MATHEMATICAL PHYSICS

**NUMERICAL METHODS**

ADITYA SINGH 2K19/EP/005

horizontal line

# 

# Introduction

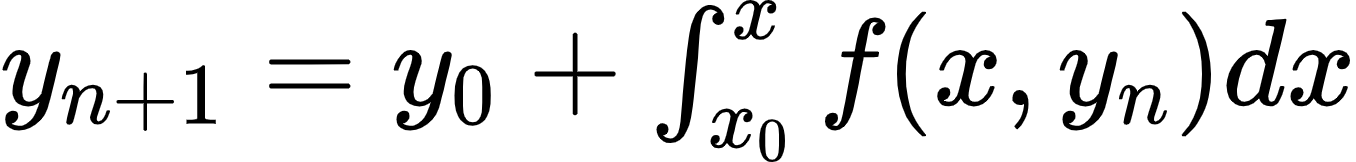
Numerical analysis, an area of mathematics and computer science that creates, analyzes, and implements algorithms for obtaining numerical solutions to problems involving continuous variables.

Many differential equations arising in applications are so complicated that it is impractical to have solution formulas.. In either situation, numerical methods provide a powerful alternative tool for solving the differential equation.

# Numerical Methods to Solve ODE

Numerical methods for solving first-order IVPs often fall into one of two large categories:linear multistep methods, or Runge–Kutta methods. A further division can be realized by dividing methods into those that are explicit and those that are implicit. For example, implicit linear multistep methods include Adams-Moulton methods, and backward differentiation methods (BDF), whereas implicit Runge–Kutta methods include diagonally implicit Runge–Kutta (DIRK), singly diagonally implicit Runge–Kutta (SDIRK),and Gauss–Radau (based on Gaussian quadrature) numerical methods. Explicit examples from the linear multistep family include the Adams–Bashforth methods, and any Runge–Kutta method with a lower diagonal Butcher tableau is explicit. A loose rule of thumb dictates that stiff differential equations require the use of implicit schemes, whereas non-stiff problems can be solved more efficiently with explicit schemes.

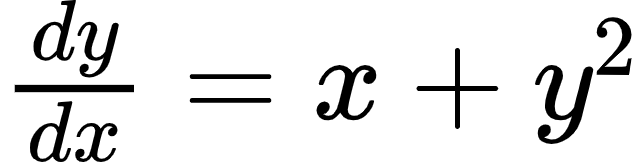
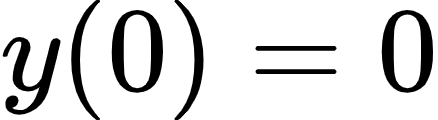
### Picard’s Method

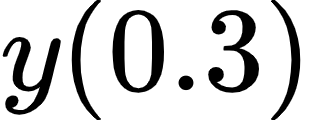


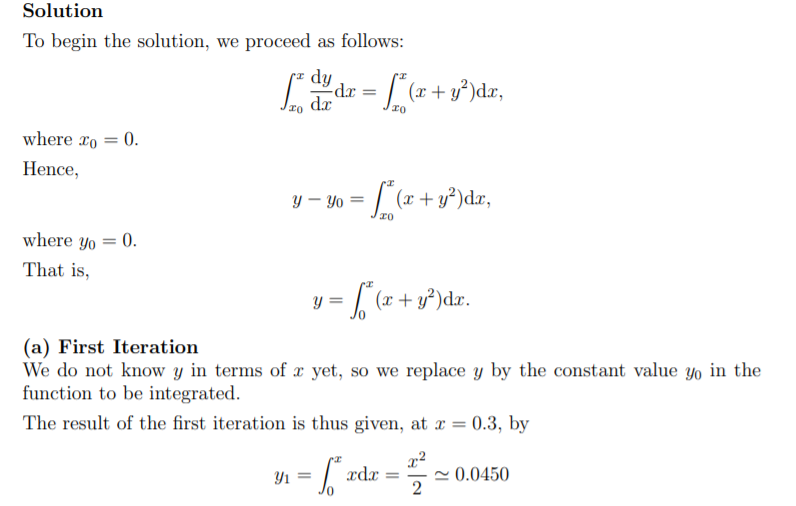
This method of solving a differential equation approximately is one of successive approximation; that is, it is an iterative method in which the numerical results become more and more accurate, the more times it is used.

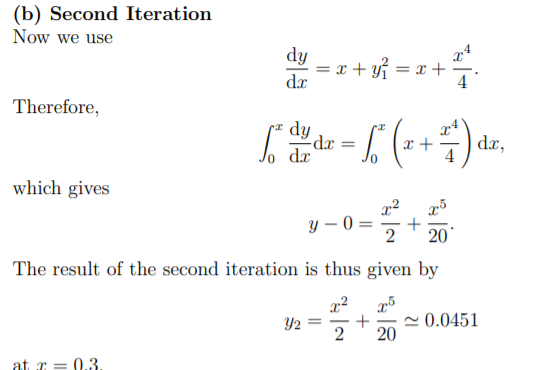
The Picard’s iterative method gives a sequence of approximations Y1(x), Y2(x), …Yk(x) to the solution of differential equations such that the nth approximation is obtained from one or more previous approximations.

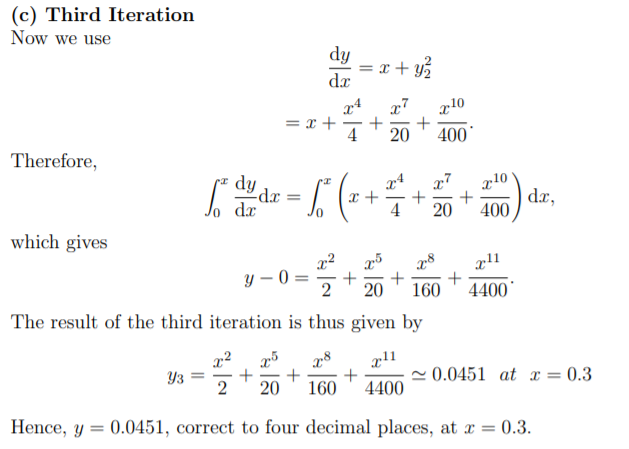
**Applying the method:**

Given, 

Solve by Picard’s Method upto third approximation and find the value of .







### Euler’s Method

The simplest numerical method for solving the initial value problem is Euler’s method.

Generate a numerical solution to an initial value problem of the form:

**y′ = f(x, y) with y(xo) = yo**

We chop this interval into small subdivisions of length h. Then, using the initial condition as our starting point, we generate the rest of the solution by using the iterative formulas:

**xn+1 = xn + h**

**yn+1 = yn + h f(xn, yn)**

to find the coordinates of the points in our numerical solution. We terminate this process when we have reached the right end of the desired interval.

**Applying the Method:**

Example : F(x, y) = x + 2y with initial condition: xo = 0, yo = 0.

Solution: We now use the Euler method formulas to generate values for x1 and y1.

x1 = xo + h => x1 = 0 + 0.25 => x1 = 0.25

y1 = yo + h f(xo, yo) => y1 = yo + h (xo + 2yo)

y1 = 0 + 0.25 (0 + 2\*0) => y1 = 0

* x1 = 0.25
* y1 = 0

We now move on to get the next point in the solution, (x2, y2).

x2 = x1 + h => x2 = 0.25 + 0.25 => x2 = 0.5

y2 = y1 + h f(x1, y1) => y2 = y1 + h (x1 + 2y1)

y2 = 0 + 0.25 (0.25 + 2\*0) => y2 = 0.0625

* x2 = 0.5
* y2 = 0.0625

We now move on to get the fourth point in the solution, (x3, y3).

x3 = x2 + h => x3 = 0.5 + 0.25 => x3 = 0.75

y3 = y2 + h f(x2, y2) => y3 = y2 + h (x2 + 2y2)

y3 = 0.0625 + 0.25 (0.5 + 2\*0.0625) => y3 = 0.21875

* x3 = 0.75
* y3 = 0.21875

We now move on to get the fifth point in the solution, (x4, y4).

x4 = x3 + h => x4 = 0.75 + 0.25 => x4 = 1

y4 = y3 + h f(x3, y3) => y4 = y3 + h (x3 + 2y3)

y4 = 0.21875 + 0.25 (0.75 + 2\*0.21875) => y4 = 0.515625

* x4 = 1
* y4 = 0.515625

We could summarize the results of all of our calculations in a tabular form, as follows:

|  |  |  |
| --- | --- | --- |
| n | xn | yn |
| 0 | 0.00 | 0.000000 |
| 1 | 0.25 | 0.000000 |
| 2 | 0.50 | 0.062500 |
| 3 | 0.75 | 0.218750 |
| 4 | 1.00 | 0.515625 |

### Runge-Kutta Method

Runge–Kutta method is an effective and widely used method for solving the initial-value problems of differential equations. Runge–Kutta method can be used to construct high order accurate numerical methods by functions' self without needing the high order derivatives of functions. In numerical analysis, the Runge–Kutta methods are a family of implicit and explicit iterative methods, which include the well-known routine called the Euler Method, used in temporal discretization for the approximate solutions of ordinary differential equations.

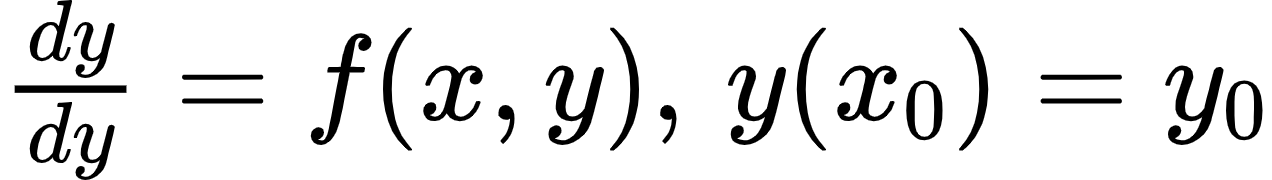
**History**

The Runge-Kutta Method was developed by two German men Carl Runge, and Martin Kutta in 1901. Carl Runge extended the approximation method of Euler to a more elaborate scheme which was capable of greater accuracy. K.W. Kutta analyzed some numerical methods and introduced amongst other methods used for the approximate solutions of ordinary differential equations. The paper, which appeared in 1901, made a complete classification of order 4 methods and introduced the famous method, known now as the classical Runge-Kutta method.

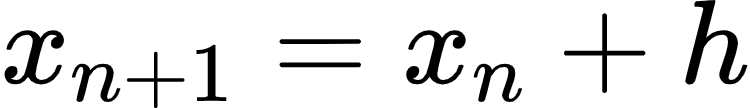
**Method**

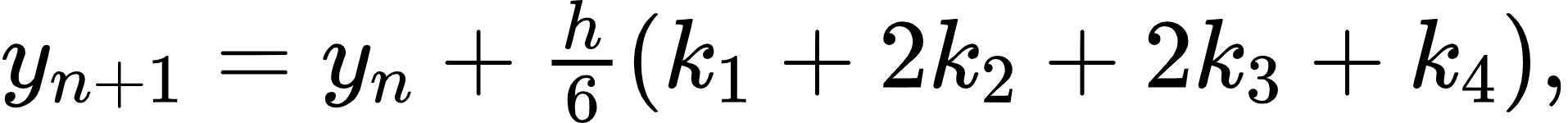
The most widely known member of the Runge–Kutta family is generally referred to as "RK4", the "classic Runge–Kutta method" or simply as "the Runge–Kutta method".

Let an initial value problem be specified as follows:

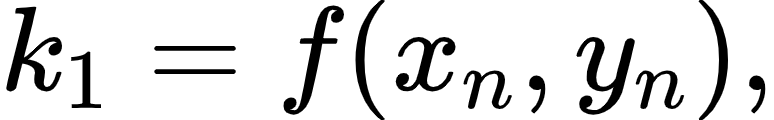


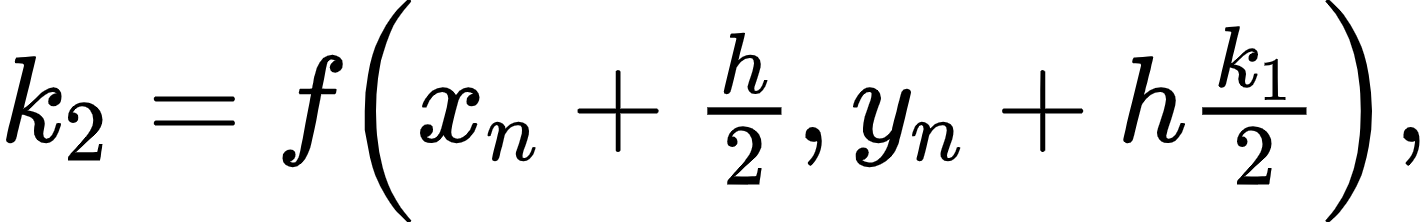
Here y is an unknown function (scalar or vector) of x, which we would like to approximate; we are told that dy/dx , the rate at which y changes, is a function of x and y. At the initial x0 the corresponding y value is y0. Now pick a step-size h > 0 and define

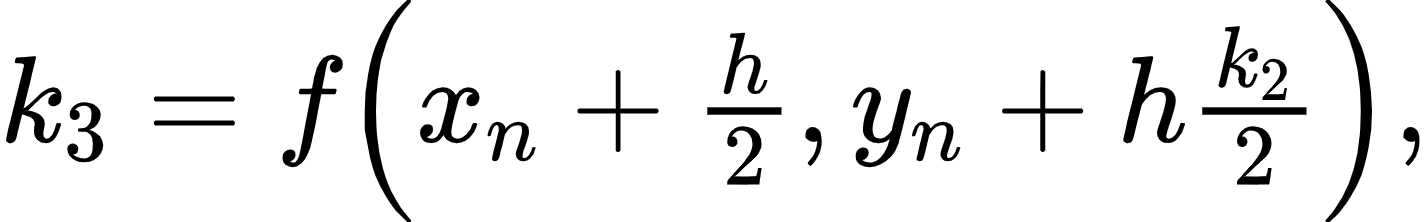


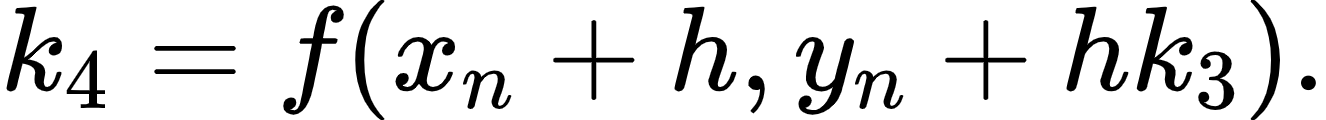


for n = 0, 1, 2, 3, ..., using









**Example**

Find y(0.2) for y′=x-y2, y(0) = 1, with step length 0.1

k1=hf(x0,y0)=(0.1)f(0,1)=(0.1)⋅(-0.5)=-0.05

k2=hf(x0+h/2,y0+k1/2)=(0.1)f(0.05,0.975)=(0.1)⋅(-0.4625)=-0.04625

k3=hf(x0+h/2,y0+k2/2)=(0.1)f(0.05,0.97688)=(0.1)⋅(-0.46344)=-0.04634

k4=hf(x0+h,y0+k3)=(0.1)f(0.1,0.95366)=(0.1)⋅(-0.42683)=-0.04268

y1=y0+16(k1+2k2+2k3+k4)

y1=1+16[-0.05+2(-0.04625)+2(-0.04634)+(-0.04268)] ∴ y(0.1)=0.95369

Again taking (x1,y1) in place of (x0,y0) repeat the process

k1=hf(x1,y1)=(0.1)f(0.1,0.95369)=(0.1)⋅(-0.42684)=-0.04268

k2=hf(x1+h/2,y1+k1/2)=(0.1)f(0.15,0.93235)=(0.1)⋅(-0.39117)=-0.03912

k3=hf(x1+h/2,y1+k2/2)=(0.1)f(0.15,0.93413)=(0.1)⋅(-0.39206)=-0.03921

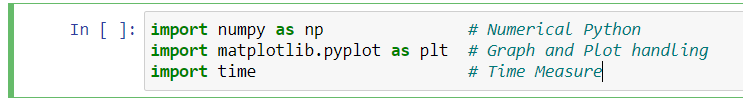
k4=hf(x1+h,y1+k3)=(0.1)f(0.2,0.91448)=(0.1)⋅(-0.35724)=-0.03572

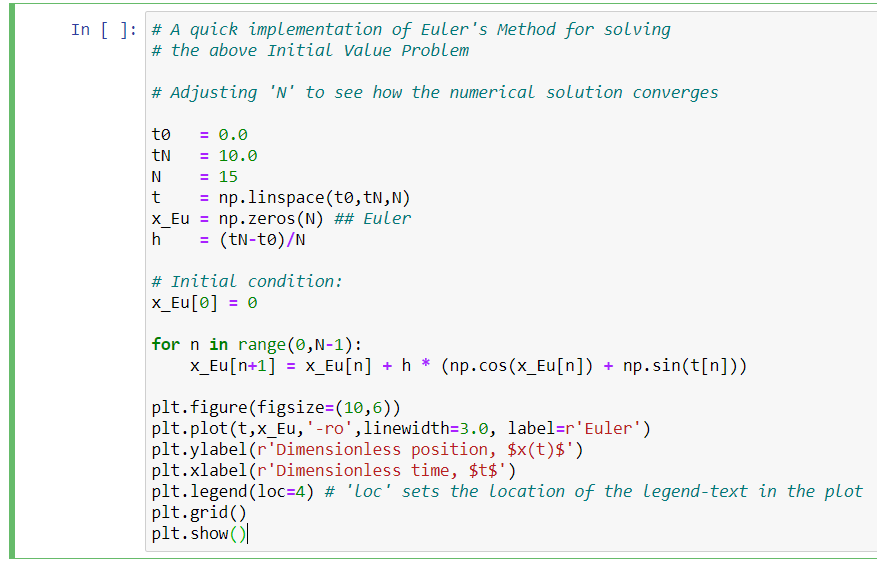
y2=y1+16(k1+2k2+2k3+k4)

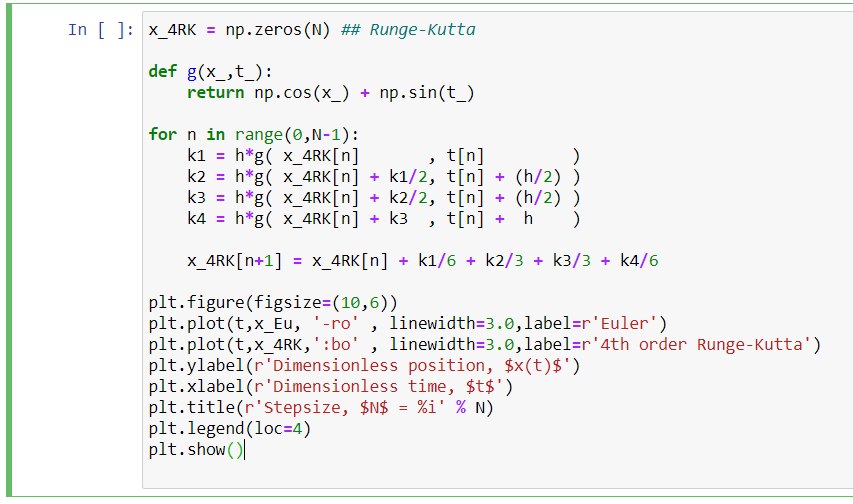
y2=0.95369+16[-0.04268+2(-0.03912)+2(-0.03921)+(-0.03572)] ∴ y2=0.91451

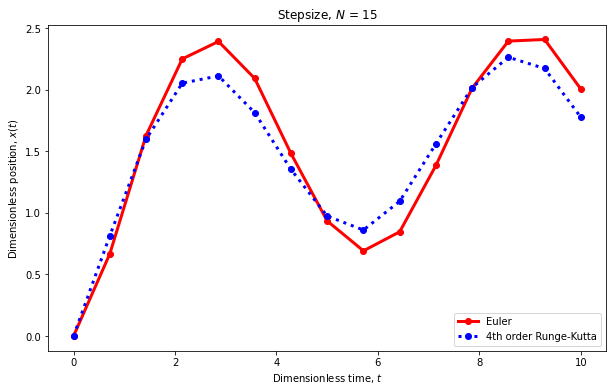
### Comparison Euler vs Runge-Kutta

I have used Python math libraries (numpy and matplotlib) to find solutions for the same First Order Differential Equation by Euler and Runge- Kutta Method. [Link](https://github.com/adityanjr/RK4-Euler/blob/master/RK4_Euler.ipynb)









### Orbiting mass around a central mass(satellite and earth)

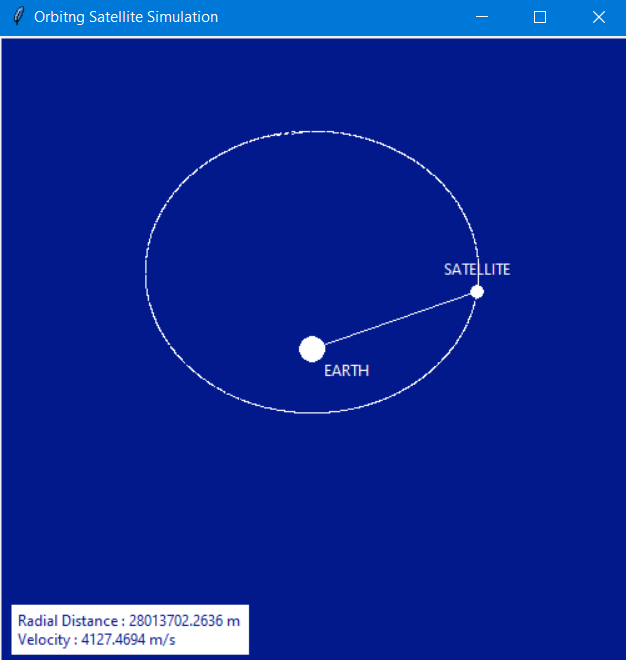
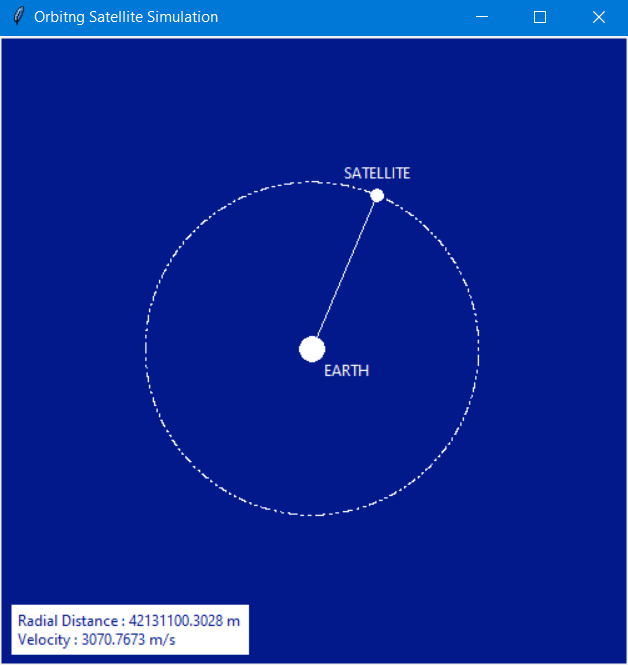
The initial conditions of the satellite are those found for a geostationary orbit.

Eccentricity, e= (1/GM) [(v2 -(GM/r)r-(r.v)v] with v = dr/dt.

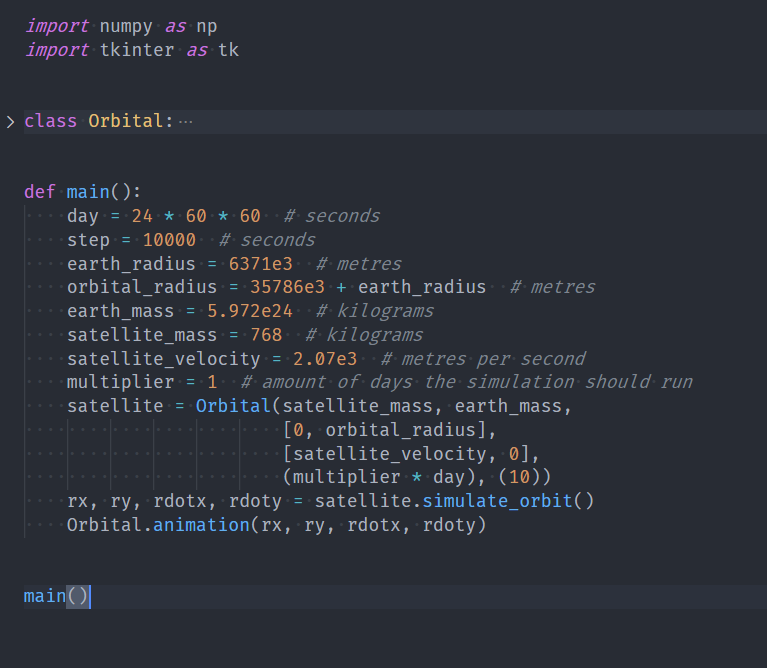
With an initial position above the Earth's surface of 35786 kilometres and an initial velocity of 3.07 kilometres per second resulting in a circular orbit.

The period of the simulation is of geostationary takes around that much time for single orbit.

The relativistic solutions can be used as the foundation for solving relativistic targeting problems that will be required by future spacecraft traveling with velocities close to the speed of light and/or orbiting in intense gravitational fields such as those that exist near black holes.



Python Code To Stimulate Orbit



The Orbital Class includes the library function which is pre-built to calculate the eccentricity of the orbit and gives the corresponding location and velocity of the satellite.

The Library Function uses several numerical methods including Runge-Kutta Method internally.

**THANK YOU**