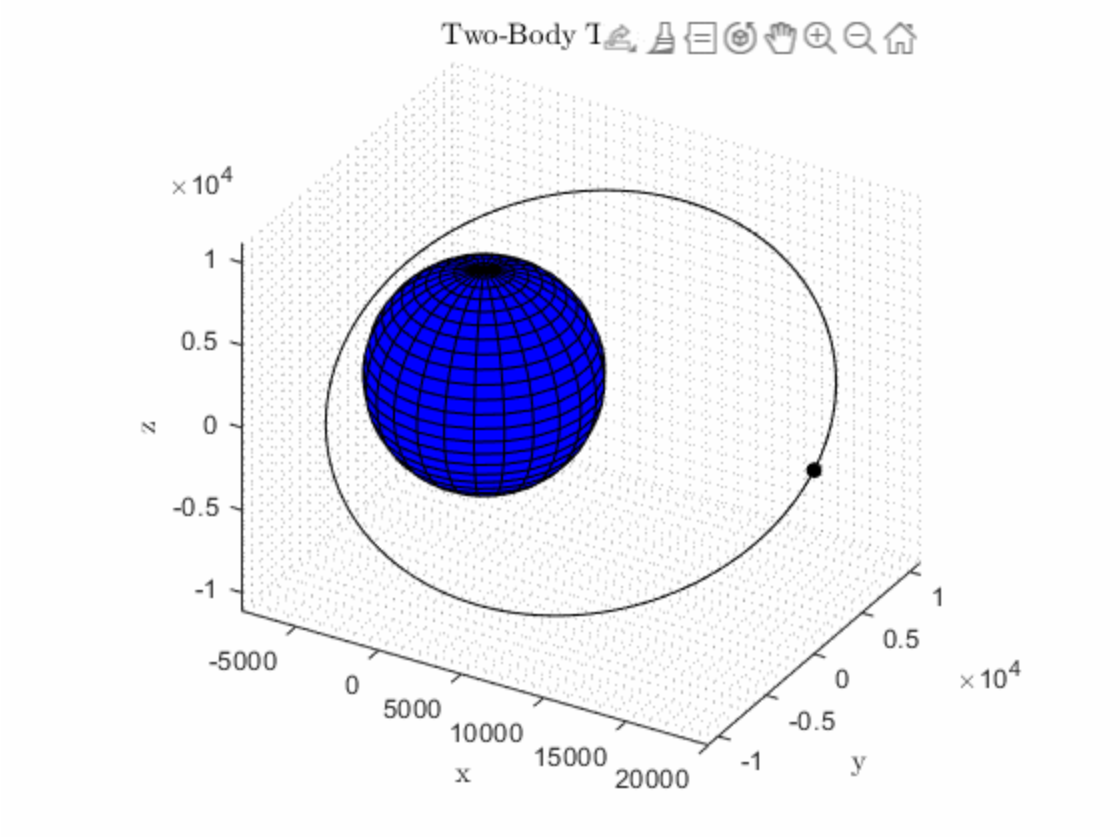
Member-only story

How to Use MATLAB to Create Two-Body Orbits

Step by step walkthrough on using MATLAB to determine how a spacecraft moves under the influence of a larger body's gravity

Zack Fizell · Following
Published in Towards Data Science · 6 min read · Jul 21, 2022



Two-Body Orbit Animated [Created by Author]

Orbital mechanics (or astrodynamics) involves applying Newton's laws of motion and the universal law of gravitation to spacecraft (such as rockets and satellites). It is used by mission planners to predict the motion of spacecraft under the influence of gravity, thrusts, and other forces. The topic of this article is the two-body problem. This is a subcategory of astrodynamics, which involves a spacecraft under the influence of a single massive body with no other forces considered. The spacecraft's mass is also considered negligible compared to the larger mass. This means the massive body will affect the motion of the spacecraft, but the spacecraft will not affect the motion of the massive body. In reality, the body of interest does not have to be a spacecraft; it could be an asteroid, comet, astronaut, etc.

. . .

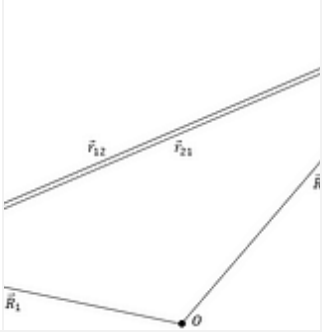
How to Solve the Two-Body Problem

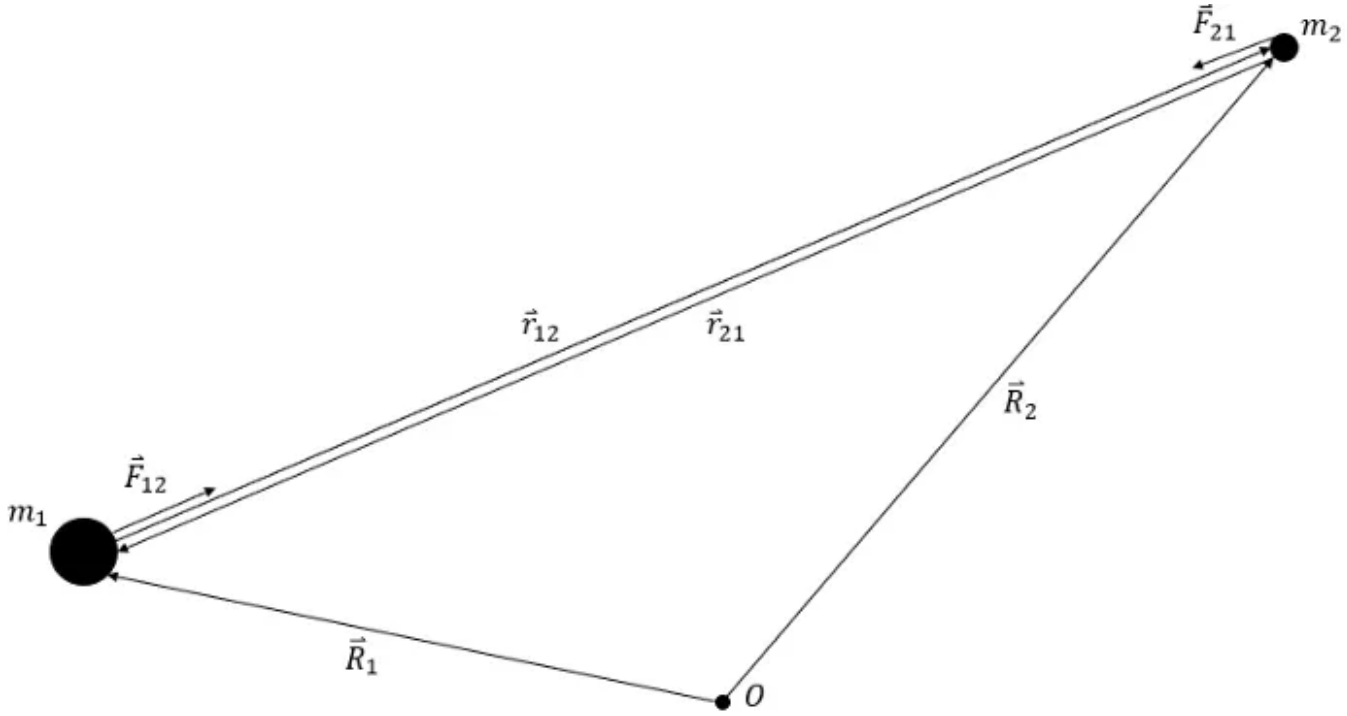
In order to generate a trajectory in a two-body system, the equations of motion for a spacecraft (mass of interest) must be derived. There are plenty of resources that show this derivation and the final equations of motion, so you do not have to do it yourself. I demonstrated this process in the below article. I encourage you to understand the equations and how they were obtained. However, if you do not want to do that and simply want to create an orbit, I have provided a diagram and the required equations of motion below for your reference.

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





Two-Body Problem Diagram [Created by Author]

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

In these equations of motion, μ is the gravitational parameter of the primary mass (planet, moon, start, etc.). x, y, and z are the Cartesian coordinates that define the spacecraft's position in inertial space. The dots represent time derivatives, so one dot is a velocity and two dots represent an acceleration. We will utilize these in the coding section of the article.

As an aside, the two-body problem is a very simplified version of the complex field of orbital mechanics. In reality, one has to account for perturbing forces such as atmospheric drag, solar radiation pressure, other gravitational forces, and effects of non-spherical bodies when designing a mission. Though not the most accurate method, the two-body problem can

be utilized for initial mission planning for spacecraft, especially if it is orbiting close to a single body. When the orbit is far away from the massive body, other forces are going to play a larger role in the motion of the spacecraft. The additional perturbing forces are not discussed here, but they are good to keep in mind.

In order to simulate an orbit, we will need to utilize numerical integration. If you do not know what this is, you will still be able to use the code to create your own orbits. In order to numerically integrate, or solve, the equations of motion for the spacecraft, we will use the `ode113` function in MATLAB. This is a built-in function that takes, as inputs, a set of derivatives to be integrated (user-defined function), a time interval of interest, initial conditions, and optionally a tolerance for the numerical integration. The set of derivatives will be the time derivatives of our state vector, *Y*, which includes the position and velocity components of the spacecraft in vector form. Both the state vector and its time derivative are shown below.

$$\vec{Y} = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \qquad \dot{\vec{Y}} = \frac{d\vec{Y}}{dt} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\, x \\ \ddot{y} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\, y \\ \ddot{z} = -\frac{\mu}{(x^2+y^2+z^2)^{3/2}}\, z \end{pmatrix}$$

That was a brief overview of the numerical integration process for the two-body problem equations of motion. Let's start the code that will simulate a trajectory for an Earth-orbiting satellite!

. . .

User-Defined ODE Function

As stated earlier, in order to create our trajectory, we will need to numerically integrate the two-body problem equations of motion. We are going to be using `ode113` to do this. This MATLAB function requires us to create our own function that will be called to do the numerical integration. We will define this function, `ODE2BP`. This function uses the Earth's gravitational parameter, `mu`, and the state vector, `Y`, from earlier to create the time derivative state vector, `dYdt`. The function returns this new vector, which will be used by `ode113` to numerically integrate. Note: in MATLAB, user-defined functions must be placed at the end of the script.

```
% User-Defined ODE Function
function dYdt = ODE2BP(t, Y)
    mu = 3.986*10^5; % Earth's gravitational parameter [km^3/s^2]
    x = Y(1); % [km]
    y = Y(2); % [km]
    z = Y(3); % [km]
    vx = Y(4); % [km/s]
    vy = Y(5); % [km/s]
    vz = Y(6); % [km/s]
    xddot = -mu/(x^2+y^2+z^2)^(3/2)*x; % [km/s^2]
    yddot = -mu/(x^2+y^2+z^2)^(3/2)*y; % [km/s^2]
    zddot = -mu/(x^2+y^2+z^2)^(3/2)*z; % [km/s^2]

    dYdt = [vx;vy;vz;xddot;yddot;zddot]; % Y'
end
```

Numerical Integration

Next, we finish defining the inputs for `ode113`. One of those inputs is the initial conditions for the spacecraft. This is in the form of the state vector from the beginning of the article. Note the components for position and velocity are in *km* and *km/s* by convention. Next, we have a time interval of interest in seconds. Finally, there are many optional arguments you can pass to `ode113`. Since orbital mechanics requires precision, I usually pass a very

small relative tolerance when numerically integrating. We then pass these inputs with the function we defined in the previous section into the ODE solver. The function outputs variables `t` and `Y`, which are the numerical integration time steps and the state vector at each time step. We can then pull the time histories for our position coordinates `x`, `y`, and `z` from the output.

```
% Creating Inputs for Numerical Integration
Y0 = [20000; 0; 0; 0; 2.9; 1.8]; % [x; y; z; vx; vy; vz] [km, km/s]
tspan = [0 24*60*60]; % One day [s]
options = odeset('RelTol', 1e-13); % Setting a tolerance

% Numerical Integration
[t, Y] = ode113(@ODE2BP, tspan, Y0, options);

% Pulling Position Data from Output
x = Y(:, 1); % [km]
y = Y(:, 2); % [km]
z = Y(:, 3); % [km]
```

Plotting Trajectory

Lastly, we can use the `x`, `y`, and `z` variables to create the trajectory of our spacecraft. We first initialize the figure with labels, a grid, and an equal set of axes (gives the true representation of the orbit in physical space). Next, we can use the `sphere` function in MATLAB to create a representation of Earth at the origin of our plot. Most importantly, the trajectory can be plotted using the `plot3` function.

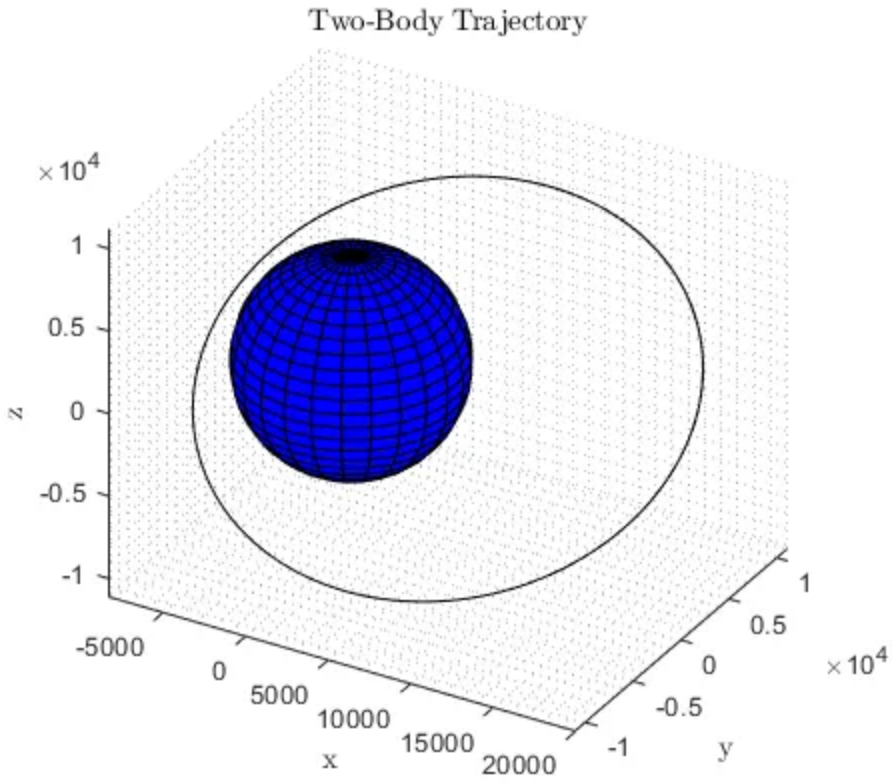
```
% Creating Figure
figure; hold on
title('Two-Body Trajectory', 'Interpreter', 'Latex')
xlabel('x', 'Interpreter', 'Latex')
ylabel('y', 'Interpreter', 'Latex')
zlabel('z', 'Interpreter', 'Latex')
axis equal
```

```
grid minor
view(30, 30)

% Creating/Plotting Spherical Earth
rm = 6378.14; % Radius of Earth [km]
[xEarth, yEarth, zEarth] = sphere(25);
surf(rm*xEarth,rm*yEarth,rm*zEarth, 'FaceColor', [0 0 1]);

% Plotting Trajectory
plot3(x, y, z, 'k')
hold off
```

The code above produces the following plot:



Two-Body Orbit [Created by Author]

That concludes the process to create a two-body orbit or trajectory with MATLAB. Toy around with the initial conditions and time interval to create your own unique orbits! Remember this is an oversimplification of most astrodynamics problems. If you want a better representation of the physics,

you will need to include perturbing forces. This can be quite the challenge, so that is why it is best to learn from the simple two-body problem and build from there. If you decide to go further, good luck on your journey!

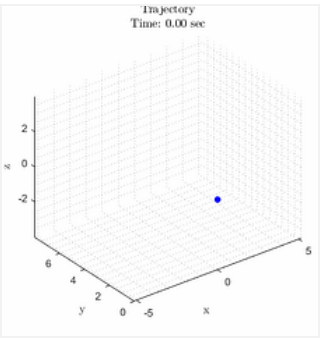
. . .

If you want to create an animation similar to the visual at the beginning of the article, check out the article below. Thank you for taking the time to read. If you found this useful, please give me a follow! Thanks!

How to Animate Plots in MATLAB

A simple method to animate data to create dynamic visuals

towardsdatascience.com



Coding Physics Space Matlab Programming

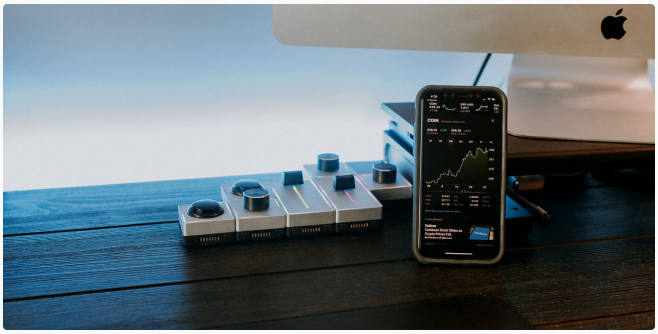
Written by Zack Fizell

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



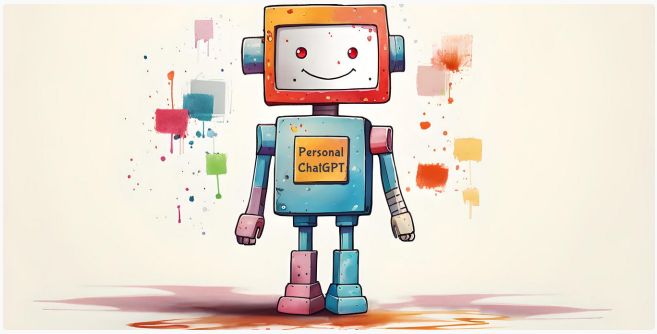
Zack Fizell in Towards Data Science

Creating Regression Models to Predict Data Responses

Learn to create and code a regression model from scratch to forecast/predict outcomes

✨ · 8 min read · Mar 4, 2022

78 1



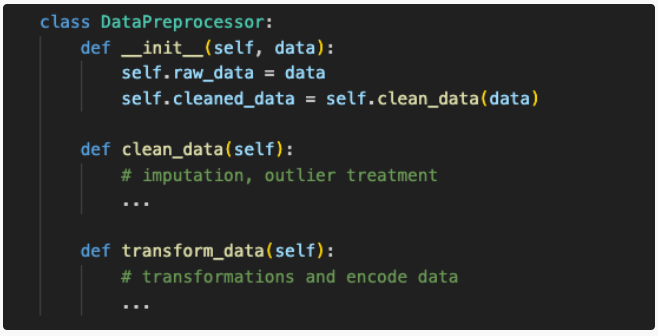
Robert A. Gonsalves in Towards Data Science

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

Zack Fizell

Object Oriented Data Science: Refactoring Code

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

✦ · 7 min read · Aug 24

243 4

10 Useful Websites You Need to Learn Python

Bonus: Some give you certificates you can use to boost your resume!

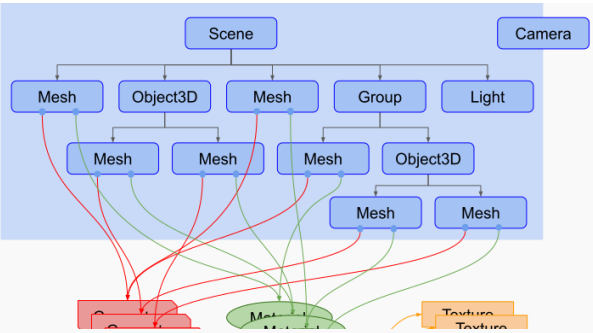
4 min read · May 12, 2022

150 1

See all from Zack Fizell

See all from Towards Data Science

Recommended from Medium



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

Hennie de Harder in Towards Data Science

An Introduction to a Powerful Optimization Technique: Simulate...

Explanation, parameters, strengths, weaknesses and use cases

· 9 min read · Mar 15

416 7

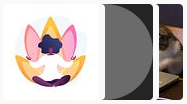
Lists

General Coding Knowledge

20 stories · 353 saves

It's never too late or early to start something

15 stories · 128 saves

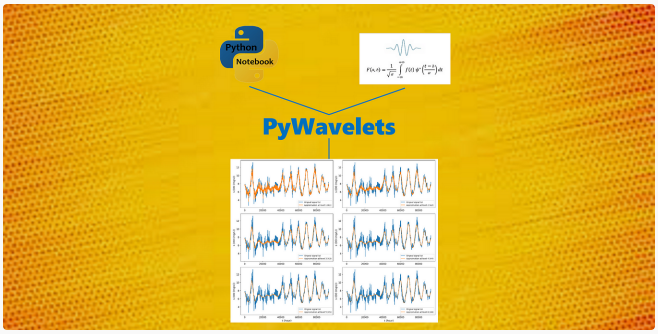


Stories to Help You Grow as a Software Developer

19 stories · 382 saves

Coding & Development

11 stories · 181 saves



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

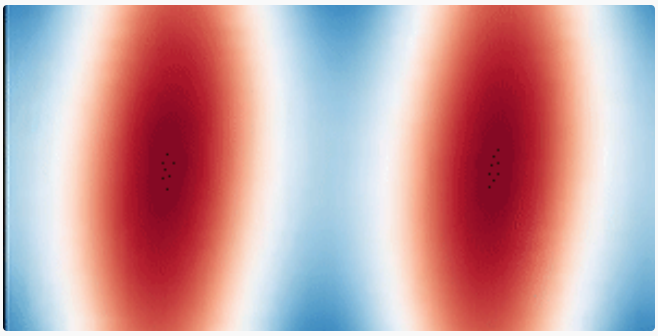
Dr. Shouke Wei

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



Philip Mocz in Level Up Coding

Create Your Own Navier-Stokes Spectral Method Fluid Simulation...

For today's recreational coding exercise, we solve the Navier-Stokes equations for an...

· 7 min read · Aug 3

634 3

Franciszek Szewczyk

Rope Simulator in C++

For starters, we're going to implement a simple rope simulator. We will be using C++,...

7 min read · Sep 11

26 1

See more recommendations