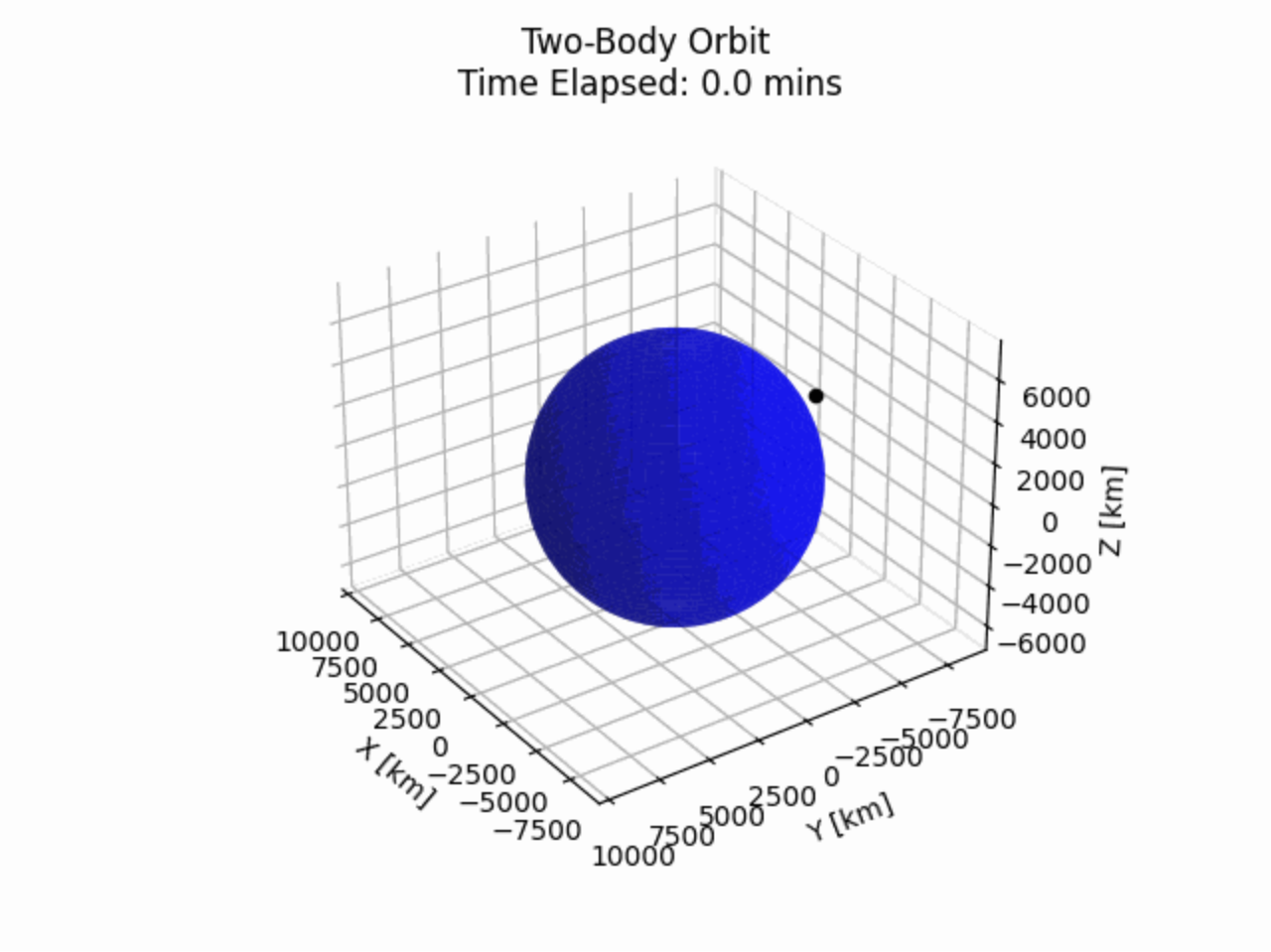
Member-only story

Use Python to Create Two-Body Orbits

Learn how to use Python to determine the motion of a spacecraft under the influence of a gravity of a larger body

Zack Fizell · Following

Published in Towards Data Science · 6 min read · Feb 19, 2022



Two-Body Orbit Animated [Created by Author]

The general two-body problem consists of solving for the motion of two masses that move under the influence of each other's gravity. In orbital mechanics, typically one of the masses is considered to have a negligible mass, causing it to have no effect on the other mass. The motion of the negligible mass is of interest and what is solved for. In reality, the smaller mass does have an effect on the larger mass, but the effect is so minuscule that it is, for all practical purposes, zero.

. . .

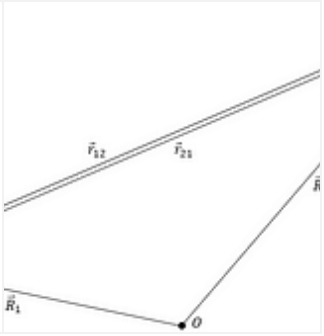
In order to create a two-body orbit in Python, you'll first need to derive the equations of motion for the mass of interest (whether it's a satellite, asteroid,

etc.). I demonstrated how to do this in another article (found below), so I encourage you to explore that and really understand the physics behind the equations. But, you will find the equations of motion below.

How to Solve the Two-Body Problem

Learn the fundamentals of orbital mechanics by deriving the equations of motion for a two-body system

medium.com



The two-body problem is a good learning tool and can be used as a first approximation for orbits that are very close to a large mass, but once our orbit moves away from this mass, other gravitational forces should be included. Even when close to the large mass, perturbing forces such as atmospheric drag, solar radiation, and effects of a non-spherical body should be included. We won't discuss those here, but it's a good thing to keep in mind. The equations of motion for a negligible mass moving under the influence of gravity of a larger mass are as follows:

$$\ddot{x} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\ x$$
$$\ddot{y} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\ y$$
$$\ddot{z} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\ z$$

Now that we have our equations of motion, we need to convert them to a state form that can be numerically integrated. The state form includes the position and velocity vector at a certain time. Here is the state and its time derivative.

$$\begin{Bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{Bmatrix} \quad \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} = -\frac{\mu}{r^3}\, x \\ \ddot{y} = -\frac{\mu}{r^3}\, y \\ \ddot{z} = -\frac{\mu}{r^3}\, z \end{Bmatrix}$$

The state time derivative will be made into a function that can be numerically integrated. Python has built-in numerical integrators that we can utilize for this. To create our orbit, we will be using `odeint` function from the *SciPy* package. Let's start our code that will generate an plot for an Earth orbiting satellite.

Importing Packages

The important packages that we will need for this code are as follows:

- *NumPy* is used for creating numerical arrays and using trigonometric functions (defined as *np* for ease of calling)
- *SciPy* is used for the `odeint` function, which is used for numerically integrating our equations of motion
- `pyplot` from *matplotlib* is used for displaying our orbit at the end of the code

```
# Importing Packages
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

Creating Function for Numerical Integration

The next step is creating a function that will be ran by `odeint` to numerically integrate our state vector. This function will include our velocity and

acceleration for each *x*, *y*, and *z* (defined above). The function should look something like this when created:

```
# Earth Model
def model_2BP(state, t):
    mu = 3.986004418E+05  # Earth's gravitational parameter
                          # [km^3/s^2]
    x = state[0]
    y = state[1]
    z = state[2]
    x_dot = state[3]
    y_dot = state[4]
    z_dot = state[5]
    x_ddot = -mu * x / (x ** 2 + y ** 2 + z ** 2) ** (3 / 2)
    y_ddot = -mu * y / (x ** 2 + y ** 2 + z ** 2) ** (3 / 2)
    z_ddot = -mu * z / (x ** 2 + y ** 2 + z ** 2) ** (3 / 2)
    dstate_dt = [x_dot, y_dot, z_dot, x_ddot, y_ddot, z_ddot]
    return dstate_dt
```

Running ODE Solver

The chosen ODE solver needs three inputs to run: the function to be numerically integrated, an initial condition, and a time array. We already created the function, called `model_2BP`. The initial conditions can be anything (make sure you are outside of the Earth's surface!). NASA has an app that can be used to pull state vectors if you want to look at a particular satellite's orbit. The time array can be created using the *NumPy* package (Note: make sure you include enough time steps to create a smooth orbit). We can run the solver now and set it equal to `sol` (meaning "solution").

```
# Initial Conditions
X_0 = -2500  # [km]
Y_0 = -5500  # [km]
Z_0 = 3400  # [km]
VX_0 = 7.5  # [km/s]
VY_0 = 0.0  # [km/s]
VZ_0 = 4.0  # [km/s]
state_0 = [X_0, Y_0, Z_0, VX_0, VY_0, VZ_0]
```

```
# Time Array
t = np.linspace(0, 6*3600, 200)  # Simulates for a time period of 6
                                 # hours [s]

# Solving ODE
sol = odeint(model_2BP, state_0, t)
X_Sat = sol[:, 0]  # X-coord [km] of satellite over time interval
Y_Sat = sol[:, 1]  # Y-coord [km] of satellite over time interval
Z_Sat = sol[:, 2]  # Z-coord [km] of satellite over time interval
```

Visualizing the Data

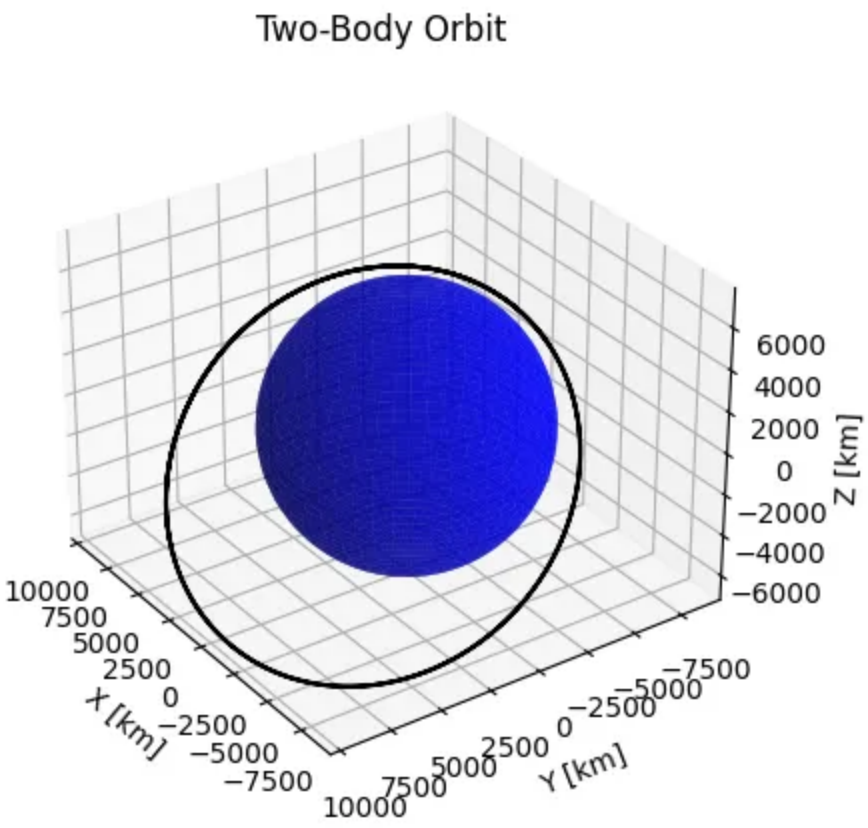
Finally, our end goal of creating the orbits can be accomplished by using *matplotlib* to create a 3D figure. We used the x, y, and z time histories from the `sol` variable, which we already obtained. To add to the visual, we can also create a sphere to represent our Earth (Earth has an average radius of 6378.14 km). We will place our Earth at the origin of our plot.

```
# Setting up Spherical Earth to Plot
N = 50
phi = np.linspace(0, 2 * np.pi, N)
theta = np.linspace(0, np.pi, N)
theta, phi = np.meshgrid(theta, phi)

r_Earth = 6378.14  # Average radius of Earth [km]
X_Earth = r_Earth * np.cos(phi) * np.sin(theta)
Y_Earth = r_Earth * np.sin(phi) * np.sin(theta)
Z_Earth = r_Earth * np.cos(theta)

# Plotting Earth and Orbit
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X_Earth, Y_Earth, Z_Earth, color='blue', alpha=0.7)
ax.plot3D(X_Sat, Y_Sat, Z_Sat, 'black')
ax.view_init(30, 145)  # Changing viewing angle (adjust as needed)
plt.title('Two-Body Orbit')
ax.set_xlabel('X [km]')
ax.set_ylabel('Y [km]')
ax.set_zlabel('Z [km]')
```

We should add one more thing to our plot. My professors always pressed during our courses to have the axes be equally spaced on orbit plots. This will give the true representation of the orbit in physical space. We can do that with the following code.

```
# Make axes limits
xyzlim = np.array([ax.get_xlim3d(), ax.get_ylim3d(),
                   ax.get_zlim3d()]).T
XYZlim = np.asarray([min(xyzlim[0]), max(xyzlim[1])])
ax.set_xlim3d(XYZlim)
ax.set_ylim3d(XYZlim)
ax.set_zlim3d(XYZlim * 3/4)
plt.show()
```

After compiling all of the code and running it, you should get the following orbit. Again, it's an approximation, and as the orbit evolves, it will start to deviate from the true orbit. In order to get a better model for an orbit, you should include perturbation forces, like the Moon's gravity, the Sun's gravity, Earth's non-spherical nature, solar radiation, and even atmospheric drag. I'll be writing an article on this topic at some point, so if you are interested, give me a follow and subscribe to my emails to stay up to date.



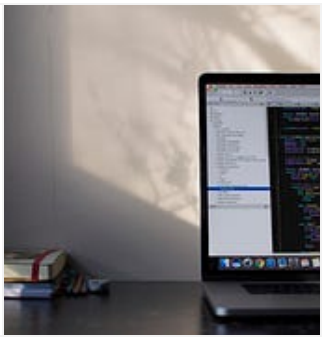
Two-Body Orbit [Created by Author]

As a side note, if you want to animate the orbit like the one at the beginning of the article, you can learn to do so by following along with this article:

How to Animate Plots in Python

Learn the basics of animating data to create dynamic visuals

towardsdatascience.com



• • •

Thank you for reading the article! I'm going to continue to write easy to follow orbital mechanics articles with code and derivations, so if you are interested, please give me a follow! Comment if you have any questions!

Programming Python Technology Science Space

More from the list: "Orbital Mechanics"

Curated by Zack Fizell

Zack ... in Towards Data ...

How to Use MATLAB to Create Two-Body Orbits

6 min read · Mar 25, 2022

Zack Fiz... in ILLUMINATI...

Maneuvering Spacecraft on Orbit

5 min read · Aug 5

Zack Fiz... in ILLUMI

Deriving the Effect o Solar Radiation Pres

7 min read · Jul 4, 2

View list

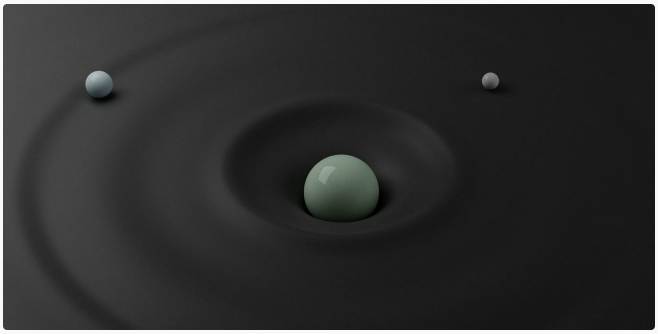
Written by Zack Fizell

Following

1.4K Followers · Writer for Towards Data Science

M.S. in Aeronautics and Astronautics — Articles on Orbital Mechanics| Machine Learning| Coding — https://medium.com/@zackfizell10/membership

More from Zack Fizell and Towards Data Science



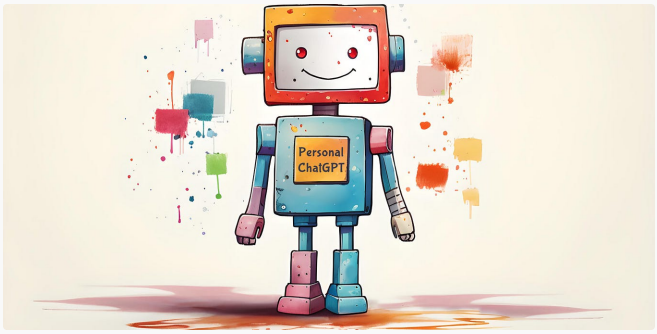
Zack Fizell in Intuition

What Are Gravity Wells?

How do we escape the gravitational pull of a planet?

✦ · 5 min read · May 2, 2022

99 4



Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

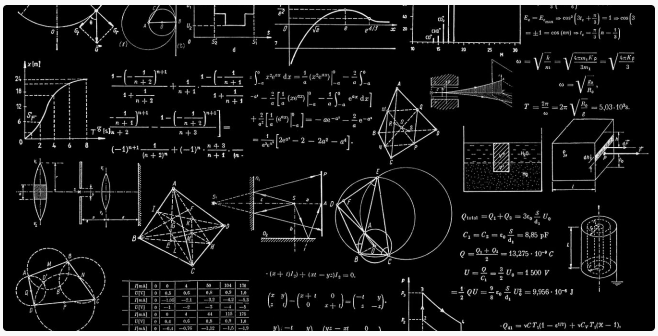
✦ · 15 min read · Sep 7

564 7

```
class DataPreprocessor:
    def __init__(self, data):
        self.raw_data = data
        self.cleaned_data = self.clean_data(data)

    def clean_data(self):
        # imputation, outlier treatment
        ...

    def transform_data(self):
        # transformations and encode data
        ...
```

Molly Ruby in Towards Data Science

Object Oriented Data Science: Refactoring Code



Zack Fizell in ILLUMINATION

Are you Smart Enough to Solve this 300-Year-Old Problem?

Elevating machine learning models and data science products with efficient code and...

✦ · 7 min read · Aug 24

243 4

A look into the problem that gave birth to modern optimal control theory

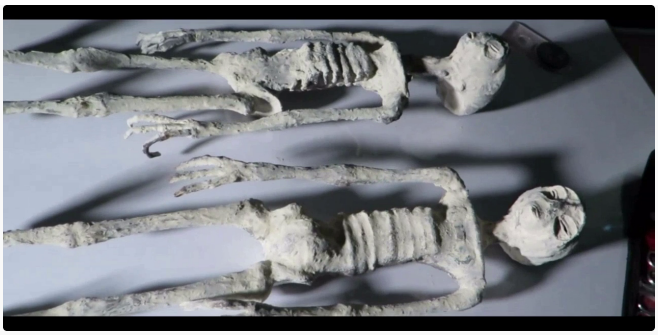
✦ · 5 min read · Aug 1, 2022

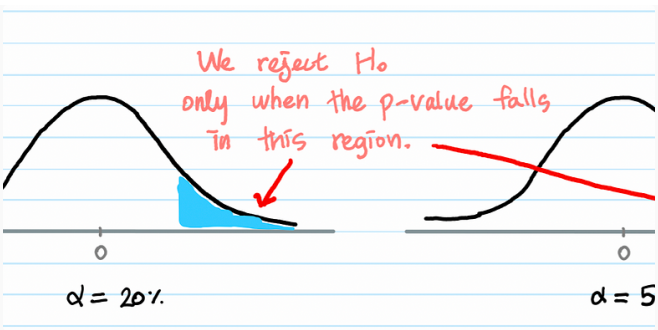
294 1

See all from Zack Fizell

See all from Towards Data Science

Recommended from Medium





alienmummy

Alien Mummies, Nazca Desert, Peru

Interview 2023 with Prof. Zuniga Aviles Roger University of San Luis Gonzaga de Ica, Peru

8 min read · Jul 13

10

Ms Aerin in IntuitionMath

Chi Square Test—Intuition, Examples, and Step-by-Step...

The best way to see if two variables are related.

✦ · 15 min read · Feb 12

377 3

Lists

Coding & Development

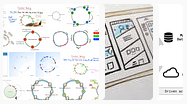
11 stories · 181 saves

ChatGPT

21 stories · 156 saves

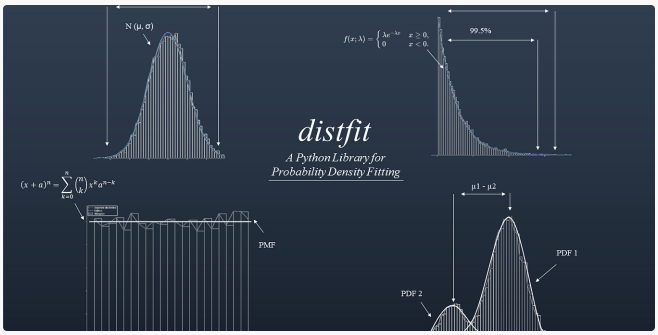
ChatGPT prompts

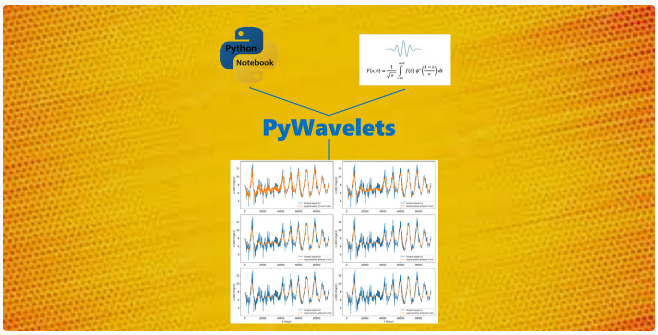
24 stories · 396 saves



General Coding Knowledge

20 stories · 353 saves





Erdogan Taskesen in Towards Data Science

How to Find the Best Theoretical Distribution for Your Data

Knowing the underlying data distribution is an essential step for data modeling and has...

· 19 min read · Feb 3

1K 10

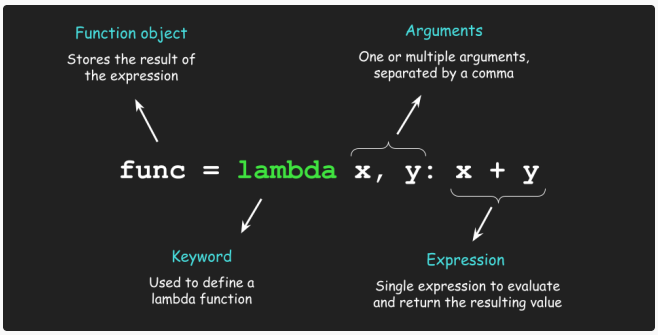
Dr. Shouke Wei

Multilevel Discrete Wavelet Transform and Noise Reduction o...

How to make multilevel Discrete Wavelet Transform and noise reduction of 1D Time...

· 8 min read · Mar 21

110 3



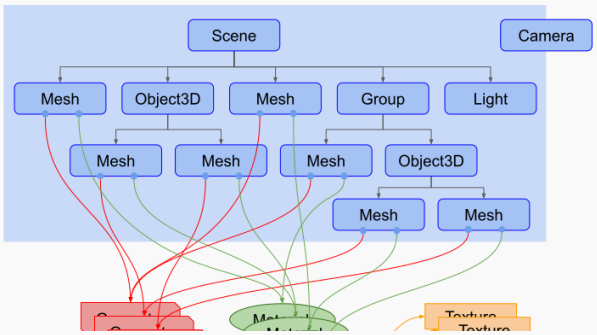
Ernest Asena

Lambda Functions in Python: Unleashing the Magic of Concise...

Welcome to a magical journey through the world of Python lambda functions, where...

4 min read · Aug 23

8



Rabin Lamichhane

Three.js

Three.js is a popular open-source JavaScript library used for creating and displaying 3D...

4 min read · Apr 27

15

See more recommendations