0} \frac{R(x, y(x); h; f)}{h} = 0$$

La consistenza garantisce che per $h \rightarrow 0$ l'errore locale tenda a zero più velocemente di h . In generale si può dimostrare che la consistenza unita al concetto di stabilità di un algoritmo determina la convergenza di un metodo numerico [2].

Possiamo ora definire il concetto di ordine di un metodo.

Definizione 10: Un metodo numerico ha ordine p se risulta che

$$R(x, y(x); h ; f) = O(h^{p+1})$$

Un risultato importante che si può dimostrare per i metodi One-step esplicativi, cioè quegli schemi One-step che a destra dell'uguaglianza non richiedono il calcolo di $f(x_i, y_{i+1})$, che possiedono la proprietà che la loro funzione di incremento sia lipschitziana di costante L rispetto al secondo argomento è che l'errore di discretizzazione rimane limitato da:

$$| e_i | \leq C h^p \frac{e^{L(x_i - x_0)} - 1}{L}$$

con C costante dell'errore di troncamento locale del metodo [2].

In questa relazione prenderemo in esame tre metodi One-step: il metodo di Eulero, il metodo di Verlet e il metodo di Runge-Kutta (classico).

2.1 Metodo di Eulero

Il primo metodo One-step che prendiamo in considerazione è quello di Eulero che è un algoritmo molto semplice e facile da implementare. L'idea di questo algoritmo è di sostituire alla derivata il rapporto incrementale o equivalentemente di usare la serie di Taylor e arrestarla al primo termine effettuando quindi un'approssimazione lineare della funzione soluzione.

Dato il problema di Cauchy:

$$\begin{cases} y'(x) = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

si sviluppa $y(x)$ con la formula di Taylor attorno al punto x_i

$$y(x) = y(x_i) + y'(x_i)(x - x_i) + R_i .$$

Calcolando in $x = x_{i+1} = x_i + h$ si ottiene:

$$y(x_{i+1}) = y(x_i) + h f(x_i, y(x_i)) + R_i .$$

Sappiamo dalla formula di Taylor che il resto è (nella forma di Lagrange):

$$R_i = \frac{1}{2} y''(\beta) h^2 \quad \beta \in (x_i, x_i + h)$$

cioè $R_i = O(h^2)$.

Notiamo quindi che l'Algoritmo di Eulero è di ordine 1 e la funzione ϕ del metodo è uguale ad f ; da R_i deduciamo che il metodo è anche consistente.

2.2 Metodo di Verlet

L'algoritmo di Verlet è un metodo che sfrutta lo sviluppo di Taylor per approssimare la soluzione di un'equazione differenziale. Quest'algoritmo è particolarmente adatto ad integrare le equazioni del moto. Infatti, se la funzione $y(t)$ rappresenta la posizione al tempo t , $y'(t)$ rappresenta la velocità e $y''(t)$ l'accelerazione, l'idea del metodo è quella di sviluppare Taylor in due istanti diversi, in avanti e indietro nel tempo:

$$y(t + \Delta t) = y(t) + \frac{dy(t)}{dt} \Delta t + \frac{1}{2} \frac{d^2y(t)}{dt^2} \Delta t^2 + \frac{1}{6} \frac{d^3y(t)}{dt^3} \Delta t^3 + O(\Delta t^4)$$

$$y(t - \Delta t) = y(t) - \frac{dy(t)}{dt} \Delta t + \frac{1}{2} \frac{d^2y(t)}{dt^2} \Delta t^2 - \frac{1}{6} \frac{d^3y(t)}{dt^3} \Delta t^3 + O(\Delta t^4)$$

Sommendo membro a membro otteniamo l'algoritmo di Verlet:

$$y(t + \Delta t) = 2y(t) - y(t - \Delta t) + \frac{1}{2} \frac{d^2y(t)}{dt^2} \Delta t^2 + O(\Delta t^4)$$

Notiamo che il metodo è del 3° ordine. I valori della velocità si ottengono invece dalla:

$$v\left(t + \frac{1}{2}\Delta t\right) \approx \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

Si può mostrare che una espressione equivalente dell'algoritmo standard di Verlet è data dal metodo Velocity Verlet [10]:

$$\begin{aligned} x_{i+1} &= x_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2 \\ v_{i+1} &= v_n + \frac{1}{2} (a_{i+1} + a_n) \Delta t \end{aligned}$$

dove:

- v_i è il valore della velocità nel tempo i ,
- a_i è il valore dell'accelerazione nel tempo i .

2.3 Metodi Runge-Kutta

I metodi Runge-Kutta (RK) sono una famiglia di algoritmi che permettono di avere un elevato ordine di accuratezza. L'idea di questi metodi è quella di aumentare il numero di valutazioni funzionali ad ogni passo per riuscire ad aumentare la precisione.
Come mostra [1], nella forma più generale i metodi RK possono essere scritti come:

$$y_{i+1} = y_i + h \phi(x_i, y_i; h; f) \quad i \geq 0$$

dove la funzione incremento ϕ è definita come :

$$\begin{aligned} \phi(x_i, y_i; h; f) &= \sum_{m=1}^s b_m K_m , \\ K_m &= f(x_i + c_m h, y_i + h \sum_{j=1}^s a_{mj} K_j) \quad m = 1, 2, \dots, s \end{aligned}$$

Determinando i valori c_m, a_{mj}, b_m attraverso considerazioni di tipo analitico e algebrico si crea un metodo RK [3].

Uno dei metodi più usati è quello del quarto ordine esplicito chiamato RK4 :

$$\begin{aligned} K_1 &= f(x_i, y_i) \\ K_2 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} K_1\right) \\ K_3 &= f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} K_2\right) \\ K_4 &= f(x_{i+1}, y_i + h K_3) \end{aligned}$$

$$y_{i+1} = y_i + \frac{h}{6} (K_1 + 2K_2 + 2K_3 + K_4).$$

È interessante studiare la complessità computazionale di questi algoritmi. Si può subito notare che nella valutazione della loro complessità va certamente tenuto in conto il numero di valutazioni della funzione f ad ogni passo, il grado di precisione che si vuole e il numero di punti nella fase di discretizzazione dell'intervallo di definizione. È naturalmente sottointeso che la funzione f deve essere una funzione calcolabile.

Considerando l'operazione di valutazione della funzione come la più costosa vediamo che l'algoritmo RK del quarto ordine presentato precedentemente deve fare 4 valutazioni funzionali pertanto è quello più lento a parità degli altri parametri.

Capitolo 3

Modelli fisico-matematici

3.1 Moto di un corpo in un campo gravitazionale

In meccanica Newtoniana, tra due corpi di massa M ed m vige la legge della gravitazione universale:

$$F = -G \frac{Mm}{d^2}$$

dove G è la costante di gravitazione universale e d è la distanza tra i (centri di massa dei) due corpi.

Ricordando che $\vec{F} = m\vec{a}$, assumendo che $M \gg m$ e prendendo un sistema di riferimento con origine in M , possiamo dunque scrivere:

$$\begin{cases} \frac{d^2x}{dt^2} = -GM \frac{x}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \\ \frac{d^2y}{dt^2} = -GM \frac{y}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \\ \frac{d^2z}{dt^2} = -GM \frac{z}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \end{cases}$$

in cui x, y, z sono le coordinate della posizione del corpo m al tempo t .

Affinché il problema sia ben posto, bisogna fornire le condizioni iniziali; inoltre, il sistema può essere riformulato, come visto nel primo capitolo, come un sistema equivalente coinvolgendo solo equazioni del primo ordine.

$$\left\{ \begin{array}{l} \frac{dx}{dt} = v_x \\ \frac{dy}{dt} = v_y \\ \frac{dz}{dt} = v_z \\ \frac{dv_x}{dt} = -GM \frac{x}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \\ \frac{dv_y}{dt} = -GM \frac{y}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \\ \frac{dv_z}{dt} = -GM \frac{z}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} \\ x(t_0) = x_0 \quad v_x(t_0) = v_{0x} \\ y(t_0) = y_0 \quad v_y(t_0) = v_{0y} \\ z(t_0) = z_0 \quad v_z(t_0) = v_{0z} \end{array} \right. \quad (1)$$

dove:

- (x_0, y_0, z_0) è il vettore che rappresenta la posizione iniziale del corpo m ,
- (v_{0x}, v_{0y}, v_{0z}) è il vettore con le componenti della velocità iniziale del corpo m ,
- x, y, z sono le coordinate della posizione del corpo m al tempo t ,
- v_x, v_y, v_z sono le componenti del vettore velocità del corpo m al tempo t .

Questo quindi è il sistema di equazioni differenziali che descrive il moto di un corpo di massa m soggetto alla gravità di una massa $M \gg m$, trascurando possibili attriti.

Nella nostra trattazione, la terza dimensione anche se è stata implementata nel codice verrà ignorata (per semplicità, ma senza perdita di generalità). Inoltre, come condizioni iniziali si avrà sempre che:

$$\begin{array}{ll} v_x(t_0) = 0 & x(t_0) = r \\ v_y(t_0) = \sqrt{\frac{GM}{r}} & y(t_0) = 0 \\ v_z(t_0) = 0 & z(t_0) = 0 \end{array}$$

Per una maggior facilità di lettura, scriveremo le equazioni in forma vettoriale:

$$m\ddot{\vec{r}} = -G \frac{Mm}{r^2} \hat{r}$$

Questo modello è ancora poco interessante numericamente, poiché anche senza ricorrere ad algoritmi numerici, si riesce a calcolarne la soluzione attraverso vie analitiche.

Per rendere il modello più complesso, e quindi necessitante l'utilizzo di tecniche numeriche per la sua risoluzione, si introduce un attrito, cioè un termine al membro destro dell'ultima equazione il cui effetto è di ridurre la velocità di m : in questo modo è possibile simulare il moto di un satellite in un'atmosfera. Poiché l'attrito dipende da una moltitudine di fattori, come l'area del satellite esposta, l'altezza dal suolo, la temperatura dell'atmosfera, ecc., è necessario modellizzare il sistema adeguatamente.

3.1.1 Satellite soggetto ad attrito

In [7] viene esposto un modello per l'attrito (viscoso) a cui è soggetto un satellite in moto nell'atmosfera terrestre:

$$D = -\frac{1}{2} \rho v^2 A c_d \hat{v}$$

in cui:

- D è la forza frenante,
- A è l'area esposta all'attrito,
- v è la velocità,
- c_d è il coefficiente di resistenza (in genere posto pari a 2),
- $\rho = 6 \times 10^{-10} e^{-\frac{(h-175)m}{T}}$ è la densità dell'aria, con h altezza dal suolo, $m = 27 - 0.012(h - 200)$ la massa molecolare dell'aria in Kg/m^3 e $T = 900 + 2.5(F10.7 - 70) + 1.5A_p$ la sua temperatura, dove A_p è l'indice geomagnetico e $F10.7$ il flusso radio solare a 10.7 cm; il primo ha valori compresi tra 0 e 400, mentre, il secondo varia da 65 a 300 SFUs [Solar Flux Units; 1 SFU = 10^{-22} Watts/ $\text{m}^2 \text{ Hz}$] secondo l'attività solare.

Pertanto, tenendo conto dell'attrito, otteniamo il modello che descrive il moto di un satellite in orbita attorno Terra :

$$m\vec{r} = -G \frac{Mm}{r^2} \hat{r} - \frac{1}{2} \rho v^2 A c_d \hat{v} \quad (2)$$

Infine, introduciamo un ulteriore elemento nella nostra modellazione, ovvero aggiungiamo un termine per descrivere l'attivazione di un motore ad un certo istante di tempo t_{acc} per una certa durata t_{durata} . Questo termine può imprime al corpo un accelerazione/decelerazione nella direzione del moto:

$$m\vec{r} = -G \frac{Mm}{r^2} \hat{r} - \frac{1}{2} \rho v^2 A c_d \hat{v} + a_m \theta(t - t_{acc}) \theta(t_{acc} + t_{durata} - t) \hat{v} \quad (3)$$

in cui $a_m > 0$ è la forza impressa dal motore e θ è la funzione di Heaviside definita come:

$$\theta(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases}$$

La funzione di Heaviside permette di controllare l'accensione e lo spegnimento della forza motore.

Notiamo quindi che l'Eq. (3) è un'equazione differenziale vettoriale non lineare del secondo ordine e certamente sarebbe estremamente complessa da studiare e risolvere senza l'ausilio di una simulazione numerica. Naturalmente, poiché descrive una situazione fisica a noi ben nota, manteniamo un intuito fisico dell'evoluzione che essa prevede.

3.2 Pendolo composto

Per il secondo modello fisico che discuteremo, non sarà possibile a priori immaginare o prevedere l'andamento e l'evoluzione del sistema, né tantomeno risolverlo per via analitica. Si tratta del pendolo composto, ovvero di un pendolo reale dotato di una propria forma e geometria, sottoposto ad una forzante esterna. L'equazione differenziale che descrive il moto di tale sistema è la seguente equazione differenziale del secondo ordine [8]:

$$\theta''(t) + \frac{b}{I} \theta'(t) + \frac{mgl}{I} \operatorname{sen}(\theta) = F \cos(wt)$$

in cui:

- θ è la posizione angolare del pendolo (con il punto di equilibrio statico posto in $\theta = 0$)
- m è la massa del pendolo,
- g l'accelerazione di gravità,
- b lo smorzamento del pendolo,
- I il momento inerzia del pendolo,
- l la lunghezza del pendolo,
- F l'ampiezza della forzante esterna impressa al pendolo,
- w la frequenza della forzante.

Notiamo subito che l'equazione è non lineare. Questo è un esempio chiaro di un sistema che per via analitica non si può risolvere e per il quale è assolutamente necessario l'uso di algoritmi numerici.

L'equazione è trasformabile in un sistema equivalente con 2 equazioni del primo ordine:

$$\begin{cases} \theta'(t) = \omega(t) \\ \omega'(t) = -\frac{b}{I}\omega(t) - \frac{mgl}{I} \operatorname{sen}(\theta) + F \cos(wt) \end{cases} \quad (4)$$

Capitolo 4

Codice e risultati computazionali

Per implementare e confrontare i vari algoritmi abbiamo costruito un integratore con il linguaggio Python che tramite la scelta dei parametri può simulare uno dei sistemi descritti nel precedente capitolo. Abbiamo utilizzato la piattaforma BitBucket per la condivisione e modifica del codice sorgente. Essa si poggia su git, uno strumento di Version Control System (VCS). I VCS sono dei software che aiutano gli sviluppatori e i team a gestire le modifiche del codice, conservando la storia di tutte le modifiche apportate al codice stesso. Se viene commesso un errore, gli sviluppatori possono tornare indietro e confrontare versioni precedenti del codice per correggere e migliorare il software.

Il codice è situato su una repository pubblica ed è scaricabile dal seguente link:

URL : https://bitbucket.org/senad_96/tirocinio/downloads/

Il programma è stato suddiviso in vari file per una maggiore leggibilità e organizzazione. In due file diversi, `params_satellite.py` e `params_pendolo.py`, sono situati i parametri dei sistemi e degli algoritmi. Ad esempio, tramite la scelta dei parametri in `param_pendolo.py` si può decidere se far apparire o meno l'attrito nel sistema e se attivare la forza motrice. Nel file `evolve.py` si può decidere quale sistema dinamico far evolvere: più precisamente, il valore della variabile `sistema` può essere settato in `satellite` o `pendolo`. Inoltre, decidendo quale input dare alla funzione `simulazione_(metodo1, metodo2, metodo3, metodo4)` si seleziona l'algoritmo da usare per la simulazione.

Al `metodo1` corrisponde Eulero, al `metodo2` Verlet, al `metodo3` RK4 ed infine il `metodo4` indica l'utilizzo della funzione `odeint` della libreria `Scipy`.

Vediamo qui di seguito alcuni parti del codice, in particolare mostriamo i file più importanti dell'integratore e l'implementazione di un algoritmo: `evolve.py`, `params_satellite.py`, `params_pendolo.py` e `algoritmo_Eulero.py`.

evolve.py

```
import time
from shutil import copyfile

#sciegli il sistema 'satellite' o 'pendolo'
sistema = 'satellite'

#Simulazioni

metodo1 = 'Eulero'
metodo2 = 'Verlet'
metodo3 = 'RK4'
metodo4 = 'odeint'

copyfile('params_' + sistema + '.py', 'params.py')

copyfile('modello_' + sistema + '.py', 'modello_fisico.py')

copyfile('tools_' + sistema + '.py', 'tools.py')

from execution import *

def simulazione_(metodo):
    simulazione_(metodo4)
    return

start_time = time.time()

simulazione_(metodo3) #sciegli che algoritmo usare in input

print("--- %s seconds ---" % (time.time() - start_time))
```

params_satellite.py

```
# Parametri fisici del sistema
Ndims = 3 # Numero delle dimensioni del problema
M1 = 5.972e24 # Kg
m2 = 1 # Kg
raggio_terrestre = 6371000 #6371000 #metri
altezza = 250 # in km nell'articolo ma qui in metri
distanza = raggio_terrestre + altezza
area = 0
#area = 25 # m^2 #si puo scielgire se mettere None

#parametri del motore/atterraggio
acc_motore = 0 #forza può essere attivata
t_acc = 5000 #secondi
durata = 100 #secondi

# Parametri di integrazione
# h_inter: passo di integrazione
# x_intervallo: intervallo di valutazione della funzione
h_inter = 0.01
x_intervallo = 12000
```

params_pendolo.py

```
# Parametri fisici del sistema
Ndims = 1 # Numero delle dimensioni del problema
m = 2 # Kg
l = 1 # m
I = 5 # kg*m^2
beta = 1 # Kg
Famp = 9# N
omega = 2 # rad/s
g = 9.81
raggio_terrestre = -1 # raggio fittizio per eludere
l'algoritmo nel break

#parametti inziali theta0 e omega0
theta0 = 0
omega0 = 0

# Parametri di integrazione
# h_inter: passo di integrazione
# x_intervallo: intervallo di valutazione della funzione
h_inter = 0.01
x_intervallo = 70000

#parametri per la biforcazione
biforcazione = False
famp_max = 10
passo = 0.001
```

algoritmo_Eulero.py

```
import numpy as np
import json
from scipy.constants import G
import modello_fisico as model
from params import *

def passo_eulero_singolo(x, dxdt):
    x_new = x + h_inter*dxdt
    return x_new

def evolve():
    with open("valori_eulero.txt" , mode='w' ) as f:
        var_lagrangiane = model.condizioni_iniziali()
        raggio = np.linalg.norm(var_lagrangiane[:Ndims])
        var_lagrangiane = np.append(var_lagrangiane,raggio)#aggiungo raggio
        json.dump(var_lagrangiane.tolist(), f)
        f.write('\n')

        for i in range(1,x_intervallo):
            # Posizione attuale
            pos = var_lagrangiane[0:Ndims]
            # Velocita' attuale
            vel = var_lagrangiane[Ndims:2*Ndims]

            # Accelerazione attuale
            acc = model.calcola_accelerazioni(var_lagrangiane[:2*Ndims],
h_inter*(i-1))

            # Aggiorna posizione e velocita'
            var_lagrangiane = \
            passo_eulero_singolo( np.append(pos, vel), \
            np.append(vel, acc))

            pos = var_lagrangiane[:Ndims]
            raggio = np.linalg.norm(pos)
            var_lagrangiane = np.append(var_lagrangiane, raggio)

            if np.linalg.norm(pos[0:Ndims]) - raggio_terrestre < 0 :
                break

            json.dump(var_lagrangiane.tolist(), f)
            f.write('\n')

    return
```

Eseguendo a questo punto il file `evolve.py` con i giusti parametri si avvierà la simulazione del sistema desiderato, secondo l'algoritmo selezionato.

Per trattare il sistema “satellite”, è stato usato il modello definito nell’Eq.(3) descritta nel precedente capitolo.

Per tutte le simulazioni vengono generati dei file `.txt` che portano il nome del rispettivo metodo utilizzato.

Da qui in seguito i termini *h_inter*, *passo* e *h* rappresenteranno lo stesso concetto pertanto verranno usati come sinonimi.

Per quanto riguarda la struttura del codice, sono stati creati i seguenti file Python con il relativo numero di righe di codice (contando anche gli spazi tra una riga e l’altra):

- `algoritmo_Eulero.py`: 57 righe
- `algoritmo_Verlet.py`: 68 righe
- `algoritmo_RK4.py`: 80 righe
- `evolve.py`: 35 righe
- `execution.py`: 30 righe
- `modello_pendolo.py`: 55 righe
- `modello_satellite.py`: 110 righe
- `params_pendolo`: 26 righe
- `params_satellite`: 22 righe
- `plot_biforcazione_chaos.py`: 43 righe
- `scipy_ode_integrators.py`: 88 righe
- `tools_pendolo.py`: 45 righe
- `tools_satellite.py`: 90 righe
- `verifica_dati.py`: 40 righe
- `file_famp.py`: 2 righe

Il numero totale di righe di codice togliendo quindi le eventuali righe inutilizzate si attesta intorno alle 300. Nel codice sono state utilizzate diverse librerie, che ci hanno permesso di semplificare alcune operazioni;

- Numpy: esclusivamente per l’elaborazione dei vettori,
- Matplotlib: per la stampa dei diversi grafici,
- Scipy: solo per l’uso della funzione `odeint` e per l’accesso alle costanti fisiche,
- Json: per il formato dei dati generati dagli algoritmi.

Di seguito discuteremo qualche esempio di simulazioni generate dal nostro integratore.

4.1 Simulazioni di moto nel campo gravitazionale terrestre

4.1.1 Test di caduta libera

Il primo esempio banale che mostriamo è la simulazione della caduta di un oggetto sulla Terra da una altezza prefissata trascurando gli attriti.

Con la scelta dei seguenti parametri in `params_satellite.py`:

$$m1 = 5.972 * 10^{24} \text{ Kg} \quad \text{massa della Terra}$$

$$m2 = 1 \text{ Kg} \quad \text{massa del oggetto}$$

$$\text{raggio_terrestre} = 6371000 \text{ m}$$

$$\text{altezza} = 250 \text{ m}$$

$$\text{area} = 0 \text{ m}^2$$

$$\text{acc}_{\text{motore}} = 0 \text{ N}$$

$$h_{\text{inter}} = 0.01 \text{ s}$$

$$x_{\text{intervallo}} = 12000 \text{ s}$$

Per un'altezza molto minore del raggio terrestre, è possibile calcolare il tempo di caduta di un oggetto in assenza di attriti mediante la formula:

$$t = + \sqrt{\frac{2 \text{ altezza}}{g}}$$

Si ha che il tempo di caduta per i parametri elencati è di 7.14 secondi (considerando $g = 9.81$).

Vediamo ora i risultati ottenuti utilizzando gli algoritmi di Eulero, di Verlet e RK4.

Figura 1: Test di caduta libera con il metodo di Eulero.

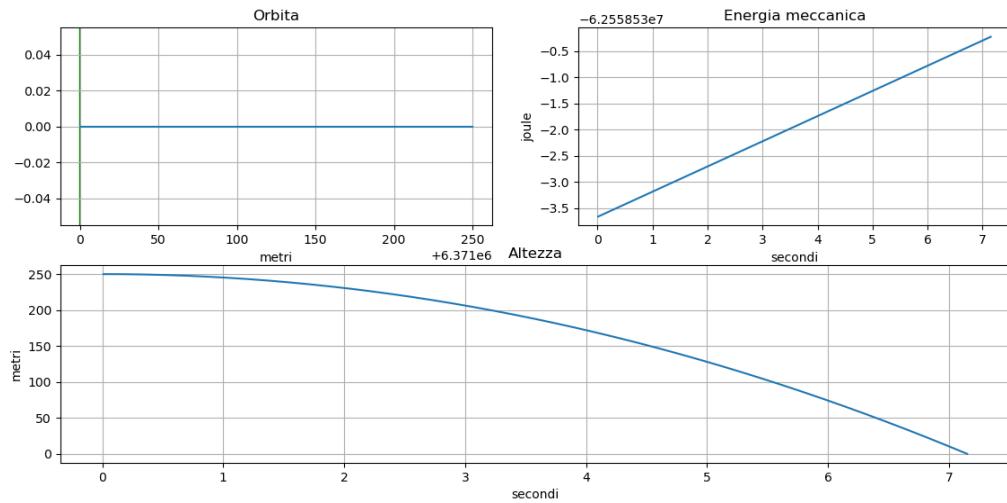


Figura 2: Test di caduta libera con il metodo di Verlet.

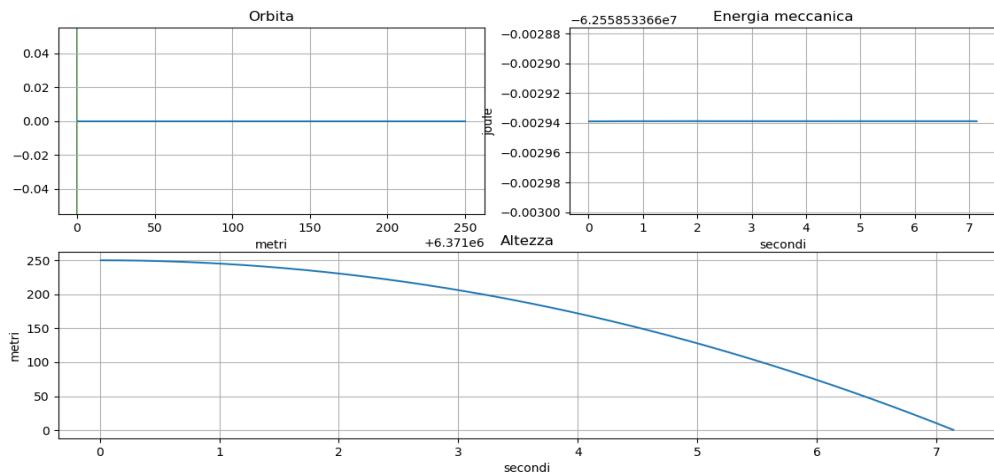
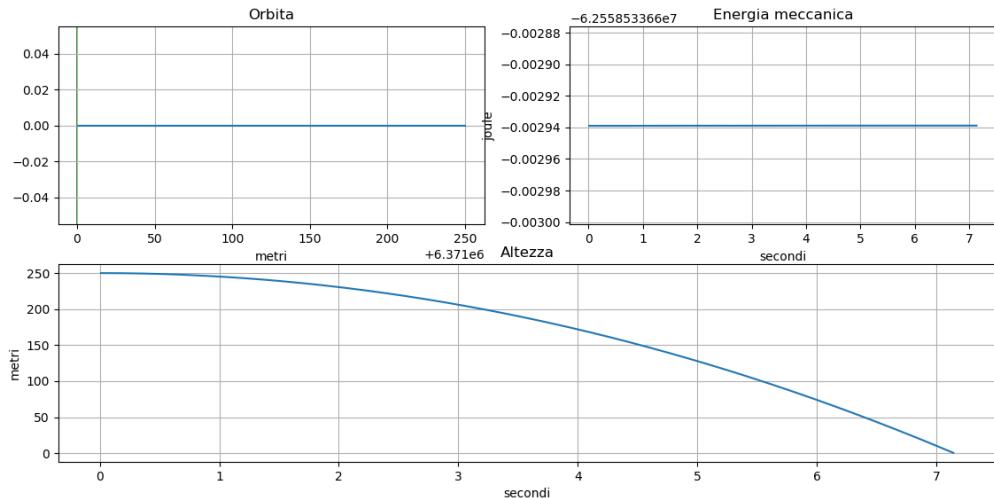


Figura 3: Test di caduta libera con RK4.



Possiamo già notare come ci siano delle differenze tra i risultati ottenuti con RK4 ed Eulero.

Figura 4: Ingrandimento dell'ultimo pannello della Figura 1.

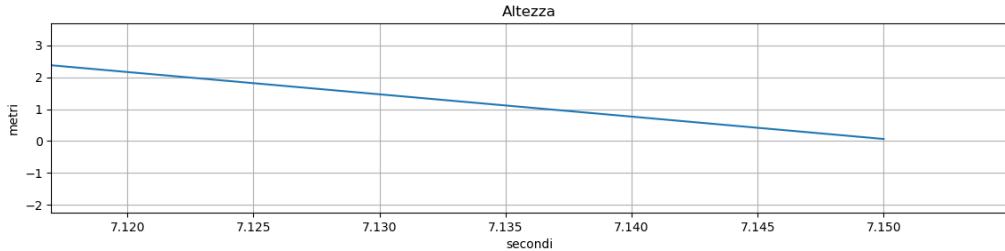


Figura 5: Ingrandimento dell'ultimo pannello della Figura 3.



Questo è ancora un esempio poco interessante, ma cambiando gli stessi parametri possiamo simulare l'orbita di un pianeta intorno ad una stella.

4.1.2 Orbite planetarie

Scegliamo i seguenti parametri in `params.satellite.py`:

$$m1 = 1.989 * 10^{30} \text{ Kg } \text{massa del Sole}$$

$$m2 = 5.972 * 10^{24} \text{ Kg } \text{massa della Terra}$$

$$\text{raggio_terrestre} = 0 \text{ m}$$

$$\text{altezza} = 152 * 10^9 \text{ m } \text{distanza Terra - Sole}$$

$$\text{area} = \text{None}$$

$$\text{acc_motore} = 0 \text{ N}$$

$$h_{\text{inter}} = 400 \text{ s}$$

$$x_{\text{intervallo}} = 500000 \text{ s}$$

Mostreremo i dati in maniera tale che sarà possibile confrontare vari grafici, tra cui uno che riporta l'evoluzione dell'errore del metodo. Il moto dei pianeti può essere considerato in buona approssimazione come un moto senza attrito e considerando la traiettoria del pianeta una circonferenza si può avere una stima dell'errore sul raggio effettuato dai metodi. Lo stesso discorso è stato applicato all'energia meccanica che sappiamo doversi conservare fisicamente, in assenza di attrito.

Figura 6: Simulazione dell'orbita terrestre con il metodo di Eulero.

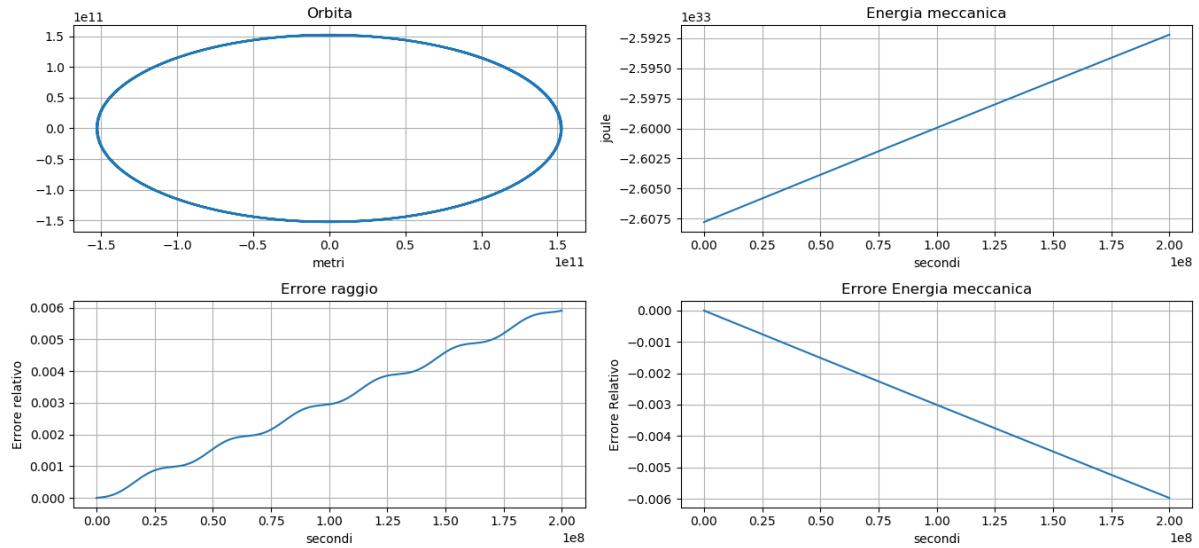


Figura 7: Versione della Figura 6 con un ingrandimento dell'orbita nel pannello in alto a sinistra.

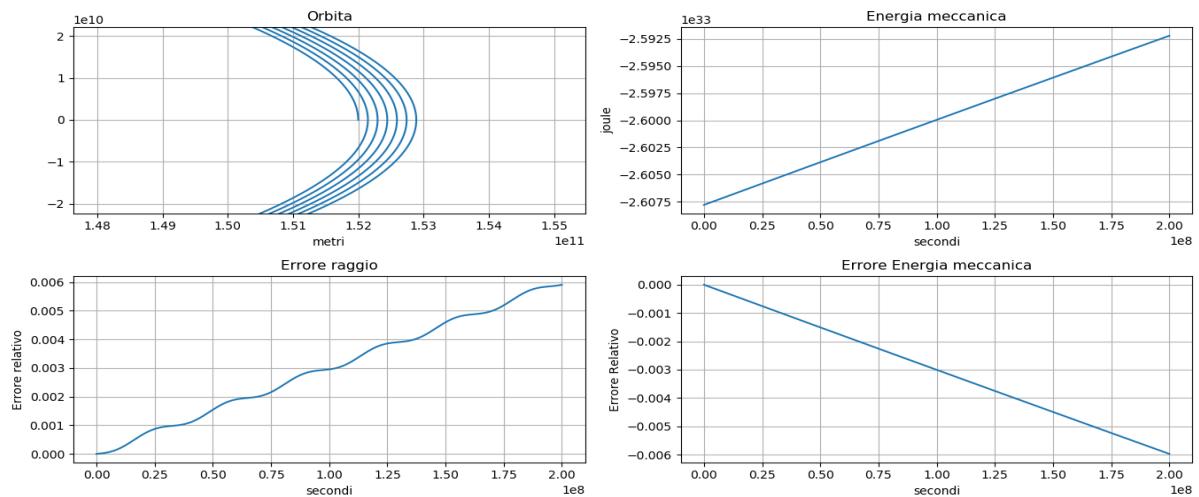


Figura 8: Simulazione dell'orbita terrestre con RK4

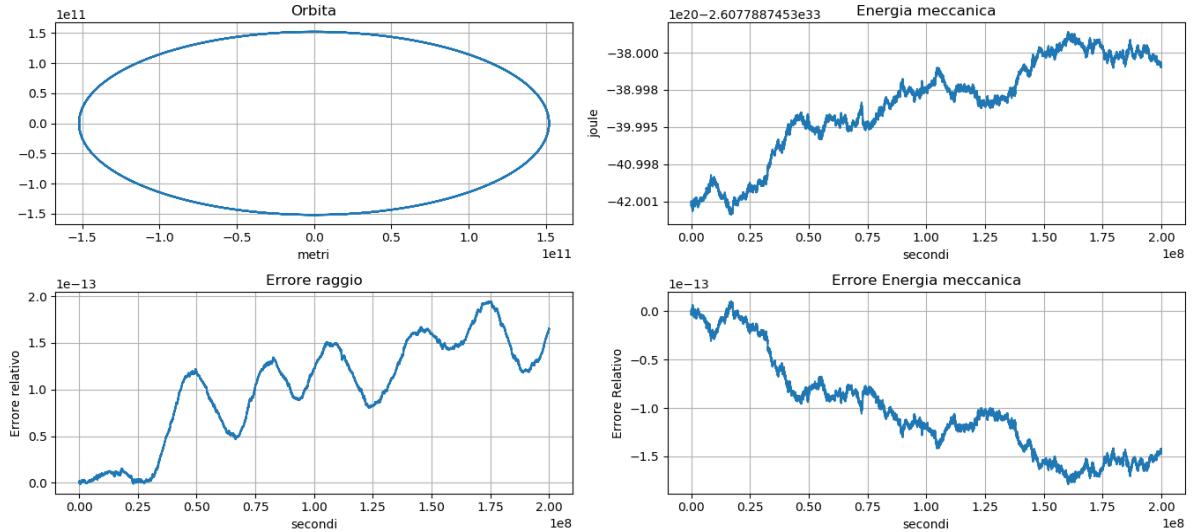
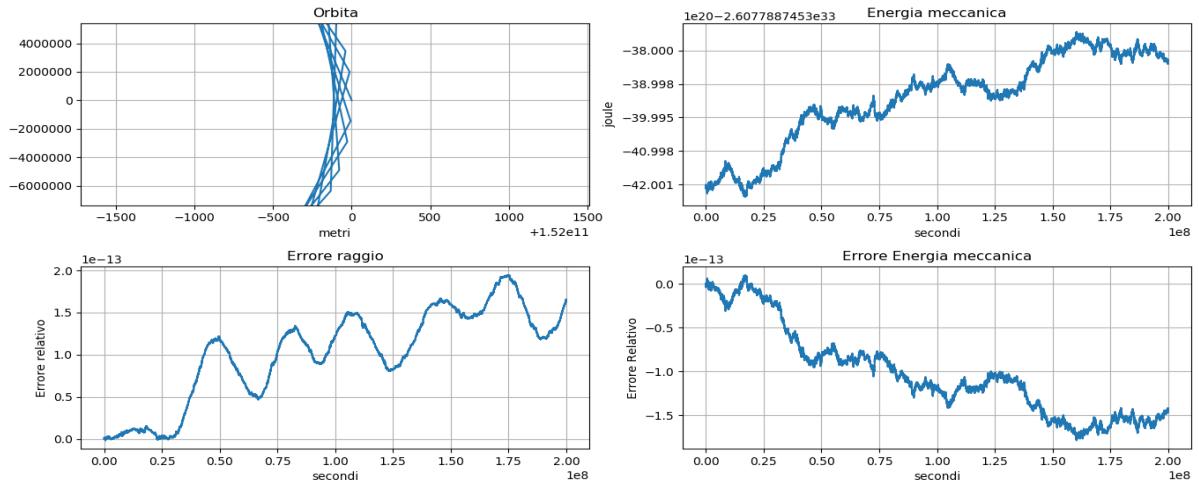


Figura 9: Versione della Figura 8 con un ingrandimento dell'orbita nel pannello in alto a sinistra.



È evidente la differenza tra i due metodi numerici. Vediamo che con il metodo di Eulero l'orbita del pianeta diventa sempre più larga con un errore sul raggio molto elevato. Questo è dovuto all'ordine di convergenza basso (1) e alla costruzione del metodo che, come spiegato, genera il punto successivo della funzione tramite l'uso locale del rapporto incrementale.

Per fare delle considerazioni riguardo la precisione dei metodi scegliamo un passo h adeguato e verifichiamo che valori restituiscono i metodi.

Cambiamo solo i parametri h_{inter} e $x_{intervallo}$ rispetto all'esempio precedente:

$$h_{inter} = 10000 \text{ s}$$

$$x_{intervallo} = 200000 \text{ s}$$

ed eseguiamo la simulazione sia con RK4 che con Verlet. Riportiamo i valori delle coordinate x e y in istanti generici per entrambe le simulazioni.

RK4

t	x	y
50k s (5p)	151992817996.5694	1477591777.767878
100k s (10p)	151971272664.97726	2955043923.0460596
500k s (50p)	151282359414.8544	1475288865.39834
100M s (10k p)	126070170200.37447	84913557136.9347
500M s (50k p)	-149602246658.37207	26891779315.34405
1.8G s (180k p)	-48696462840.653946	-143988383235.54214

Verlet

t	x	y
50k s (5p)	151992818005.1665	1477592708.704806
100k s (10p)	151971272704.7985	2955045785.0783267
500k s (50p)	151282360521.77188	14752898197.97201
100M s (10k p)	126078379129.18143	84901518364.59769
500M s (50k p)	-149589273354.85043	26968976404.01084
1.8G s (180k p)	-48999937521.58502	-143886201043.1922

Eseguiamo ora l'algoritmo di Verlet che per la posizione ha un errore del 3° ordine, cambiando però i parametri h_{inter} e $x_{intervallo}$ (rispetto al caso precedente dimezziamo h_{inter} e raddoppiamo $x_{intervallo}$) :

$$h_{inter} = 5000 \text{ s}$$

$$x_{intervallo} = 400000 \text{ s}$$

Verlet

t	x	y
50k s (10p)	151992817999.05814	1477592010.5012856
100k s (20p)	151971272675.61157	2955044388.54918
500k s (100p)	151282359694.99454	14752891198.386147
100M s (20k p)	126072210445.70808	84910563153.0688
500M s (100k p)	-149599087901.58075	26910591597.531765
1.8G s (360k p)	-48766270972.05583	-143964931564.9776

Notiamo che con il passo $h_{inter} = 10000$, gli algoritmi di Verlet ed RK4 restituiscono risultati diversi. Tuttavia, dimezzando il passo h_{inter} , i valori prodotti con il metodo di Verlet possiedono un numero maggiore di cifre identiche a quelle dei valori ottenuti

con RK4. Pertanto, avendo due o più metodi di ordine diverso e modificando di volta in volta il passo degli algoritmi è possibile fare considerazioni riguardo alla precisione e al numero di cifre corrette nei dati prodotti senza ricorrere ad altri strumenti matematici di analisi.

In generale, una diminuzione del passo h_{inter} pur permettendo di guadagnare precisione comporta un aumento del tempo di simulazione e anche un maggior numero di errori aritmetici dovuti alle operazioni.

Riportiamo ora il grafico che mostra la differenza in valore assoluto tra i vari risultati prodotti dai due metodi (RK4 e Verlet), per varie scelte di h_{inter} . Iniziamo con $h_{inter} = 10000$ per entrambi i metodi di integrazione.

Figura 10: Differenze in valore assoluto nella coordinata x tra il metodo di Verlet e RK4 con passo h=10K.

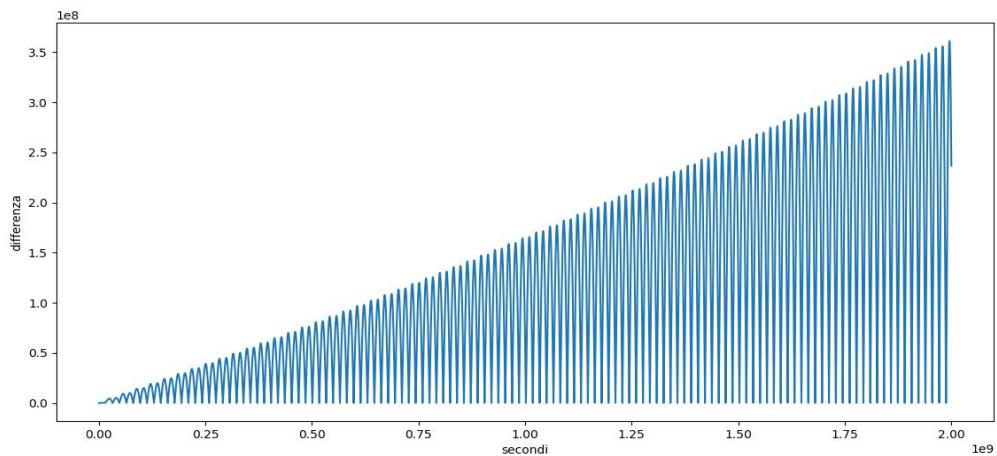
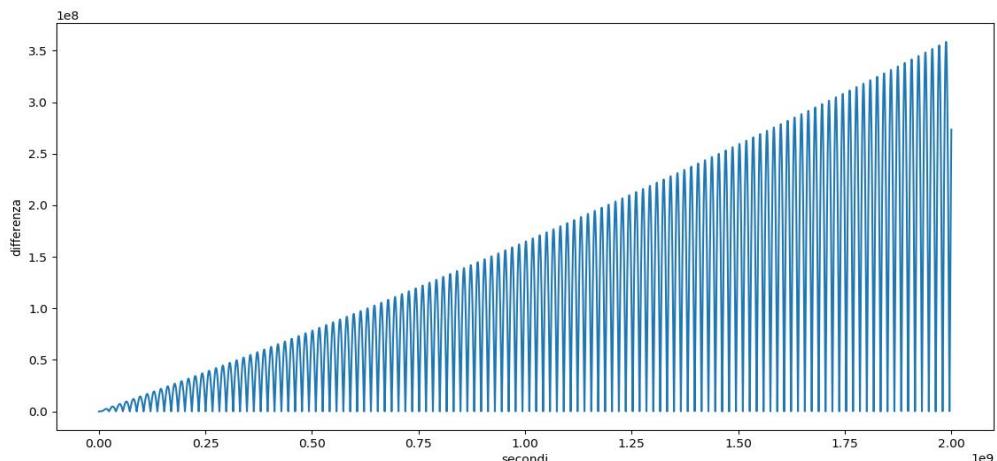
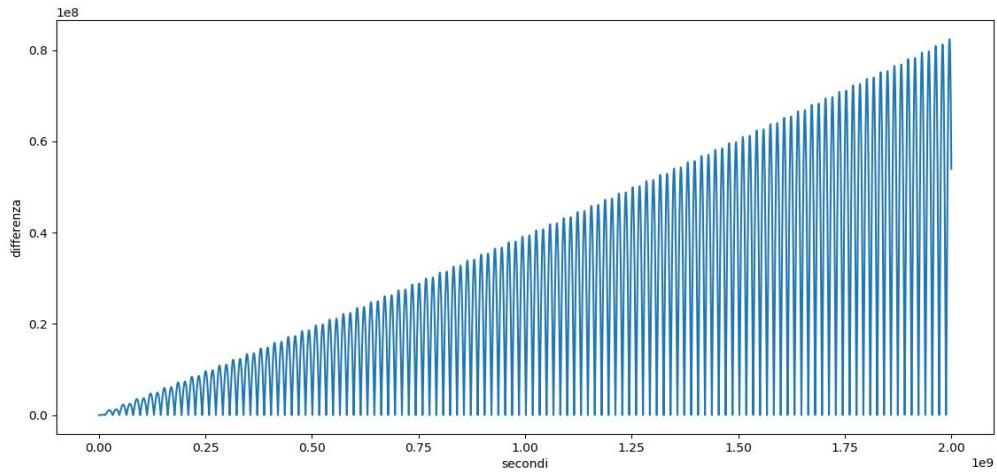


Figura 11: Differenze in valore assoluto nella coordinata y tra il metodo di Verlet e RK4 con passo h=10K.



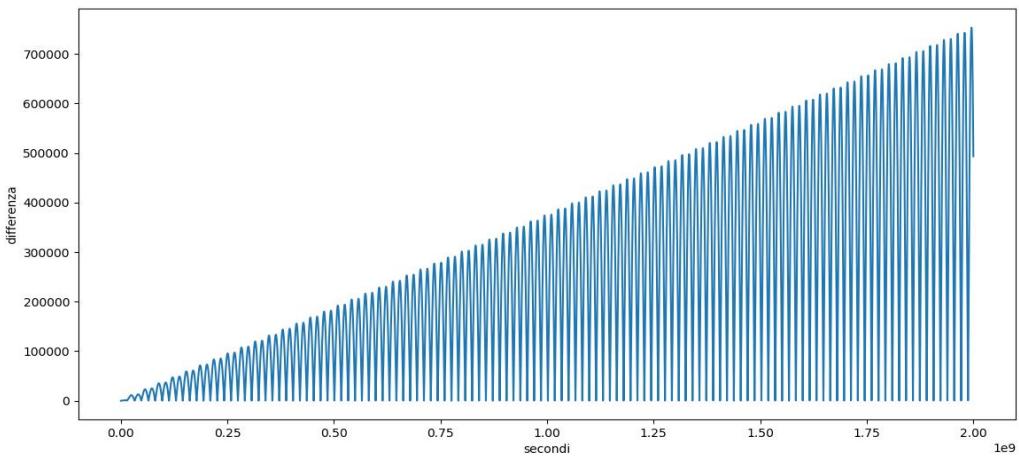
Se invece consideriamo RK4 con passo $h_{inter} = 10000$ e Verlet con passo $h_{inter} = 5000$, troviamo che la differenza nei valori della coordinata x si riduce di un fattore ~ 4 . Lo stesso avviene per la coordinata y .

Figura 12: Differenze in valore assoluto nella coordinata X tra il metodo di RK4 con $h=10K$ e Verlet con $h=5K$.



Riducendo ulteriormente di un fattore 10 il passo utilizzato nell'integrazione secondo Verlet, le differenze si riducono di un ordine di grandezza.

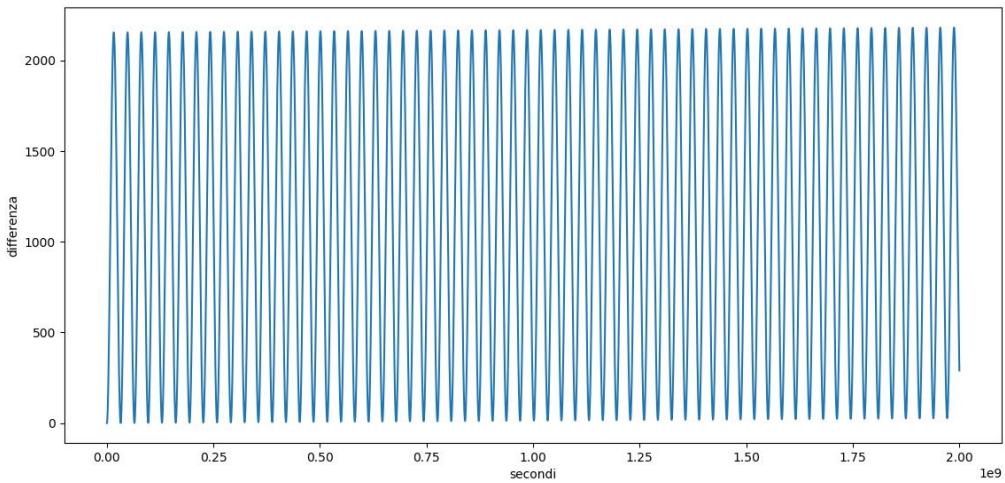
Figura 13: Differenze in valore assoluto nella coordinata X tra il metodo di RK4 con $h=10K$ e Verlet con $h=500$.



Da questi grafici notiamo che anche con un passo elevato per tempi relativamente brevi la differenza tra i due metodi è piccola.

L'ultimo grafico mostra come la differenza tra i due metodi sia drasticamente ridotta grazie alla diminuzione del passo di Verlet ed anche per lunghe evoluzioni l'errore sia trascurabile. Confrontiamo ora i valori del raggio ottenuti con i due metodi con $h_{inter} = 10000$ per RK4 e $h_{inter} = 500$ per Verlet.

Figura 14: Differenza tra il raggio dell'orbita restituito da Rk4 con $h=10K$ e Verlet con $h=500$



Si può notare che si hanno risultati diversi nel raggio in media di $1000\text{-}1200\text{ m}$. Contando il numero delle variazioni notiamo che sono 63; questo risultato è coerente con il fatto che $2 * 10^9$ secondi sono circa 63 anni.

4.1.3 Satellite in un'atmosfera

Passiamo ora alla simulazione dell'orbita di un satellite in discussione nel capitolo precedente, è un altro sistema dinamico non lineare. Riscriviamo il modello matematico:

$$m\vec{r}' = -G \frac{m_1 m_2}{r^2} \hat{r} - \frac{1}{2} \rho v^2 A c_d \hat{v} + a_m \theta(t - t_{acc}) \theta(t_{acc} + t_{durata} - t) \hat{v}$$

Sceglieremo come parametri in `params_satellite.py` i seguenti valori :

$$m1 = 5.972 * 10^{24} \text{ Kg} \quad \text{massa della Terra}$$

$$m2 = 1200 \text{ Kg} \quad \text{massa del satellite}$$

$$\text{raggio_terrestre} = 6371000 \text{ m}$$

$$\text{altezza} = 600000 \text{ m} \quad \text{altezza di partenza}$$

$$\text{area} = 25 \text{ m}^2 \quad (\text{A})$$

$$\text{acc_motore} = 0 \text{ N} (a_m)$$

$$h_{inter} = 1 \text{ s}$$

$$x_{intervallo} = 200000 \text{ s}$$

$$F10.7 = 80 \text{ SFUs}$$

$$A_p = 50$$

In altre parole, simuliamo l'orbita di un satellite di massa 1200 Kg ad un'altezza di 600 Km, con area esposta all'attrito di 25 m^2 .

Possiamo immaginare che per questo sistema le differenze tra i risultati prodotti dagli algoritmi saranno più sensibili a variazioni dei parametri di integrazione.

Da qui in poi abbiamo anche usato come ulteriore confronto il metodo *odeint* della libreria *Scipy* che implementa un ulteriore metodo di risoluzione. Alla funzione *odeint* è possibile dare in input diversi parametri di integrazione, tra questi è presente il campo *hmax* che rappresenta il passo massimo che può usare l'algoritmo adattivo di integrazione implementato da *odeint*.

Figura 15: Simulazione del moto di un satellite con RK4.

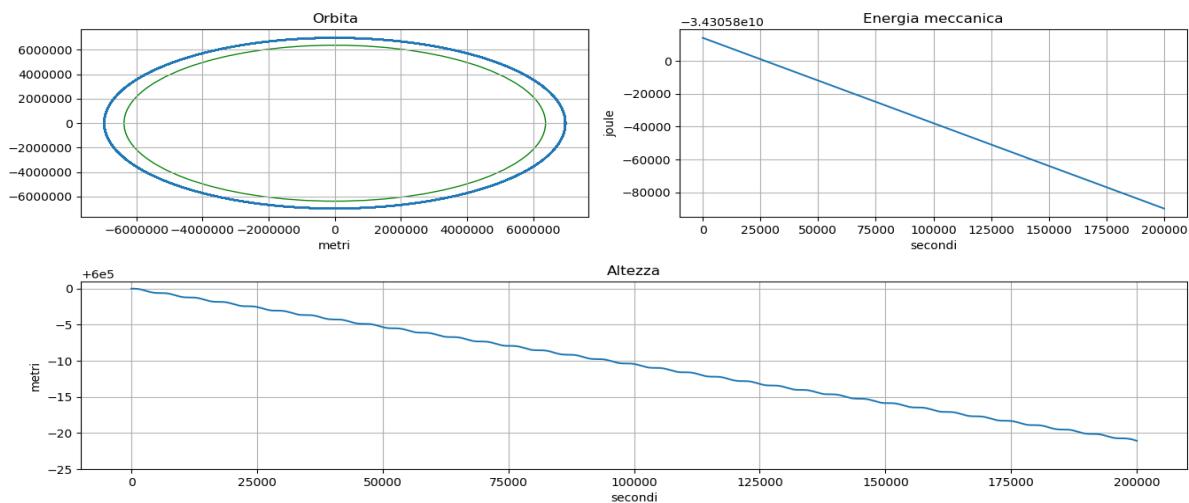


Figura 16: simulazione del moto di un satellite con *odeint*.

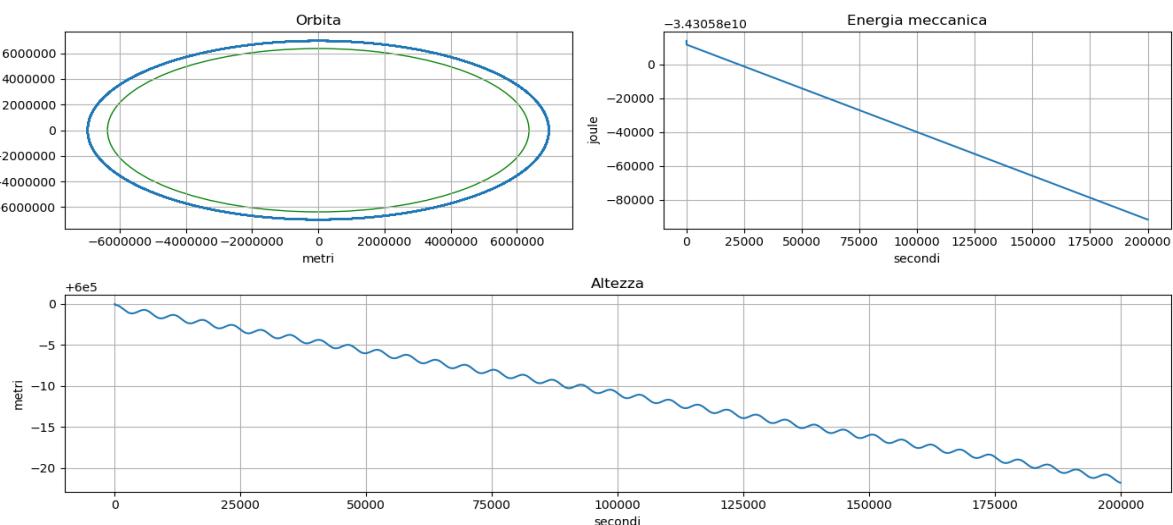


Figura 17: Simulazione del moto del satellite con il metodo di Verlet.

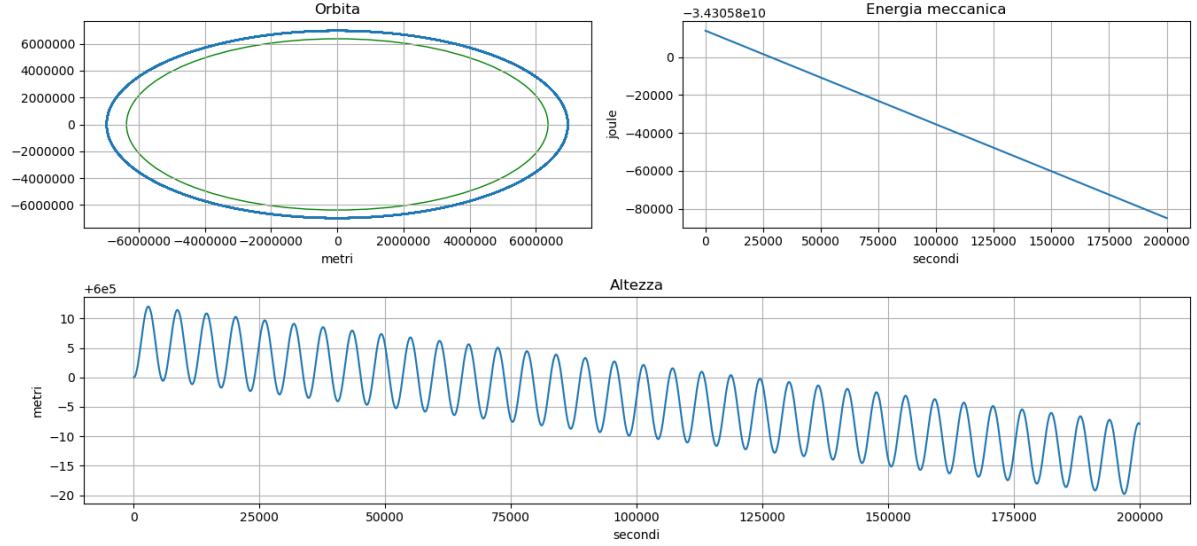
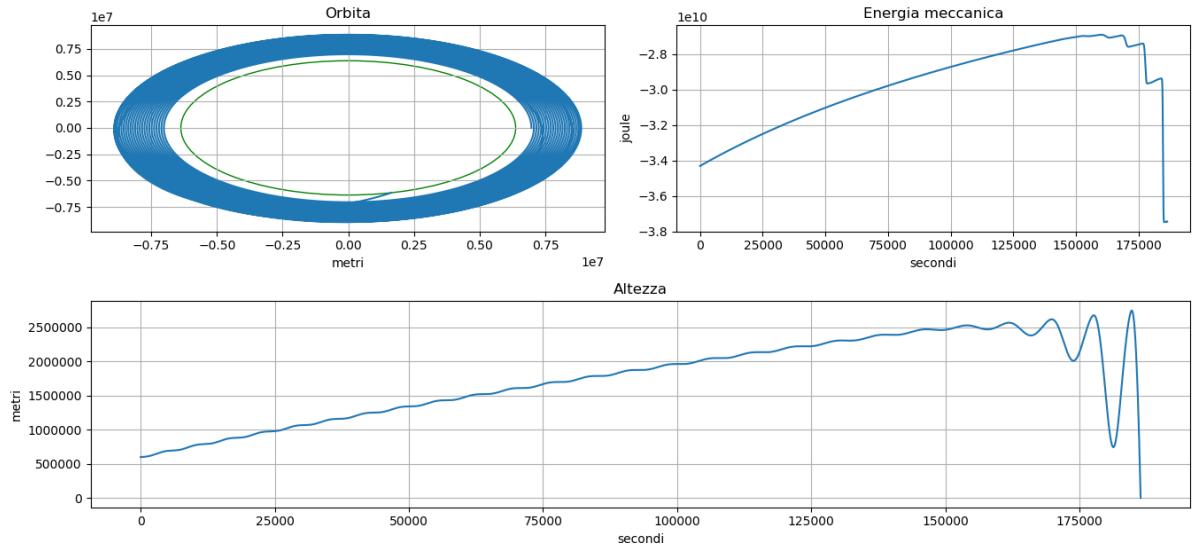


Figura 18: Simulazione del moto di un satellite con il metodo di Eulero.

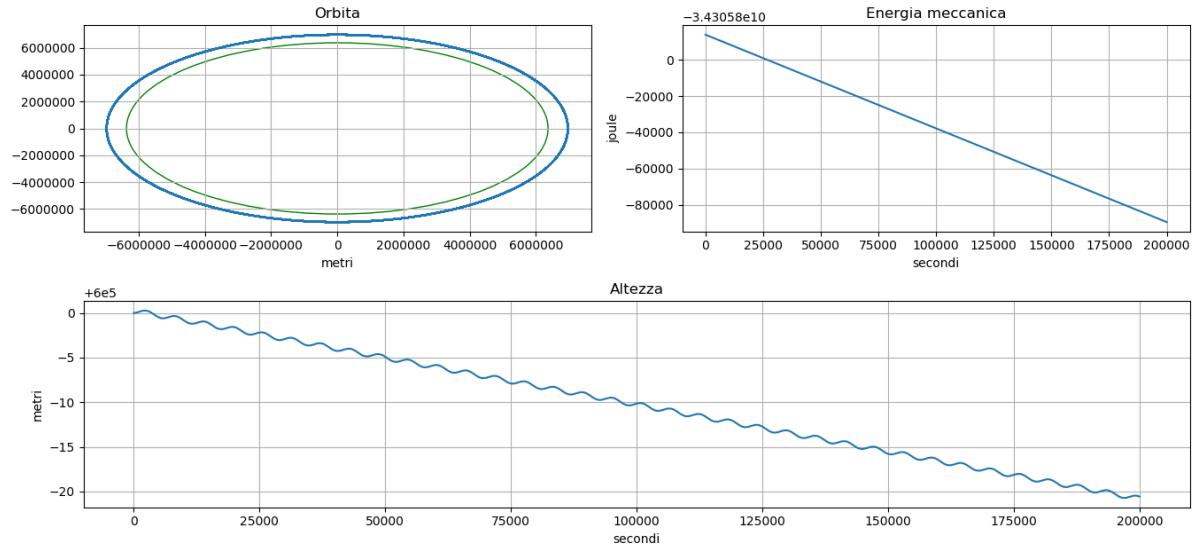


I risultati di `odeint` e `RK4` sono equivalenti, mentre si nota un totale errore nei dati del metodo di Eulero: nella simulazione l'altezza del satellite tende addirittura ad aumentare. Tuttavia, anche con valori di h_{inter} molto piccoli il metodo di Eulero non è adatto e commette un errore non accettabile. Con questo passo, l'algoritmo di Verlet restituisce dei dati che oscillano e che non sono attendibili. Diminuiamo pertanto il passo a 0.2, cioè 5 volte più piccolo dei precedenti, ed eseguiamo 1 milione di iterazioni (5 volte di più), ovvero assegniamo i valori:

$$h_{inter} = 0.2 \text{ s}$$

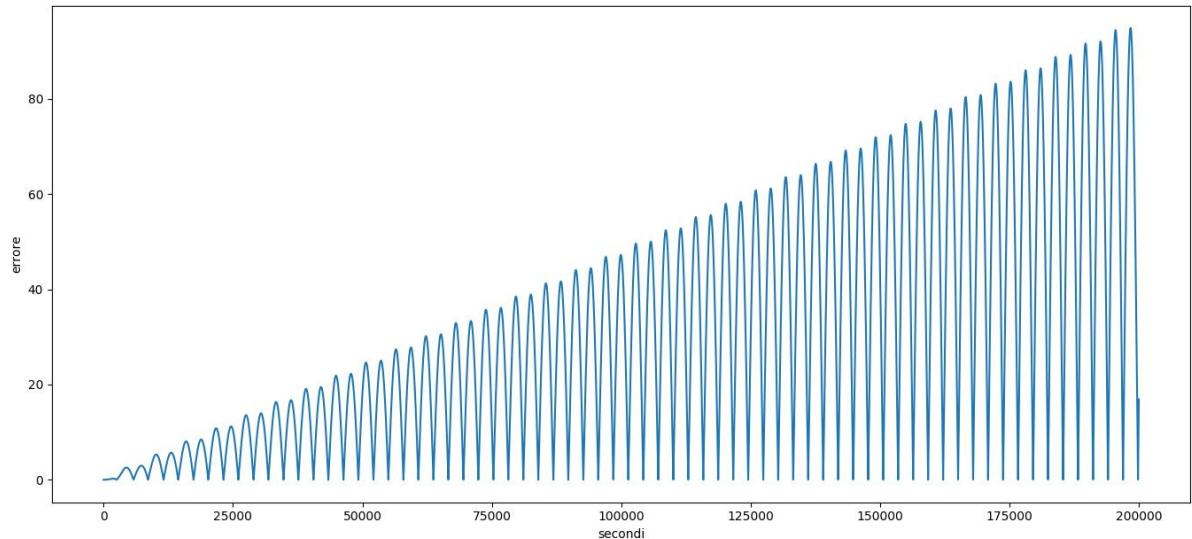
$$x_intervallo = 1000000 \text{ s}$$

Figura 19: Simulazione del moto di un satellite con il metodo di Verlet con $h=0.2$.



Possiamo verificare come in questo modo il metodo di Verlet raggiunga la precisione degli algoritmi del quarto ordine, vedendo la differenza in valore assoluto nella coordinata x tra i risultati prodotti con RK4 (con passo 1) e con Verlet (con passo 0.2).

Figura 20: Differenza in valore assoluto dei valori della coordinata x tra RK4 con $h=1$ e Verlet con $h=0.2$.



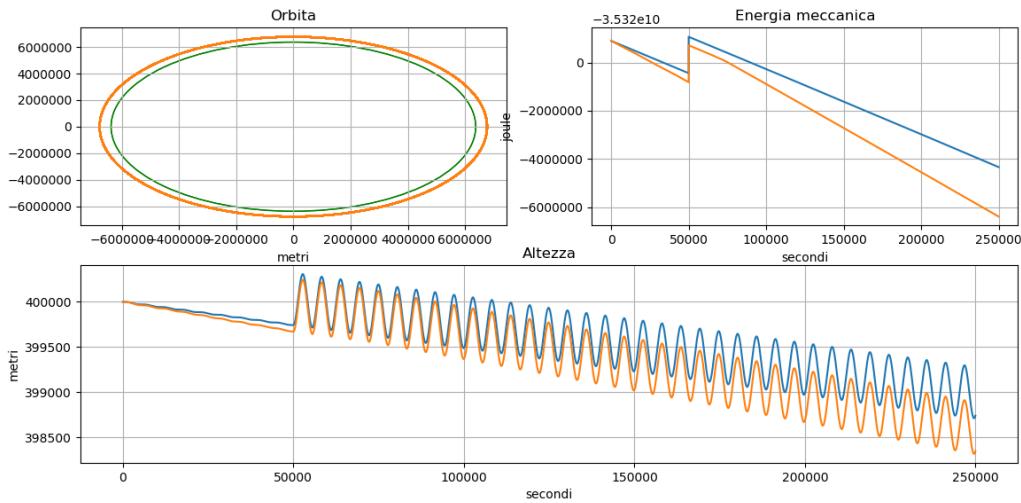
Se decidiamo di simulare l'attivazione di un motore nel sistema dobbiamo assegnare dei valori alle variabili acc_motore , t_acc e $durata$. Impostiamo dei valori ragionevoli e manteniamo gli altri parametri invariati:

$$\begin{aligned} altezza &= 400000 \text{ m} \\ acc_motore &= 20 \text{ N (è una forza)} \\ t_acc &= 50000 \text{ s} \end{aligned}$$

$$\begin{aligned}durata &= 10 \text{ s} \\h_inter &= 0.5 \\x_intervallo &= 500000 \text{ s}\end{aligned}$$

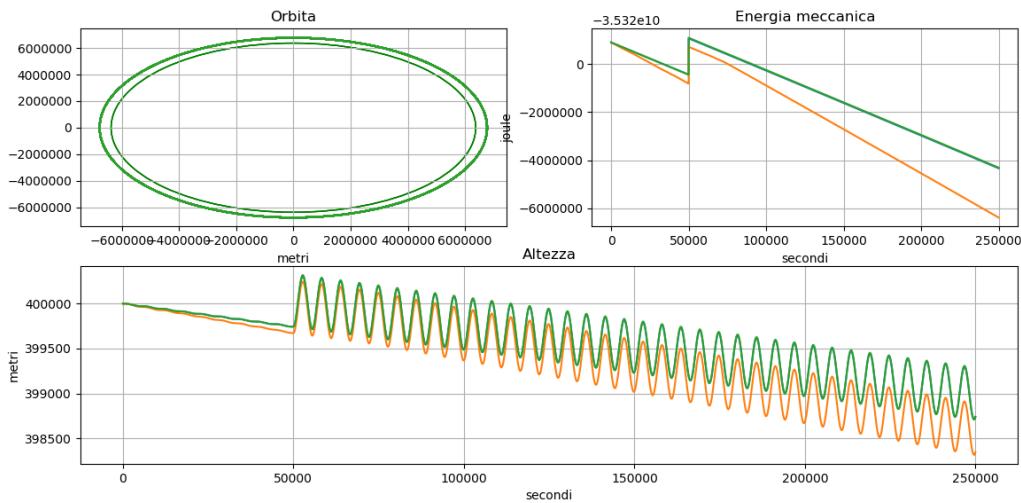
Eseguendo le simulazioni con RK4 e *odeint* otteniamo i seguenti grafici, in cui la curva blu è per RK4 e la curva arancione è per *odeint*. In questo caso alla funzione *odeint* non è stato dato come parametro di input nessun passo h (h_{max}).

*Figura 21: Simulazione del moto di un satellite dotato di motore. La curva blu è RK4 mentre la curva arancione è *odeint* (senza input h_{max}).*



Dando in input a *odeint* il parametro $h_{max} = h_inter$, si ottiene il seguente grafico che mostra come *odeint* (curva verde) e RK4 (curva blu) siano sostanzialmente identici.

*Figura 22: Dati della Figura 21 con in più i dati risultanti dall'esecuzione di *odeint* con input $h_{max} = h_inter$. Notiamo come la curva verde che rappresenta *odeint* sia sovrapposta alla curva blu di RK4.*



Se impostiamo invece una forza frenante nel sistema, cioè impostiamo *acc_motore* negativa, simuliamo la presenza di un motore frenante nel satellite. Vediamo i risultati per i seguenti parametri:

$$acc_{motore} = -20 \text{ N}$$

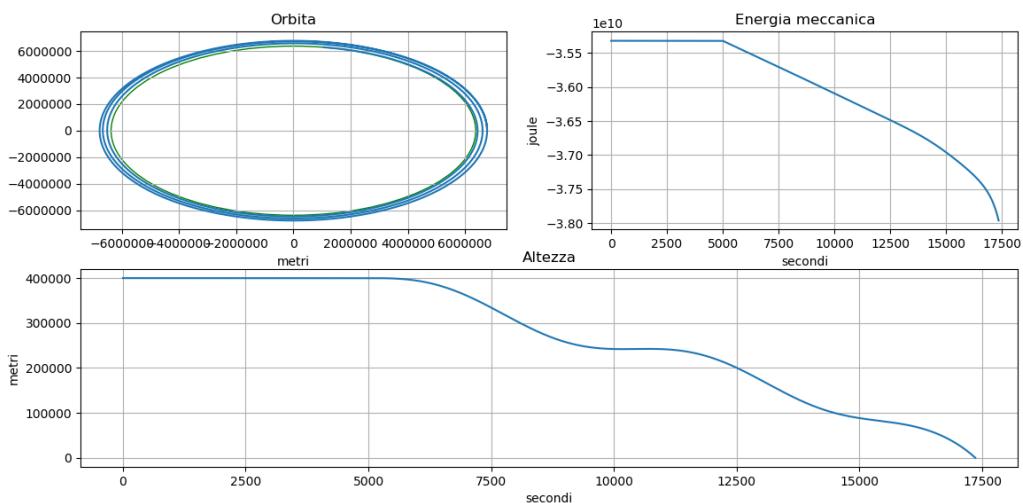
$$t_{acc} = 5000 \text{ s}$$

$$durata = 100000 \text{ s}$$

$$h_{inter} = 0.5 \text{ s}$$

$$x_{intervallo} = 100000 \text{ s}$$

Figura 23 : Simulazione dell' orbita di un satellite dotato di una forza frenante usando RK4.



Queste simulazioni permettono ad esempio di calcolare quanti chilometri il satellite percorra prima dell'atterraggio (o dello schianto) e quindi anche quante orbite effettuerà intorno al pianeta.

4.2 Simulazioni del pendolo composto

4.2.1 Panoramica di evoluzioni

Prendiamo ora in esame il pendolo composto, un altro sistema non lineare, la cui evoluzione è descritta dall'Eq.(4). Vedremo che scegliendo parametri anche di poco diversi si avranno simulazioni con differenze del tutto imprevedibili.

Iniziamo impostando i parametri nel file `params_pendolo.py` come segue:

$$\begin{aligned} m &= 2 \text{ Kg} \\ l &= 1 \text{ m} \end{aligned}$$

$$I = 5 \text{ Kg} * m^2$$

$$\text{beta} = 1 \text{ Kg}$$

$$F_{amp} = 4 \text{ N}$$

$$\omega = 2 \frac{\text{rad}}{\text{s}}$$

$$g = 9.81 \frac{\text{m}}{\text{s}^2}$$

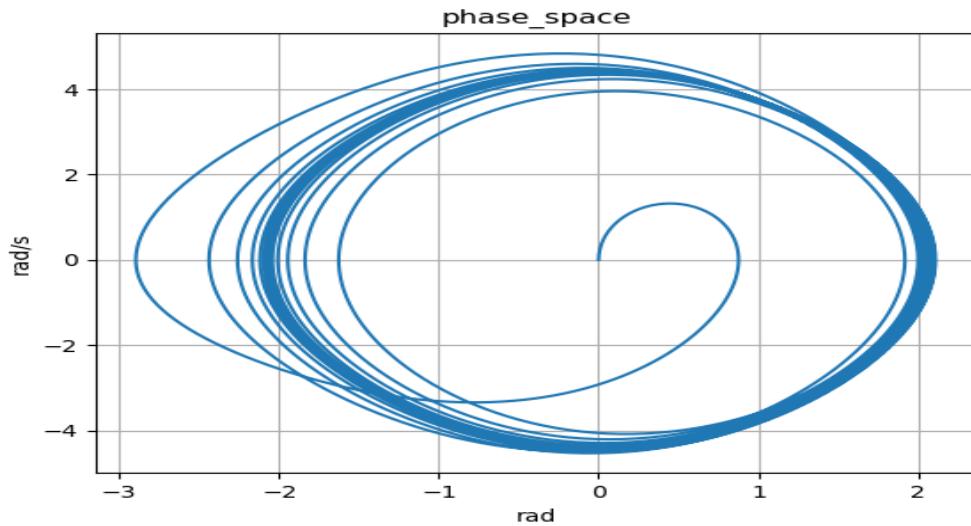
Come parametri inziali e di integrazione fissiamo:

$$\begin{aligned} \theta_0 &= 0 \\ \omega_0 &= 0 \end{aligned}$$

$$\begin{aligned} h_{inter} &= 0.01 \text{ s} \\ x_{intervallo} &= 50000 \text{ s} \end{aligned}$$

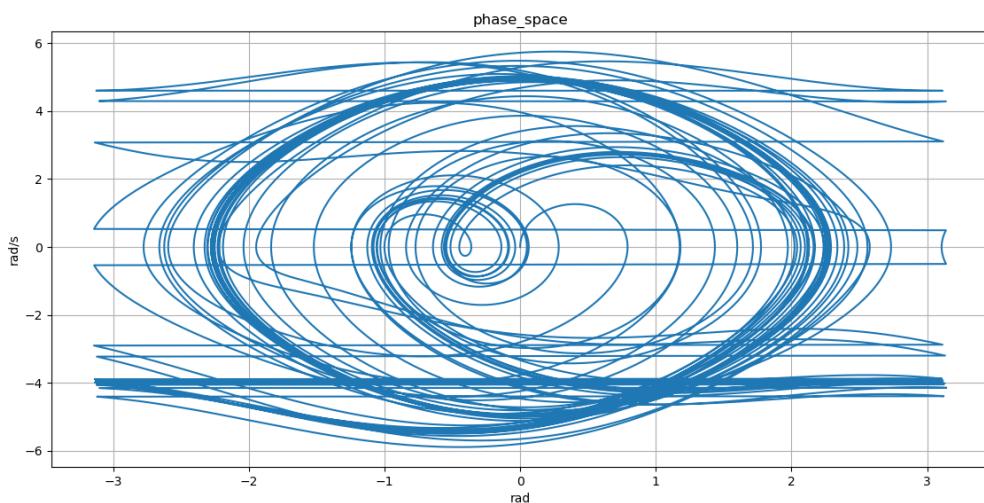
Con queste scelte otteniamo il seguente grafico che mostra come la velocità angolare in funzione della posizione angolare.

Figura 24: Simulazione del pendolo composto con RK4.



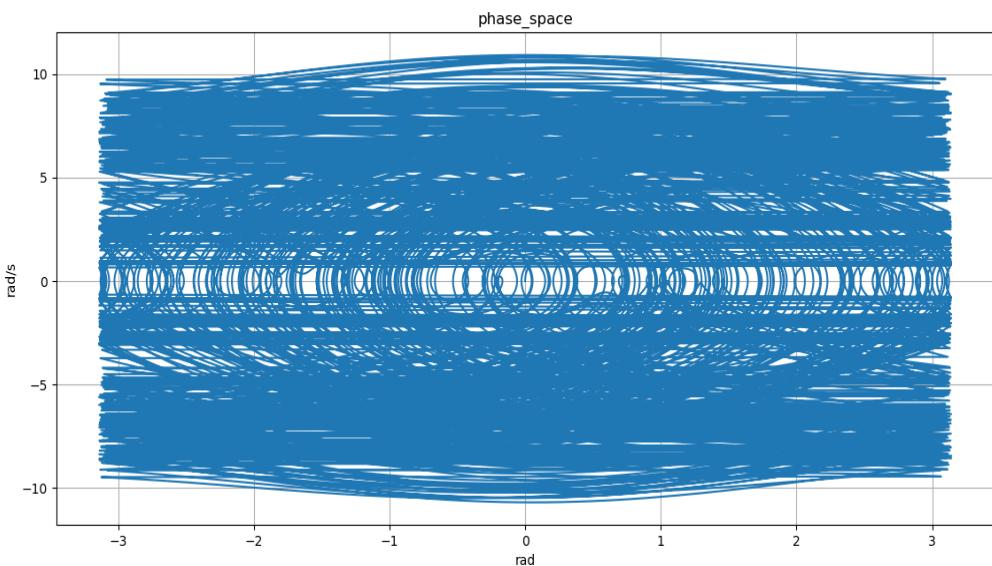
Variando il momento di inerzia da $5 \text{ Kg} * \text{m}^2$ a $6 \text{ Kg} * \text{m}^2$ e aumentando la lunghezza di 0.5 m otteniamo la seguente evoluzione in cui si intravedono nuovamente delle traiettorie nello spazio delle fasi che vengono percorse ripetutamente prima di giungere nello stato di quiete:

Figura 25: Simulazione del pendolo composto con RK4 variando il momento d'inerzia e la lunghezza rispetto alla Figura 24.



Se impostiamo invece la variabile F_{amp} a 10 N , mantenendo invariati tutti gli altri parametri, notiamo come il sistema non si stabilizzi in nessuno stato.

Figura 26: Simulazione del pendolo composto con $F_{amp} = 10$ eseguito con RK4.



Piccole variazioni dei dati iniziali provocano enormi variazioni nell’evoluzione del sistema: siamo quindi di fronte ad un sistema dal comportamento caotico.

4.2.2 Caos e diagramma delle biforcazioni

Per visualizzare il comportamento caotico del modello in esame, si può usare un grafico detto grafico delle biforcazioni. Esso mostra come la variazione di un parametro modifichi i punti di equilibrio del sistema e la loro stabilità, generando una struttura ricorrente frattale.

Per costruire il grafico delle biforcazioni è necessario tenere costanti tutti i parametri del sistema eccetto uno che subisce variazioni minime: per ogni valore di tale parametro si ripete la simulazione dell’evoluzione del sistema. Tale operazione viene quindi ripetuta un numero N di volte; per ogni simulazione, dopo un transiente iniziale di 500 passi di integrazione (corrispondenti a 250 s di evoluzione), si registra il valore della posizione angolare. A questo punto si crea un grafico in cui sull’asse orizzontale rappresentiamo il parametro del sistema che stiamo perturbando e sull’asse verticale il valore di θ registrato. La scelta di monitorare θ è puramente arbitraria: otterremmo qualitativamente gli stessi risultati utilizzando la velocità angolare.

Per il pendolo abbiamo scelto come variabile da perturbare F_{amp} (l’intensità della forzante esterna). Anche qui, abbiamo libertà di scegliere quale parametro fisico perturbare per evidenziare il comportamento caotico del modello. Le rimanenti variabili sono impostate come segue:

$$\begin{aligned} m &= 2 \text{ Kg} \\ l &= 1 \text{ m} \\ I &= 5 \text{ Kg} * \text{m}^2 \end{aligned}$$

$$\begin{aligned} \text{beta} &= 1 \text{ Kg} \\ \omega &= 2 \frac{\text{rad}}{\text{s}} \end{aligned}$$

$$g = 9.81 \frac{\text{m}}{\text{s}^2}$$

Come parametri iniziali e di integrazione per le varie simulazioni, infine, sono stati impostati i valori:

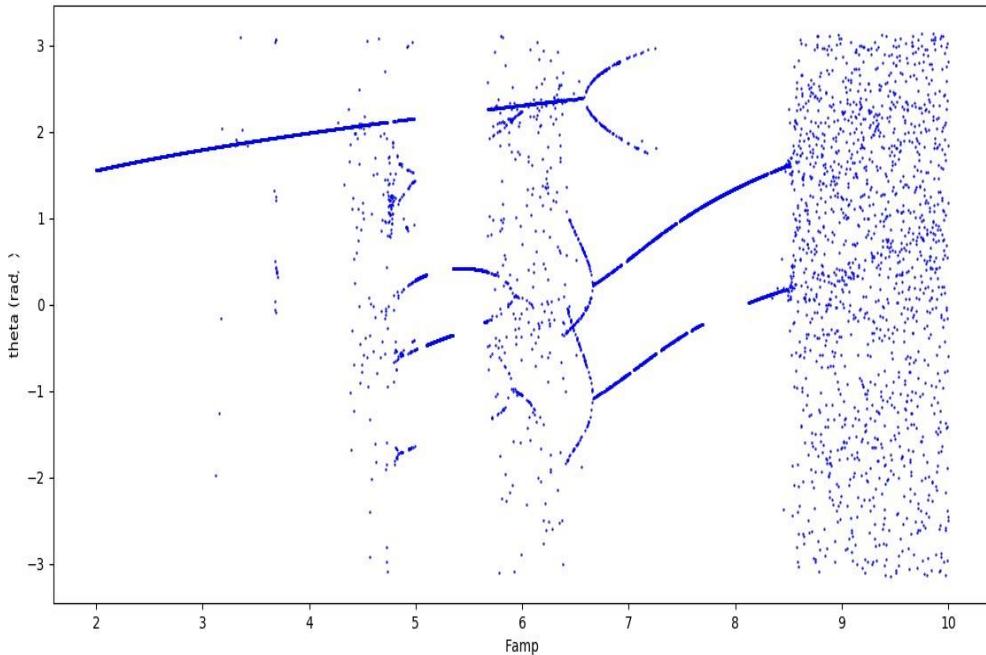
$$\begin{aligned} \theta_0 &= 0 \\ \omega_0 &= 0 \\ h_{\text{inter}} &= 0.5 \\ x_{\text{intervallo}} &= 500 \end{aligned}$$

La variabile F_{amp} viene perturbata di 0.001 a partire da $F_{\text{amp}} = 2 \text{ N}$ fino a $F_{\text{amp}} = 10 \text{ N}$. A tale scopo, impostiamo le due variabili :

$$\begin{aligned} f_{\text{amp_max}} &= 10 \\ passo &= 0.001 \end{aligned}$$

Il diagramma delle biforcazioni che otteniamo è riportato in Figura 27¹.

Figura 27: Grafico delle biforcazioni generato con le simulazioni di `odeint`.



¹ Per visualizzare un grafico delle biforcazioni si deve impostare la variabile `biforcazione` a `True` nel file `param_pendolo.py` e bisogna eseguire il file `plot_biforcazione_chaos.py` .

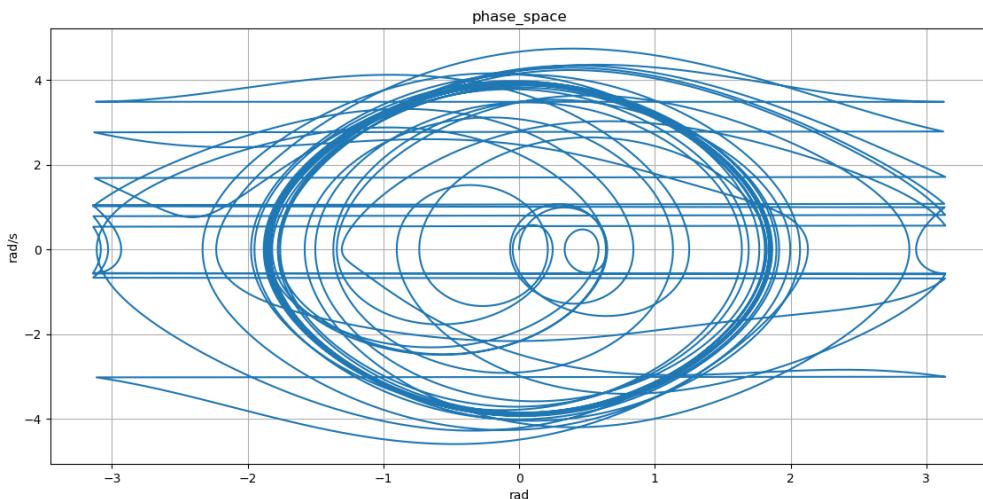
Questo grafico fornisce delle informazioni importanti. Innanzitutto, possiamo notare come per F_{amp} che va da 2 N a circa 4.5 N i valori di θ registrati si dispongano su una curva continua. In questo regime il sistema risponde a perturbazioni di F_{amp} con piccole variazioni del proprio comportamento. Per valori maggiori di F_{amp} , il sistema alterna comportamenti altamente imprevedibili (per esempio per F_{amp} da circa 8.5 N a 10 N) a regimi in cui predilige un numero finito di valori di θ (per esempio per F_{amp} da circa 7.5 N a 10 N). Nelle zone del grafico corrispondenti a tali regimi “più ordinati”, notiamo una forma caratteristica: le biforcazioni. Nel passare per una biforcazione, il numero degli stati di periodo accessibili al sistema (e da esso assunti) varia da un numero finito ad un altro. La particolare struttura appena descritta ricorre spesso nell’analisi dei sistemi dinamici non lineari e appare in altri contesti (per esempio nella mappa logistica). Si noti, infine, come la transizione dal regime con un numero finito di stati di periodo al regime caotico sia brusca.

Da tale diagramma possiamo dedurre per quali valori della variabile perturbata il sistema diventi caotico. Generare questo grafico, tuttavia, è complicato dal punto di vista della complessità computazionale. Il grafico sopra descritto ha richiesto circa 30 minuti di calcolo in un computer con processore dual core Intel i5 di medie prestazioni.

Andiamo ora a vedere il comportamento del pendolo nello spazio delle fasi (θ vs. θ') per alcuni valori specifici di F_{amp} , in modo tale da poter osservare direttamente stati di periodo e regime caotico.

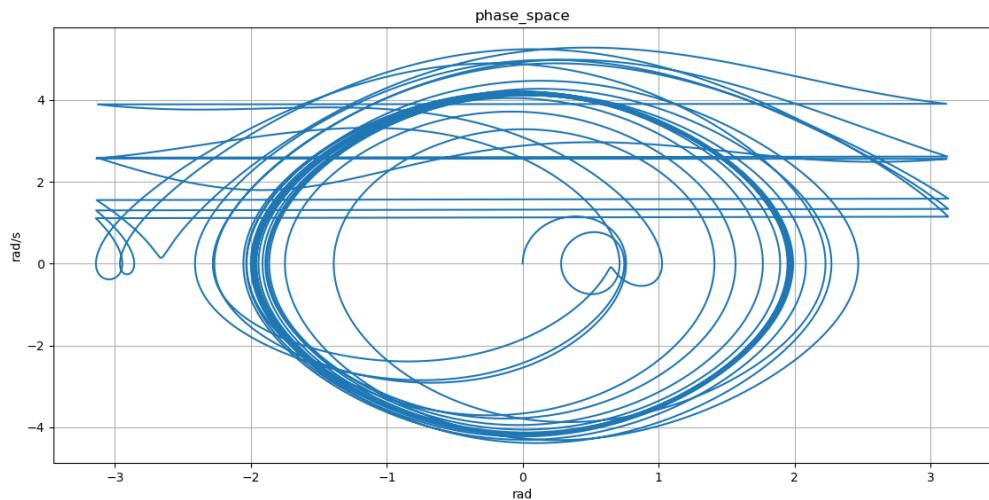
Per $F_{amp} = 3\text{ N}$, il diagramma delle biforcazioni indica uno stato di periodo. Nello spazio delle fasi si osserva effettivamente una traiettoria ricorrente, ovvero la curva più marcata nel grafico. Essa evidenzia che il sistema in quella dinamica è pressoché stabile.

Figura 28 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 3\text{ N}$



Lo stesso vale per $F_{amp} = 3.5\text{ N}$, come mostrato di seguito.

Figura 29 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 3.5 \text{ N}$.



Per valori di F_{amp} circa pari a 6 N , il diagramma delle biforcazioni mostra un certo disordine, indice del fatto che l'evoluzione del sistema passi per un regime caotico. Mostriamo ora i risultati di tre simulazioni che verificano tale comportamento.

Figura 30 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 6.1 \text{ N}$.

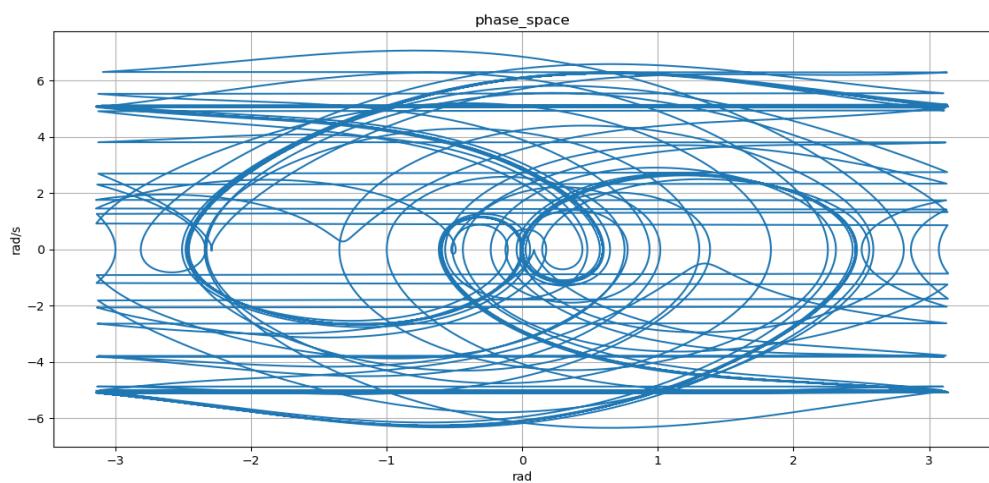


Figura 31 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 6.22 N$.

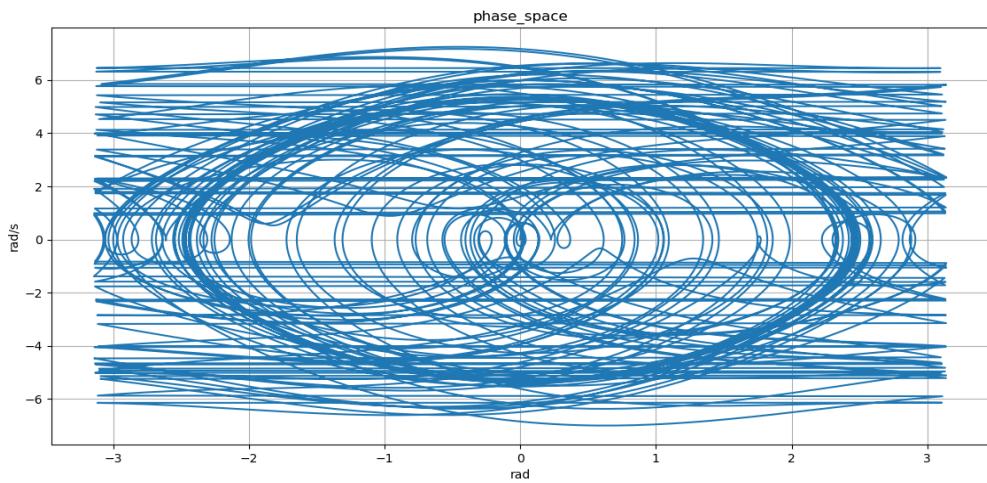
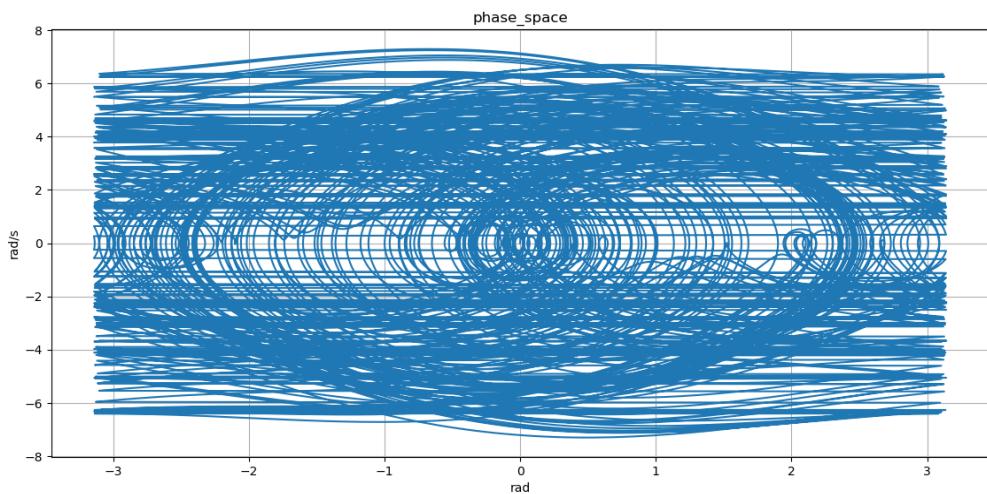


Figura 32 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 6.25 N$.



Come discusso in precedenza, per valori di F_{amp} tra $6 N$ e $8.5 N$ il pendolo può stabilizzarsi in 2, 3 o 4 stati. I grafici delle seguenti simulazioni illustrano in dettaglio questo comportamento.

Figura 33 : Simulazione del pendolo con *odeint* con parametro $Famp = 6.522 \text{ N}$.

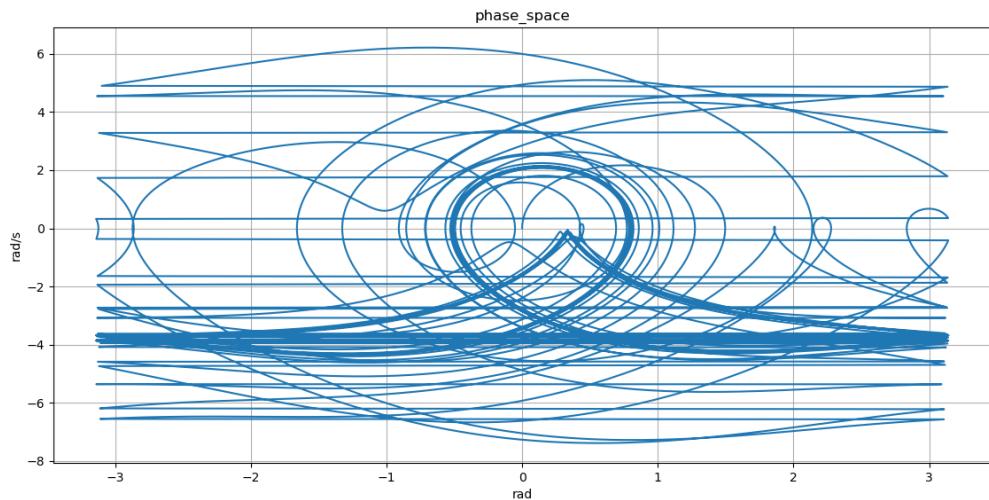
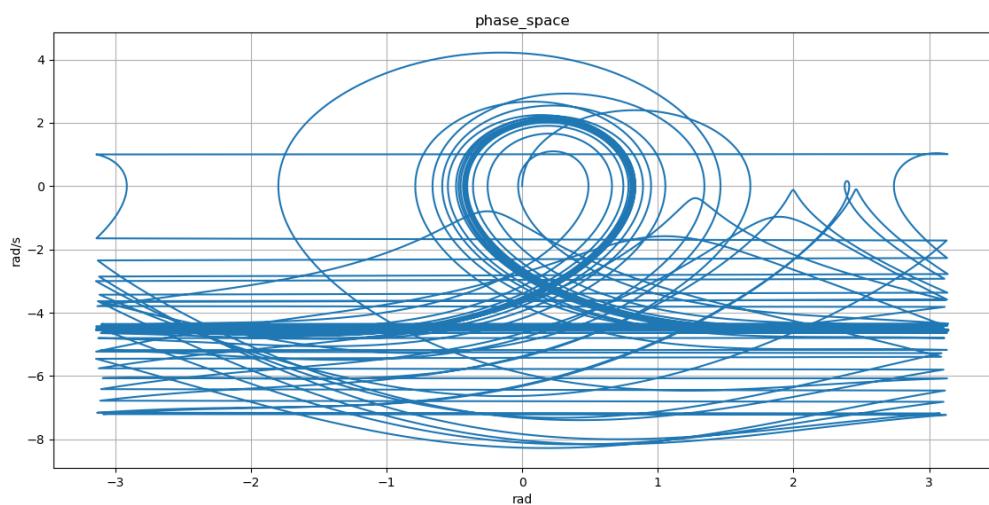


Figura 34 : Simulazione del pendolo con *odeint* con parametro $Famp = 7.20 \text{ N}$.



Secondo il grafico delle biforcazioni dopo il valore 8.5 N della variabile $Famp$ il sistema diventa caotico. Ancora una volta, possiamo illustrare tale affermazione con dei grafici nello spazio delle fasi.

Figura 35 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 8.5 \text{ N}$.

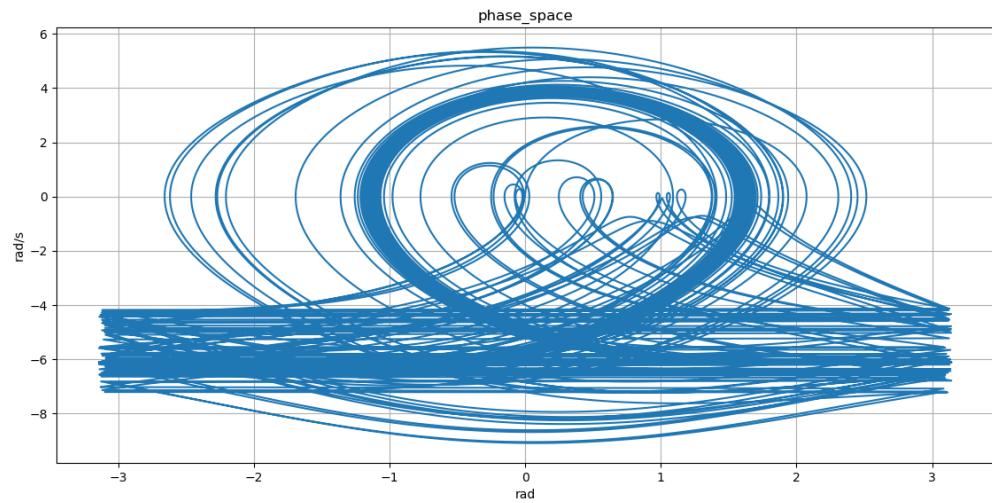


Figura 36 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 8.6 \text{ N}$.

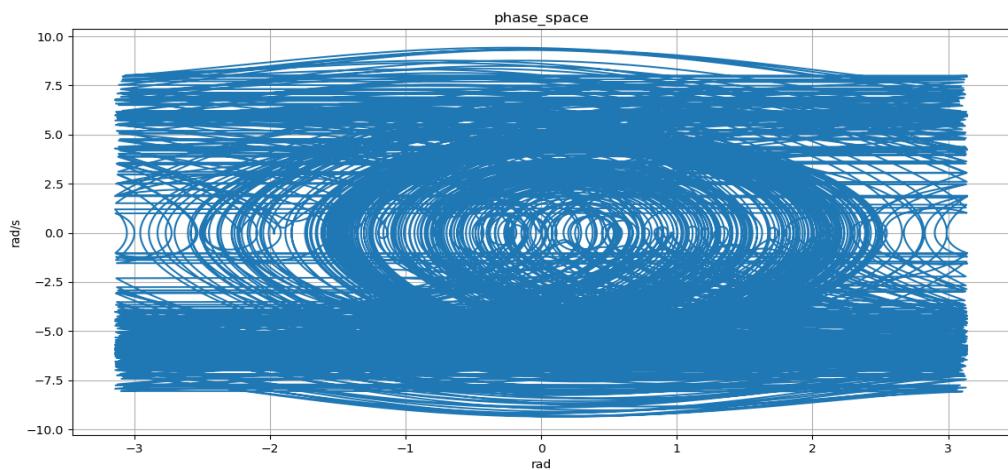
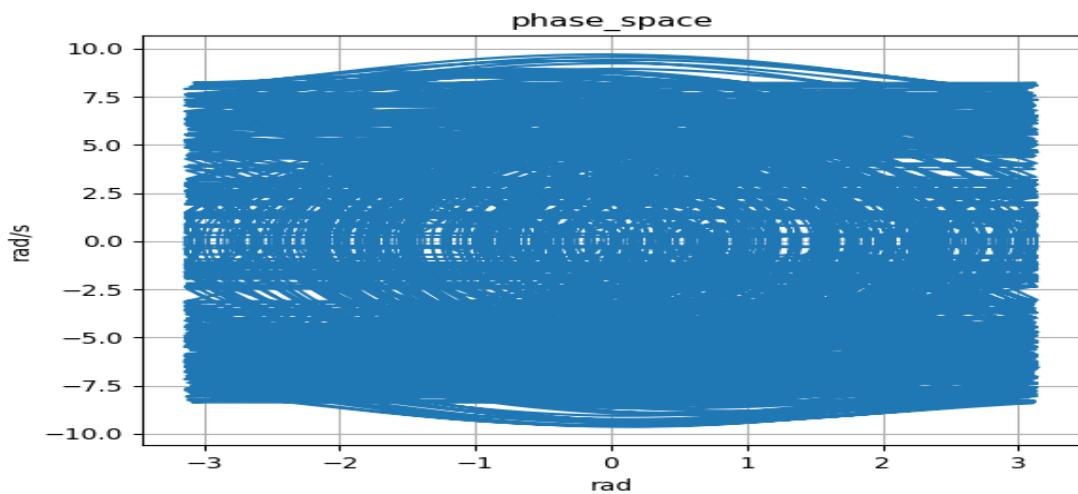


Figura 37 : Simulazione del pendolo con `odeint` con parametro $F_{amp} = 9 \text{ N}$.



Capitolo 5

Conclusioni

In questa relazione di tirocinio sono state descritte le più comuni tecniche per ottenere algoritmi numerici che approssimino la soluzione di sistemi di equazioni differenziali ordinarie. In seguito, sono stati descritti tre famosi algoritmi aventi diverso ordine di precisione: il metodo di Eulero, il metodo di Verlet e il metodo Runge-Kutta del quarto ordine (RK4).

Tali algoritmi sono stati implementati ed utilizzati per studiare due modelli fisici non lineari, al fine di confrontare le prestazioni dei diversi metodi. Nei confronti è stato anche utilizzata la funzione *odient* della libreria *Scipy* che implementa un algoritmo di integrazione con passo adattivo, al contrario di Eulero, Verlet ed RK4 che sono a passo fisso.

Dai nostri sistemi presi in esame, cioè il moto di un satellite in un'atmosfera e l'evoluzione temporale di un pendolo composto sottoposto ad una forzante esterna, emerge che gli algoritmi di ordine elevato (RK4 e *odeint*) sono quelli più adatti per l'evoluzione di tali sistemi. In tutte le simulazioni è emersa (l'ovvia) l'inadeguatezza del metodo di Eulero dovuta essenzialmente al suo ordine basso. Secondo i nostri risultati, *odeint* sembra avere un ordine e una precisione ancora più elevata di RK4, almeno nella simulazione del pendolo.

Nella simulazione del moto di un satellite in un'atmosfera è stato fondamentale determinare i parametri del modello; infatti, poiché l'attrito fra il satellite e l'atmosfera dipende da una serie di variabili, è stata necessaria una loro giusta modellizzazione. I due parametri da determinare nel modello che hanno un impatto significativo sulle simulazioni sono [7]: A_p ed $F10.7$. Il primo rappresenta l'indice geomagnetico e ha valori compresi tra 0 e 400, il secondo è il flusso radio solare e varia da 65 a 300 SFUs [Solar Flux Units; 1 SFU = 10^{-22} Watts/m² Hz] a seconda dell'attività solare.

Nelle nostre simulazioni abbiamo scelto i valori $A_p = 80$ e $F10.7 = 100$ che hanno rappresentato uno scenario dell'atmosfera nella media delle condizioni.

Nella parte finale della tesi, lo sviluppo del diagramma delle biforcazioni per il pendolo composto sottoposto ad una forzante esterna ci ha consentito mostrare direttamente il comportamento caotico di tale sistema fisico. Generare tale grafico con un'elevata precisione richiede un grande sforzo computazionale. In uno scenario applicativo, un investimento di risorse di questo tipo può essere vantaggioso poiché i diagrammi di biforcazione sono uno strumento molto utile nell'analisi dei sistemi.

In questa relazione sono state effettuate diverse semplificazioni. Innanzitutto, non è stata affrontata tutta la parte teorica della stabilità degli algoritmi, cioè la zero stabilità

e l'assoluta stabilità. Inoltre, non abbiamo discusso in dettaglio come il passaggio da una singola equazione differenziale a un sistema di equazioni possa influenzare la precisione dei metodi numerici utilizzati.

In letteratura esistono altri metodi di risoluzione ancora più avanzati di quelli implementati durante questo tirocinio, quali i metodi multi-step o i metodi impliciti. Essi possono permettere di trattare in maniera più sofisticata l'evoluzione numerica di modelli più complessi di quelli presi in esame da noi e, in particolare, dei cosiddetti problemi "stiff" (casi in cui alcuni termini delle equazioni differenziali da integrare esibiscono forti variazioni).

Va notato che i sistemi di equazioni differenziali ordinarie, sebbene utili a modellizzare un elevato numero di fenomeni, non sono sufficienti a descrivere altri tipi fenomeni, i quali richiedono invece un trattamento mediante l'uso di equazioni differenziali alle derivate parziali.

Mostriamo, infine, la seguente tabella che mostra il tempo di calcolo (in media) di alcune delle simulazioni numeriche effettuate precedentemente su un computer di medie prestazioni con CPU: Intel i5 6267U @ 2.9 GHz.

Figura	Tempo di calcolo
6	~57 s
8	~2 min
15	~53 s
16	~27 s
17	~35 s
18	~24 s
19	~3 min
24	~5 s
26	~5 s
27	~30 min

Bibliografia

- [1] Quarteroni A. Sacco R. Saleri F. Gervasio P. *Matematica Numerica* , Springer Verlag , 4° edizione , 2014.
- [2] Gori L. *Calcolo Numerico* , Edizioni Kappa , 5° edizione , 2006.
- [3] Butcher J. C. *Numerical Methods for Ordinary Differential Equations* , John Wiley & Sons , 2008.
- [4] Ricci F. Fiorentini V. *Fondamenti di Fisica Computazionale*, 2018.
- [5] Bramanti M. D. Pagani C. Salsa S. *Analisi Matematica 2* , Zanichelli , 2009.
- [6] Di Nola A. Appunti di Dinamica Molecolare, 2006.
- [7] Acedo L. Kinematics effects of atmospheric friction in spacecraft flybys , Advances in Space Research , 2017.
- [8] Gray D. Davidson The Damped Driven Pendulum : Bifurcation Analysis of Experimental Data , 2011.
- [9] Guo Xiao, Chaotic Physical Pendulum , 2016.
<https://www.zybuluo.com/guoxiaowhu/note/353303>
- [10] http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node5.html
- [11] <https://www.compadre.org/PICUP/resources/Numerical-Integration/>