Initial Value Problems

Ezra Harris

December 7, 2021

Abstract

When approximating an initial value problem (IVP) of an ordinary differential equation (ODEs), performing a high order Runge-Kutta Order Four Method allows us to achieve high-accuracy approximations of the differential equation. This can be applied to such ODEs as the orbiting behavior of commits within the solar system and allow scientists to make accurate predictions on their velocity. But these expansions still can not reach exact machine precision due to round off error.

1 Introduction

Differential equations are utilized in science, engineering, and mathematics to model problems that involve the change of some variable with respect to another variable. Differential equations that satisfies a given initial condition are known as initial-value problems, and can be solved using specified and equally spaced points along the curve. These initial-value problems have three theories in which need to be satisfied in order for them to be properly approximated at a unique solution:

1. A function $f(t, y)$ is said to satisfy a Lipschitz condition in the variable y on a set $D \subset R^2$ if a constant $L > 0$ exists with $|f(t, y_1) - f(t, y_2,)| <= L|y_1 - y_2|$
2. A set $D \subset R^2$ is said to be convex if whenever (t_1, y_1) and (t_2, y_2) belong to D, then $((1 - \lambda)t_1 + \lambda t_2, (1 - \lambda)y_1 + \lambda y_2)$ also belongs to D for every λ in [0,1].
3. Suppose $f(t, y)$ is defined on a convex set $D \subset R^2$ If a constant $L > 0$ exists with $|\frac{\partial f}{\partial y}(t, y)| <= L$, for all $(t, y) \in D$ then f satisfies a Lipschitz condition on Din the variable y with a Lipschitz constant L

With those theories fulfilled the next aspect of a proper initial-value problem is that when there is small changes in the statement of the problem, there is only small changes in the solution in order to reduce the effects of round off error. In order to do this the initial value problem must:

1. Have a unique solution, $y(t)$, to the problem and
2. Have existing constants $\varepsilon_0 > 0$ and $k > 0$ such that for any ε. in $(0, \varepsilon)$, whenever $\partial(t)$ is continuous with $|\partial(t)| < \varepsilon$ for all t in [a,b]

From there, approximation techniques such as High Order Runge-Kutta Methods can be used to solve for initial-value problems such as the modeling of cometary orbits. Comets, which spend most of their life time far out in the solar system, move very slow, but can on rare occasions have their orbits bring them in towards the sun, where they move extremely fast. This relationship is between the Sun, with mass M at the origin, and a commit of mass m with position vector r is GMm/r^2 in direction $-r/r$, where r is $\sqrt{x^2 + y^2}$. The two equations are:

$$\frac{d^{(2)}x}{dt^{(2)}} = -GM\frac{x}{r^3};\ r = \sqrt{x^2 + y^2} \tag{1}$$

$$\frac{d^{(2)}y}{dt^{(2)}} = -GM\frac{y}{r^3};\ r = \sqrt{x^2 + y^2} \tag{2}$$

2 Method

Runge-Kutta methods have the high-order local truncation error of the Taylor methods but eliminate the need to compute and evaluate the derivatives of $f(t, y)$. The objective of Runge-Kutta method is to obtain an approximation of the well-posed initial value problem and begins with determining the values for a_1,α_1, an β_1 with the property that $a_1 f(t + \alpha_1, y + \beta_1)$approximates[1].:

$$T^{(2)}(t, y) = f(t, y) + \frac{h}{2} f'(t, y) \tag{3}$$

with error no greater than $O(h^2)$, which is the same as the order of the local truncation of error for the Taylor method of Order two. Similarly to Taylor Series Expansions, Runge-Kutta methods can be manipulated to get higher orders of error. The term $T^{(3)}(t, y)$ can be approximated with error $0(h^3)$ by an expression of the form:

$$f(t + \alpha_1, y + \partial_1 f(t + \alpha_2, y + \partial_2 f(t, y))) \tag{4}$$

But Runge-Kutta methods of order three are not generally used. Instead, the most common Runge-Kutta method in use is of order four in difference-equation form:

$$\mathrm{w}_0 = \alpha,$$
$$\mathrm{k}_1 = hf(t_i, w_i),$$
$$\mathrm{k}_2 = hf(t_i + \tfrac{h}{2}, w_i + \tfrac{1}{2}k_1),$$
$$\mathrm{k}_3 = hf(t_i + \tfrac{h}{2}, w_i + \tfrac{1}{2}k_2),$$
$$\mathrm{k}_4 = hf(t_{i+1}, w_i + k_3),$$
$$\mathrm{w}_{i+1} = w_i + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

for each i = 0,1,..,N-1. This Runge-Kutta Order Four has local truncation error $0(h^4)$, provided the solution y(t) has five continuous derivatives.

3 Analysis and Results

To solve IVP we must first find our auxiliary variable for the second derivative in order to cast our functions to first order derivative. Suppose that:

$$\tfrac{dx}{dt} = v_x$$
$$\tfrac{dy}{dt} = v_y$$

Therefore,

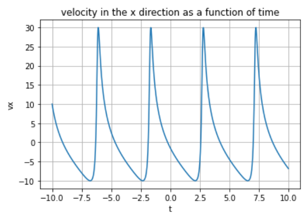
$$\tfrac{dv_x}{dt} = -GM\tfrac{x}{r^3} \;\; = f(x, y, \mathrm{vx}, x)$$
$$\tfrac{dv_y}{dt} = -GM\tfrac{y}{r^3} = f(x, y, \mathrm{vy}, x)$$

Finally we cast $\frac{dx}{dt}$ and $\frac{dy}{dt}$ as a single ODE of r. We also cast $\frac{dv_x}{dt}$ and $\frac{dv_y}{dt}$ as a single ODE of s. Where $\frac{d\vec{r}}{dt} = \vec{s}$ and $\frac{ds}{dt} = f(\vec{r}, \vec{s}, t)$. We then used these ODEs to code $\vec{r}$ as seen in appendix A. Once the vector is coded up creating the function for RK4 and RK4 integrate was a matter of using our original knowledge that rk1 is $k1 = h * f(r)$. [2] From there we used RK4 to integrate $f(r)$ at $y0$ to tspan as seen in Appendix A. We then used our integrative function to find the velocity in the y and x direction as a function, velocity in the y and x direction function of x.

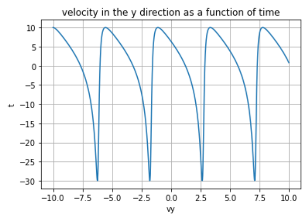
Using this method we were able to simultaneously solve the system of equation for the velocity in the x and y directions as well as the position data. The differential equations do not depend on time, nor mass of the comet, only on the relative position of the comet. It is possible to make a more physically accurate model by accounting for the effect the comet's mass on the sun making this a two body problem but for the purposes of this exercise this is not necessary.

In order to achieve the desired results the initial conditions had to become nonphysical. We set the gravitational constant G equal to 10 and the mass of the sun equal to 200. The initial conditions for the elliptic orbit are$x = 10$, $y = 10$, $v_x = 0$, $v_y = 10$. The initial conditions for the parabolic orbit are $x = 10$, $y = 0$, $v_x = 0$, $v_y = 14$ We did this in order to achieve the desired initial conditions and parameters for an elliptic orbit. Physical parameters proved to be very unwieldy for achieving the desired orbits and the comet would typically be kicked into a parabolic orbit. Finally in order to successful solve the system we had to use an array splicing method found in the function RK4 integrate in Appendix A. [3]

These are the graphs for an elliptic orbit:



(a) Vx(t)



(b) Vy(t)

Figure 1: Velocity

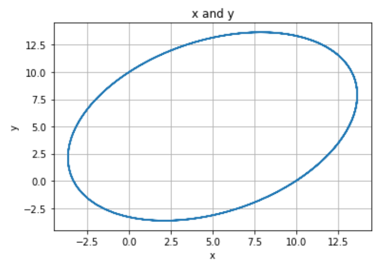
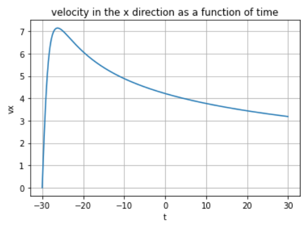
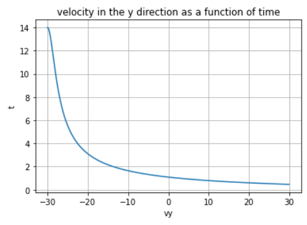


Figure 2: Position x and y elliptical

These are the graphs of a parabolic orbit:



(a) Vx(t)



(b) Vy(t)

Figure 3: Velocity

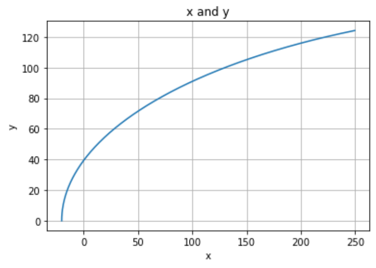


Figure 4: Position x and y parabolic

As we can see a larger velocity creates a parabolic orbit. Additionally from the graps we can see the periodic nature of the velocity in the elliptic orbit.

4 Conclusions

In this project we used the Runge-Kutta 4(RK4) method to evaluate a system of ODE's. In order to do this we cast two second order equations as a system of four first order equations. In the code we implemented the RK4 method using a vector that allowed for the simultaneous solution of the four differential equations. Finally in order to successful compile the solution we implemented an array splicing method to fill four arrays with our desired values. Through this analysis we discovered conditions for a parabolic orbit vs an elliptic orbit. Namely increasing initial velocity. Future work will be making the parameters physically accurate and finding the range of velocities for a parabolic vs elliptical orbits.

Appendix A

This is the Python code used

```
    # -*- coding: utf-8 -*-
"""
Created on Fri Dec  3 22:09:43 2021

@author: Ezra
"""

import numpy as np
import matplotlib.pyplot as plt

G = 10
M = 200

def f(r):
    x,y,vx,vy = r
    s = np.hypot(x,y)
    dvxdt = -G*M*x/s**3
    dvydt = -G*M*y/s**3
    return np.array([vx,vy,dvxdt,dvydt],float)

a = -10.0
b = 10.0
N = 1000.0
h = (b-a)/N

t = np.arange(a,b,h)

y0 = np.array([10,0,10,10],float)

def RK4(f,r,h):
    k1 = h*f(r)
    k2 = h*f(r+.5*k1)
    k3 = h*f(r+.5*k2)
    k4 = h*f(r+k3)
    return r + (k1+2*k2+2*k3+k4)/6

def RK4integrate(f, y0, tspan):
    u = np.zeros([len(tspan),len(y0)])
    u[0,:]=y0
    for k in range(1, len(tspan)):
        u[k,:] = RK4(f, u[k-1], tspan[k]-tspan[k-1])
    return u


sol_RK4 = RK4integrate(f, y0, t)
x,y,vx,vy = sol_RK4.T

plt.figure(0)
plt.plot(x,y)
plt.title("x and y")
plt.grid(True)

plt.figure(1)
plt.plot(t,y)
plt.title("y as a function of time")
plt.grid(True)

plt.figure(2)
plt.plot(t,x)
plt.title("x as a function of time")
plt.grid(True)

plt.figure(3)
plt.plot(t,vy)
plt.title("velocity in the y direction as a function of time")
plt.grid(True)

plt.figure(4)
plt.plot(t,vx)
plt.title("velocity in the x direction as a function of time")
plt.grid(True)

plt.figure(5)
plt.plot(x,vx)
plt.title("velocity in the x direction as a function of x")
plt.grid(True)

plt.figure(6)
plt.plot(y,vy)
plt.title("velocity in the y direction as a function of y")
plt.grid(True)
```

Works Cited

[1] Richard L. Burden, Douglas J Faires, and Annette M. Burden. *Numerical Analysis.* Cengage Learning, Boston, MA, 2016.

[2] Mark Newman. *Computational Physics.* University of Michigan, Ann Arbor, MI, 2012.

[3] Lutz Lehmann. Cannot get rk4 to solve for position of orbiting body in python. https://stackoverflow.com/questions/53645649/cannot-get-rk4-to-solve-for-position-of-orbiting-body-in-python, 2018. Online - Accessed 5 Dec 2021. Provided RK4Integrate Function/Array splicing method that made the code work. Used there G and M parameters.