P346 : DIY Project
Orbits with Adaptive Step Size RK4

Abhinav Roy
Roll Number 2011003
SPS, Batch 20

Ordinary differential equations (ODEs) can be used to model a wide variety of fascinating and complex physical systems. Thus the challenge of modelling the physical system can be reduced to solving a set of coupled first-order differential equations since it is always possible to convert an ordinary differential equation into a set of these equations. In this project, we aim to code the numerical integrator with adaptive step-size RKF45 using the Cash-Karp parameters in order to model our physical system, the motion of a planet around a sun.

I. Introduction

In astrodynamics, the trajectory states can be determined over time by propagating them by use of numerical integrators. The numerical integrators approximate a solution to a nonlinear ODE initial value problem. As an example consider a general second order equation,

$$p(x)\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x)$$

This can be rewritten as,

$$\frac{dy}{dx} = z(x)$$

$$\frac{dz}{dx} = \frac{r(x) - q(x)z(x)}{p(x)}$$

Therefore, the general set of equations to solve will be

$$\frac{dy_i(x)}{dx} = f_i(x, y_1, ...y_i, ...y_N)$$

where i runs from 1 to N.

In the model presented here, we used the motion of a body around another as our physical system. We assume an initial state and evolve it forward in time until a specified final time. In this example, Newton's equation of motion is

$$\frac{d^2r}{dt^2} = -\frac{GM}{r^2}$$

Rewriting them and working in the x, y and z components, we obtain

$$\frac{dx}{dt} = v_x(t) \; ; \quad \frac{dv_x}{dt} = -\frac{GMx}{r^3}$$

$$\frac{dy}{dt} = v_y(t) \; ; \quad \frac{dv_y}{dt} = -\frac{GMy}{r^3}$$

$$\frac{dz}{dt} = v_z(t) \; ; \quad \frac{dv_z}{dt} = -\frac{GMz}{r^3}$$

where,

$$r = \sqrt{x^2 + y^2 + z^2}$$

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.

$$\begin{Bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{Bmatrix} = \begin{Bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{Bmatrix}$$

This state time derivative will be made into a function that will be numerically integrated. For the integration, we will write a fourth-order Runge-Kutta with variable step size and a fourth-order Runge-Kutta with fixed step size. The methods will be fed the same initial conditions. The number of steps taken as well as the time for each method will be then compared. The Runge-Kutta methods are described below along with how they are implemented in the code.

II. Method

The first method implemented is the fourth-order Runge-Kutta method with fixed step size, which is a method that allows us to solve complicated ODE with relative fast computation and with high accuracy. 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 method by functions' self without needing the high order derivatives of functions. The form for the fourth-order Runge-Kutta for any variable y_i will be

$$\begin{aligned} k1_i &= \Delta t\ f_i(x, y_1, ...y_i, ...y_n) \\ k2_i &= \Delta t\ f_i(x + \frac{\Delta t}{2}, y_1 + \frac{k1_1}{2}, ...y_i + \frac{k1_i}{2}, ...y_n + \frac{k1_n}{2}) \\ k3_i &= \Delta t\ f_i(x + \frac{\Delta t}{2}, y_1 + \frac{k2_1}{2}, ...y_i + \frac{k2_i}{2}, ...y_n + \frac{k2_n}{2}) \\ k4_i &= \Delta t\ f_i(x + \Delta t, y_1 + k3_1, ...y_i + k3_i, ...y_n + k3_n) \\ y_i &+= \frac{k1_i}{6} + \frac{k2_i}{3} + \frac{k3_i}{3} + \frac{k4_i}{6} \end{aligned}$$

This is represented graphically in Figure 1.

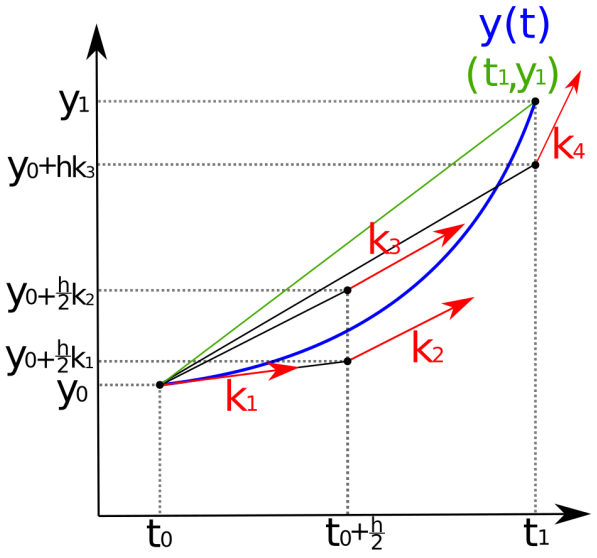

Figure 1. The fourth-order Runge-Kutta evaluates the derivative four times over a given time step. By choosing the weights for each derivative, lower orders of error can be canceled out.

In this method, a fifth-order Runge-Kutta that requires six evaluations of the function is also used to calculate a fourth order Runge-Kutta with the same six evaluations, but in a different combination. The form for the fifth order Runge-Kutta for any variable y_i is

$$k1_i = \Delta t\ f_i(x, y_1, ...y_i, ...y_n)$$

$$k2_i = \Delta t\ f_i(x + a_2\Delta t,\ ...y_i + b_{21}k1_i, ...)$$

$$...$$

$$k6_i = \Delta t\ f_i(x + a_6\Delta t, y_i + b_{61}k1_i + b_{62}k2_i + b_{63}k3_i + b_{64}k4_i + b_{65}k5_i, ...)$$

$$y_i += c_1k1_i + c_2k2_i + c_3k3_i + c_4k4_i + c_5k5_i + c_6k6_i$$

The embedded fourth-order formula is

$$y*_i += c*_1\ k1_i + c*_2\ k2_i + c*_3\ k3_i + c*_4\ k4_i + c*_5\ k5_i + c*_6\ k6_i$$

The difference between these two answers gives an estimate of the error. The coefficients are listed in Table 1.

Table 1. List of the coefficients for the adaptive Runge-Kutta method

Cash-Karp Parameters for Runge-Kutta Method								
i	a_i	b_{ij}					c_i	$c*_i$
1							37/378	2825/27648
2	1/5	1/5					0	0
3	3/10	3/40	9/40				250/621	18575/48384
4	3/5	3/10	-9/10	6/5			125/594	13525/55296
5	1	-11/54	5/2	-70/27	35/27		0	277/14336
6	7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	512/1771	1/4
	j =	1	2	3	4	5		

Once the error $\Delta = y_{n+1} - y*_{n+1}$ has been calculated, the step is rescaled. The relation between Δ_1 corresponding to a step size Δt_1 and Δ_2 corresponding to a step size Δt_2 is given by

$$\Delta t_1 = \Delta t_2 \left|\frac{\Delta_1}{\Delta_2}\right|^{0.2}$$

The time steps are then adjusted to maintain Δ within a prescribed level of accuracy. Smaller step size is used regions where the solution is tricky and error's magnitude is low, but one has to be careful because if smaller steps are used, then more steps are needed and hence the error will pile up. While a bigger step size could miss an important detail. Therefore, the adaptive step is a game of balancing between accuracy and time of computing. This in practise means that calculating the error on the steps and compare that error to desire accuracy and either increase or decrease the step to achieve the desired accuracy.

III. Pseudo-Code

```
In [ ]:

#Input Initial State of Body: state = [X, Y, Z, VX, VY, VZ]
#Define Function: func = d(state)/dt

#Define 1 step for RK4 using Cash-Karp parameters:
    #Input: Function, State, Time, Timestep
    #Output: New State, New State*, New Time

#Define Adaptive Orbital Code:
    #Input: Function, Initial State(s0), Initial Time(t0), Final Time(tf), Initial Timestep(dt0)

    #Working:
    #While time < final time:
        #Calculate 1 Step of RK4 for Initial State s0 or final array's last state
        #Calculate error between s1 and s1*
            #error in step = sqrt((x-x*)^2 + (y-y*)^2 + (z-z*)^2)
        #Variable rho is calculated
            #rho = specified error / error in step
        #If rho > 1, error in step is smaller, the accuracy is fine
            #accept s1; append to final array
        #If rho < 1, error in step is greater, the accuracy is not fine
            #don't accept s1; recalculate s1 for new dt
        #Recalculate time step as dt = dt*(rho)^0.2
            #rho > 1, time step increases
            #rho < 1, time step decreases

    #Output: Final Array of States
```

Figure 2. The pseudo-code for the solution of the adaptive step size orbital motion in multiple dimensions.