Modellazione e rappresentazione digitale delle orbite dei satelliti medicei

Michele Antonio Portulano

May 27, 2025

Abstract

In questo articolo vengono presentati tre metodi digitali per rappresentare graficamente, con Python e le librerie `matplotlib` e `spiceypy`, le orbite dei satelliti medicei, a partire dalle loro caratteristiche matematiche, fisiche e dai dati di osservazione. A più di quattro secoli dalla loro scoperta da parte di Galileo Galilei, nel gennaio del 1610, i quattro principali satelliti di Giove — Io, Europa, Ganimede e Callisto — continuano a rivestire un ruolo centrale nello studio della dinamica orbitale.
Le traiettorie, quasi perfettamente ellittiche, sono caratterizzate da eccentricità molto basse: $e = 0{,}0041$, 0,0094, 0,0011 e 0,0074 rispettivamente. Ad oggi, i satelliti noti di Giove sono saliti a 92, ma questi quattro rimangono i più studiati per importanza storica e rilevanza scientifica.
I metodi presentati includono: una rappresentazione teorica basata su eccentricità e lato semiretto o parametro orbitale; una ricostruzione bidimensionale fondata su dati SPICE; e una visualizzazione tridimensionale delle orbite nel sistema di riferimento inerziale J2000.

1 Le orbite

Prima di rappresentare graficamento uno dei sistemi più affascinanti e studiati della storia dell'astronomia, è necessario chiarire alcuni aspetti fondamentali legati alla geometria e alla fisica delle orbite.
Tra il 1609 e il 1619, l'astronomo e matematico Johannes Kepler (italianizzato in Giovanni Keplero), basandosi sulle osservazioni di Tycho Brahe, formulò tre leggi fondamentali per la comprensione del moto dei pianeti attorno al Sole, ponendo così le basi della meccanica celeste moderna.

Di seguito vengono presentate sinteticamente le tre leggi di Keplero, essenziali per descrivere la geometria e la dinamica delle orbite planetarie:

1. **Prima legge (legge delle orbite)**: ogni pianeta descrive un'orbita ellittica attorno al Sole, che occupa uno dei due fuochi dell'ellisse.
2. **Seconda legge (legge delle aree)**: il segmento che unisce il pianeta al Sole copre aree uguali in intervalli di tempo uguali. Questo implica che il pianeta si muove più velocemente quando è più vicino al Sole (perielio) e più lentamente quando è più lontano

(afelio):

$$\frac{dA_2}{dt} > \frac{dA_1}{dt}, \quad A_2 > A_1$$

3. **Terza legge (legge dei periodi)**: il quadrato del periodo orbitale T di un pianeta è proporzionale al cubo del semiasse maggiore a della sua orbita:

$$\frac{T^2}{a^3} = \frac{4\pi^2}{G\,m_2}, \quad T^2 \propto a^3$$

L'orbita di un corpo celeste può essere descritta come una sezione conica e, in particolare, come un'ellisse nel caso di orbite chiuse, in coordinate polari. Considerando l'eccentricità e e il parametro orbitale (lato semiretto) $l = a(1-e^2)$, e sostituendo nell'equazione polare della conica $r = \frac{l}{1+e\cos\vartheta}$, è possibile esprimere l'orbita mediante le seguenti equazioni parametriche:

$$\begin{cases} x = \frac{l}{1+e\cos\vartheta}\cos\vartheta \\ y = \frac{l}{1+e\cos\vartheta}\sin\vartheta \end{cases} \tag{1}$$

2 Rappresentazione digitale delle orbite medicee

A questo punto è possibile rappresentare digitalmente le orbite dei satelliti medicei, utilizzando il linguaggio Python e la libreria `matplotlib` per la generazione dei grafici.

Verranno presentati tre approcci distinti:

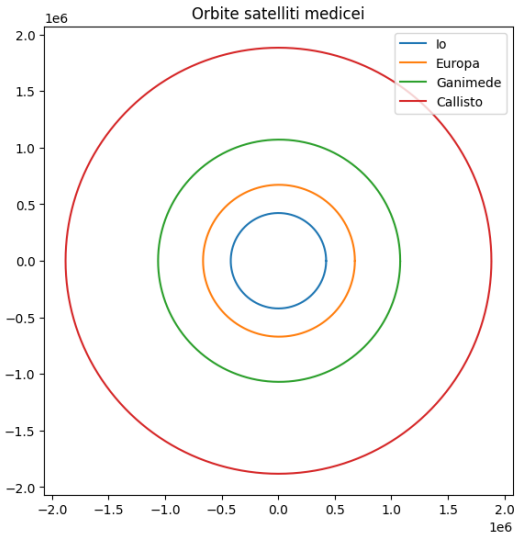
- Un primo metodo, basato su una rappresentazione teorica dell'orbita ellittica costruita a partire dall'eccentricità e dal lato semiretto o parametro orbitale di ogni satellite.
- Un secondo metodo, più accurato, fondato sull'utilizzo di dati di posizione effettivi estratti dai database forniti dal sistema SPICE (*Spacecraft Planet Instrument C-matrix Events*), sviluppato dal NAIF (Navigation and Ancillary Information Facility) del *Jet Propulsion Laboratory* (JPL/NASA).
- Un terzo metodo, che consente una rappresentazione tridimensionale delle orbite, sempre a partire da dati SPICE, ma includendo anche la coordinata spaziale z, così da visualizzare le orbite complete nello spazio tridimensionale nel sistema di riferimento inerziale J2000.

I programmi sviluppati per l'elaborazione delle orbite e delle simulazioni descritte sono disponibili nella dispensa indicata tra le References [2].

2.1 Metodo basato su eccentricità e lato semiretto

Nel primo approccio, definite le eccentricità e i lati semiretti di ciascuna orbita satellitare come variabili, è possibile richiamare tali parametri all'interno della funzione `polar`, come illustrato tra le righe 15 e 19, ottenendo così le coordinate x, y dei punti orbitali.

```python
import matplotlib.pyplot as plt
import numpy as np

theta = np.arange(0, 2*np.pi, np.pi/360)
e_Io = 4.1*10**(-3)
e_Europa = 9.4*10**(-3)
e_Ganimede = 7.4*10**(-3)
e_Callisto = 1.1*10**(-3)

l_Io = 421700
l_Europa = 671100
l_Ganimede = 1070400
l_Callisto = 1882700

def polar(e, l, theta):
    r = l / (1 - e * np.cos(theta))
    x = r * np.cos(theta)
    y = r * np.sin(theta)
    return x, y

x_Io, y_Io = polar(e_Io, l_Io, theta)
x_Europa, y_Europa = polar(e_Europa, l_Europa, theta)
x_Ganimede, y_Ganimede = polar(e_Ganimede, l_Ganimede, theta
    )
x_Callisto, y_Callisto = polar(e_Callisto, l_Callisto, theta
    )

plt.figure(figsize=(10, 10))
plt.plot(x_Io, y_Io, label='Io')
plt.plot(x_Europa, y_Europa, label='Europa')
plt.plot(x_Ganimede, y_Ganimede, label='Ganimede')
plt.plot(x_Callisto, y_Callisto, label='Callisto')
plt.gca().set_aspect('equal', adjustable='box')
plt.legend()
plt.show()
```



2.2 Rappresentazione mediante SPICE

Un secondo metodo, più accurato, per rappresentare lungo il piano xy le orbite medicee è quello di estrarre da un database scientifico le coordinate del satellite per l'intera durata del proprio periodo orbitale.

Attraverso l'ambiente di calcolo SPICE, sviluppato e mantenuto dal *Navigation and Ancillary Information Facility (NAIF)*, una divisione del *Jet Propulsion Laboratory (NASA/JPL)*, è possibile accedere a dati ad alta precisione riguardanti la posizione e la velocità dei corpi celesti.

Il NAIF fornisce un insieme di *kernel SPICE*, file contenenti informazioni su:

- il tempo (kernel di tipo `LSK`),
- le orbite planetarie e satellitari (`SPK`),
- costanti fisiche e modelli di orientamento (`PCK`),
- e altri parametri geometrici rilevanti.

Tali kernel sono distribuiti tramite il *Planetary Data System (PDS)* della NASA, all'interno del quale il NAIF costituisce un nodo tematico specializzato nella navigazione e nella geometria planetaria.

Utilizzando SPICE, è possibile calcolare con precisione la posizione istantanea di un satellite rispetto al suo corpo centrale (in questo caso, i satelliti galileiani rispetto al baricentro di Giove), per una sequenza temporale che copre l'intero periodo orbitale. Questi dati possono poi essere xy per ottenere una rappresentazione planare dell'orbita apparente nel sistema di riferimento

inerziale J2000. Tale proiezione corrisponde alla vista "dall'alto" dell'orbita, eliminando la componente z e quindi approssimando l'orbita con una curva ellittica in due dimensioni.

Caricati, come mostrato tra le righe 6 e 8, i kernel attraverso la funzione `spice.furnsh()`, è necessario dichiarare, tramite la funzione `spice.str2et()`, il tempo iniziale t_0 da cui far partire la simulazione.
Successivamente, per ogni satellite (keys) da rappresentare, bisogna definire il codice NAIF (values) utilizzando la seguente struttura:

```
oggetto = {
    'keys': 'values',
}
```

A questo punto è necessario definire la funzione `calcola_orbita(id_satellite, tempo_base, passi=720)` con cui si calcolano le posizioni del satellite in un intervallo di tempo che copre un intero periodo orbitale.

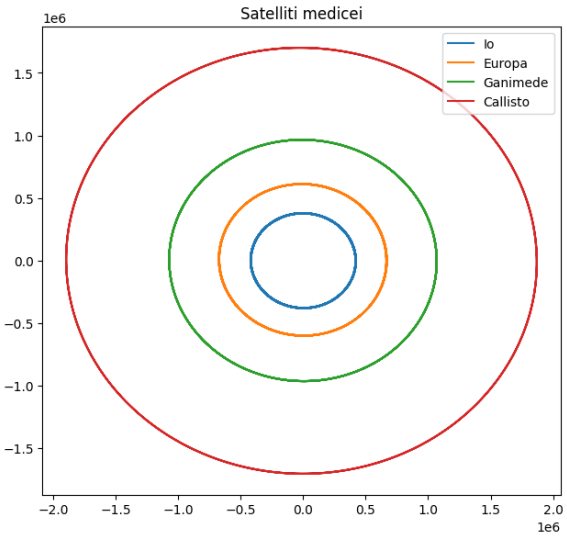
La funzione accetta in input:

- `id_satellite`: il codice NAIF che identifica univocamente il satellite di interesse,
- `tempo_base`: il tempo di riferimento espresso in epoche eteriche, richiamato dalla funzione `spice.str2et()`,
- `passi`: il numero di intervalli temporali in cui suddividere l'orbita per campionare le posizioni (720).

A questo punto, come mostrato tra riga 34 e 37, attraverso un ciclo for per ogni satellite plottiamo i punti xy ignorando i punti z.
Infine, scarichiamo dalla memoria i kernel precedentemente caricati utilizzando la funzione `spice.unload()` per un discorso di efficienza del programma.

```
1  import spiceypy as spice
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  spice.furnsh("Kernel Nasa/naif0012.tls")
7  spice.furnsh("Kernel Nasa/jup365.bsp")
8  spice.furnsh("Kernel Nasa/pck00010.tpc")
9
10
11 tempo_osservazione = spice.str2et("2025-01-01T00:00:00")
12
13 satelliti = {
14     'Io': '501',
15     'Europa': '502',
16     'Ganimede': '503',
```

```
    'Callisto': '504'
}


angoli = np.linspace(0, 2 * np.pi, 720)


def calcola_orbita(id_satellite, tempo_base, passi=720):
    posizioni = []
    for angolo in angoli:
        tempo_shiftato = tempo_base + angolo * 691200
        stato, _ = spice.spkezr(id_satellite, tempo_shiftato
            , "J2000", "NONE", "599")
        posizioni.append(stato[:3])
    return np.array(posizioni)

plt.figure(figsize=(10, 10))

for nome, id_satellite in satelliti.items():
    orbita = calcola_orbita(id_satellite, tempo_osservazione
        )
    x, y = orbita[:, 0], orbita[:, 1]
    plt.plot(x, y, label=nome)

plt.gca().set_aspect('equal')
plt.legend()
plt.title("Satelliti medicei")
plt.grid(True)
plt.show()

spice.unload("Kernel Nasa/naif0012.tls")
spice.unload("Kernel Nasa/jup365.bsp")
spice.unload("Kernel Nasa/pck00010.tpc")
```



3 Rappresentazione tridimensionale delle orbite medicee

In conclusione, utilizzando SPICE è possibile ottenere le coordinate spaziali (x, y, z) dei quattro satelliti medicei rispetto a Giove e rappresentare l'intero sistema tridimensionalmente.

A differenza del primo script, che visualizzava le orbite solo nel piano xy mediante un grafico bidimensionale, la versione completa include anche la coordinata z, consentendo così una rappresentazione tridimensionale. Per questo scopo si importa dalla libreria `matplotlib` il modulo `mpl_toolkits.mplot3d`, con il comando

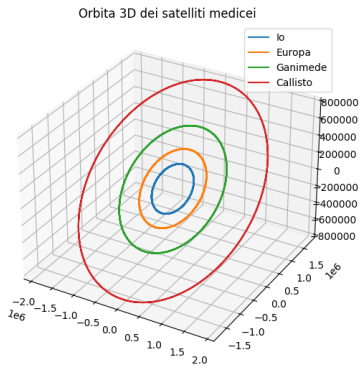
```
from mpl_toolkits.mplot3d import Axes3D
```

che permette di creare grafici nello spazio tridimensionale.

A questo punto, il programma mantiene una struttura simile a quella dello script precedente, fino alla definizione della funzione `calcola_orbita`. L'utilizzo del modulo `mpl_toolkits.mplot3d`, tuttavia, richiede alcune modifiche necessarie per ottenere una rappresentazione tridimensionale delle orbite:

```
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

for nome, id_satellite in satelliti.items():
    orbita = calcola_orbita(id_satellite, tempo_osservazione)
```

```
    x, y, z = orbita[:, 0], orbita[:, 1], orbita[:, 2]
    ax.plot(x, y, z, label=nome)

ax.set_title("Orbita 3D dei satelliti medicei")
ax.legend()
plt.show()
```



References

[1] NASA Navigation and Ancillary Information Facility (NAIF), *Kernel SPICE*, https://naif.jpl.nasa.gov/naif/
Kernel utilizzati: `naif0012.tls`, `jup365.bsp`, `pck00010.tpc`.

[2] Dispensa dei programmi utilizzati
https://github.com/MichyPortu08/Orbite-satelliti-medicei-github/tree/main