Free Fall With Drag, A Study of Numerical Solutions

1 Introduction

Numerical solutions for ordinary differential equations (ODE) are methods used to find numerical approximations to the solutions of the ODE at a given point or at given points. I will be addressing the ODE:

$$\begin{cases} \frac{dv}{dt} = g - \frac{c_d}{m}v^2 \\ v(0) = 0 \end{cases} \tag{1}$$

For this experiment time = 15s, g = 9.81 m/s^2, $m = 75kg, c_d = .25kg/m$.

The exact solution of the ODE above is given by:

$$v(t) = \sqrt{\frac{gm}{c_d}}tanh(\sqrt{\frac{gc_d}{m}}t) \tag{2}$$

I will be analyzing the accuracy of numeric solutions.

2 Algorithm

I analyzed the accuracy of three different methods of obtaining a numerical solution to the ODE mentioned in the introduction. We will go over the algorithms for

1. Euler's Method
2. Second Order Runge-Kutta Method
3. Fourth Order Runge-Kutta Method

For all the following algorithms:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y_0 = y(t_0) \end{cases} \tag{3}$$

where y(t0) is the initial condition and is known.

2.1 Using Euler's Method

The Euler Method when used to calculate a numerical solution goes like this:

1. For a given step size, h,

$$y(t + h) = y(t) + h \cdot f(t, y(t)) \tag{4}$$

2. Continue performing this calculation until

   $$t + h = t_f \tag{5}$$

   where

   $$t_f \tag{6}$$

   is the point at which you want to figure out the solution to the ODE

3. $$y(t_f) \tag{7}$$

   is the numerical solution to the ODE using Euler's Method

2.2 Second Order Runge-Kutta Method (RK2)

The Second Order Runge-Kutta Method when used to calculate a numerical solution goes like this:

1. For a given step size, h,

   $$y(t + h) = y(t) + k_2 \tag{8}$$

   $$k_1 = h \cdot f(t, y(t)) \tag{9}$$

   $$k_2 = h \cdot f(t + h/2, y(t) + k_1/2) \tag{10}$$

2. Continue performing this calculation until

   $$t + h = t_f \tag{11}$$

   where

   $$t_f \tag{12}$$

   is the point at which you want to figure out the solution to the ODE

3. $$y(t_f) \tag{13}$$

   is the numerical solution to the ODE using Euler's Method

2.3 Second Order Runge-Kutta Method (RK4)

The Fourth Order Runge-Kutta Method when used to calculate a numerical solution goes like this:

1. For a given step size, h,

$$y(t+h) = y(t) + (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4)/6 \tag{14}$$

$$k_1 = h \cdot f(t, y(t)) \tag{15}$$

$$k_2 = h \cdot f(t + h/2, y(t) + k_1/2) \tag{16}$$

$$k_3 = h \cdot f(t + h/2, y(t) + k_2/2) \tag{17}$$

$$k_4 = h \cdot f(t + h, y(t) + k_3) \tag{18}$$

2. Continue performing this calculation until

$$t + h = t_f \tag{19}$$

where

$$t_f \tag{20}$$

is the point at which you want to figure out the solution to the ODE

3. $$y(t_f) \tag{21}$$

is the numerical solution to the ODE using Euler's Method

2.4 My Code

In my code I followed these algorithms exactly. I used a variable called `time step` in place of h, a variable called `velocity` instead of y(t), and a function called `acceleration` to solve for f(t, y(t)). In order to step through until the end time of 15s, I used a `for` loop that started at h and added h + t until it reached 15. Because of the way my for loop is set up, the end time must be divisible by the time step or the approximation will be off.

3 Results

In `MatLab`, my simulation code is invoked at the prompt by calling the function `FallingObjects`. The command

```
>> FallingObjects
```

will display 3 plots on the screen and four tables below the prompt. The plots consist of:

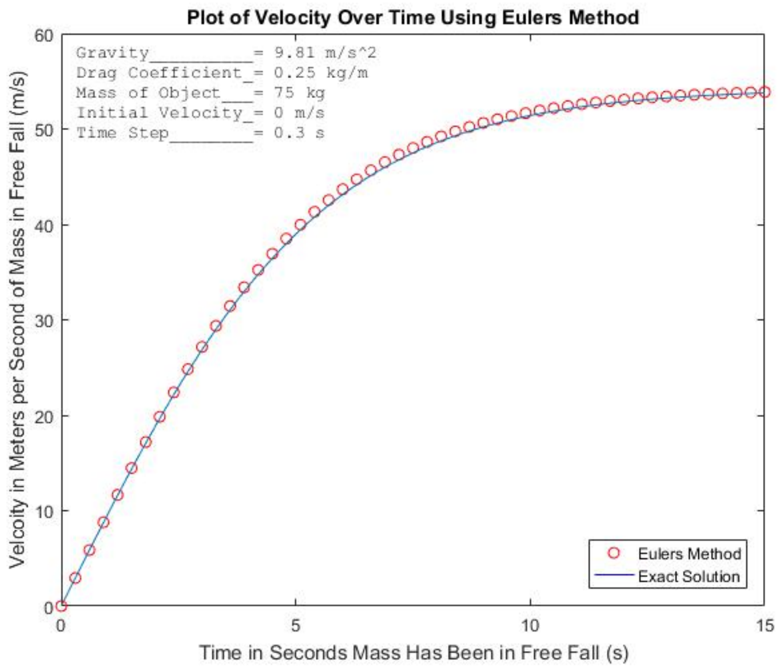
1. A plot of velocity vs time containing the exact velocity as a blue line and numerical approximations found using Euler's Method as open red circles

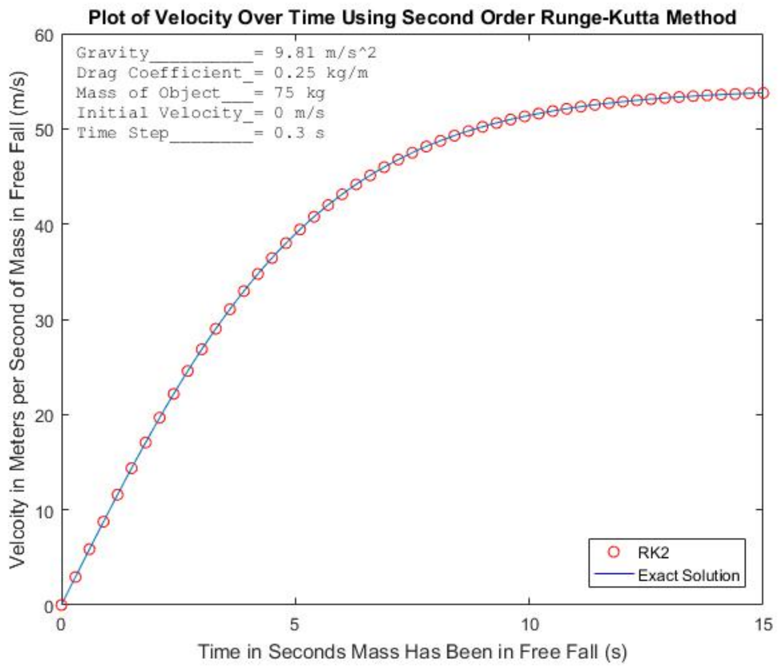
2. A plot of velocity vs time containing the exact velocity as a blue line and numerical approximations found using RK2 Method as open red circles
3. A plot of velocity vs time containing the exact velocity as a blue line and numerical approximations found using RK4 Method as open red circles

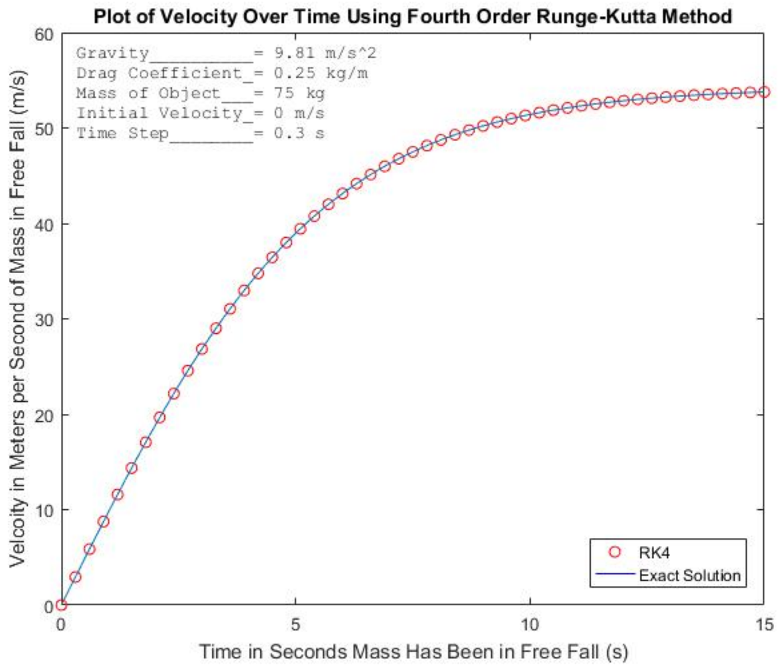
The tables will consist of:

1. Table with five columns containing step size, approximations found with Euler's Method using those step sizes, the exact solution, the absolute error of each approximation, and the percent error of each approximation from left to right respectively.
2. Table with five columns containing step size, approximations found with RK2 Method using those step sizes, the exact solution, the absolute error of each approximation, and the percent error of each approximation from left to right respectively.
3. Table with five columns containing step size, approximations found with RK4 Method using those step sizes, the exact solution, the absolute error of each approximation, and the percent error of each approximation from left to right respectively.
4. Table with five columns containing step size, approximations found with Euler's Method using those step sizes, approximations found with RK2 using those step sizes, approximations found with RK4 using those step sizes, and the exact solution from left to right respectively.

Here are the 3 plots:







Here are the 4 tables:

Time_Step	Eulers_Method	Exact_Solution	Difference1	Percent_Difference1
0.3	53.87285694	53.77354821	0.09930873	0.184679518
0.1	53.80771036	53.77354821	0.034162146	0.063529648
0.05	53.79075856	53.77354821	0.017210349	0.032005232
0.025	53.78218547	53.77354821	0.008637262	0.016062287
0.0125	53.77787483	53.77354821	0.004326624	0.008046008

Time_Step	RK_2	Exact_Solution	Difference2	Percent_Difference2
0.3	53.77034728	53.77354821	-0.003200926	-0.005952604
0.1	53.77321013	53.77354821	-0.000338075	-0.000628702
0.05	53.77346472	53.77354821	-8.35E-05	-0.000155265
0.025	53.77352746	53.77354821	-2.07E-05	-3.86E-05
0.0125	53.77354304	53.77354821	-5.17E-06	-9.62E-06

Time_Step	RK_4	Exact_Solution	Difference3	Percent_Difference3
0.3	53.77354622	53.77354821	-1.99E-06	-3.70E-06
0.1	53.77354819	53.77354821	-2.33E-08	-4.34E-08
0.05	53.77354821	53.77354821	-1.44E-09	-2.68E-09
0.025	53.77354821	53.77354821	-8.93E-11	-1.66E-10
0.0125	53.77354821	53.77354821	-5.63E-12	-1.05E-11

Time_Step	Eulers_Method	RK_2	RK_4	Exact_Solution
0.3	53.87285694	53.77034728	53.77354622	53.77354821
0.1	53.80771036	53.77321013	53.77354819	53.77354821
0.05	53.79075856	53.77346472	53.77354821	53.77354821
0.025	53.78218547	53.77352746	53.77354821	53.77354821
0.0125	53.77787483	53.77354304	53.77354821	53.77354821

4 Conclusion

Based on the plots you can tell that Euler's Method is less accurate than the RK2 and RK4 methods. However, it is hard to tell from the plots whether or not RK2 is more accurate than RK4 or vice-versa. Looking at the tables however, we can see that the absolute errors and percent differences are smaller for the RK4 method than for the RK2 method. So from the plots and tables we have determined the methods from most accurate to least accurate are as follows:

1. Fourth Order Runge-Kutta Method
2. Second Order Runge-Kutta Method
3. Euler's Method

We can find the order of accuracy of each method by looking at the absolute errors. The order of accuracy is dependent on the method and can be found by looking at error and step size. A numerical solution is said to be nth order accurate if the error, E, is proportional to the step-size h to the nth power:

$$E(h) = Ch^n \tag{22}$$

The corresponding big-O notation:

$$O(h^n) \tag{23}$$

Looking at the tables, we see we have absolute error for the step sizes .1, .05, .025, and .0125. We can use these errors to determine the order of accuracy. For Eulers Method:

$$\begin{cases} E(.1) = 2 \cdot E(.05) = 4 \cdot E(.025) = 8 \cdot E(.0125) \\ Ch_1^n = 2 \cdot Ch_2^n \\ h_1 = 2h_2 \\ C2^n h_2^n = 2Ch_2^n \\ 2^n = 2 \\ n = 1 \end{cases} \tag{24}$$

The order of accuracy of Euler's Method is therefore 1. For Second Order Runge-Kutta Method:

$$\begin{cases} E(.1) = 4 \cdot E(.05) = 16 \cdot E(.025) = 64 \cdot E(.0125) \\ Ch_1^n = 4 \cdot Ch_2^n \\ h_1 = 2h_2 \\ C2^n h_2^n = 4Ch_2^n \\ 2^n = 4 \\ n = 2 \end{cases} \tag{25}$$

The order of accuracy of Second Order Runge-Kutta Method is therefore 2. For Fourth Order Runge-Kutta Method:

$$\begin{cases} E(.1) = 16 \cdot E(.05) = 256 \cdot E(.025) = 4096 \cdot E(.0125) \\ Ch_1^n = 16 \cdot Ch_2^n \\ h_1 = 2h_2 \\ C2^n h_2^n = 16Ch_2^n \\ 2^n = 16 \\ n = 4 \end{cases} \tag{26}$$

The order of accuracy of Fourth Order Runge-Kutta Method is therefore 4.

A Implementation in MatLab

The `MatLab` implementation with comments is given here:

FallingObjects.m:

```
% FallingObjects USES ALL THE OTHER FILES IN THIS DIRECTORY. IT PRODUCES THREE
% GRAPHS/PLOTS AND FOUR TABLES


g = 9.81;   %
c = .25;    % DEFINES GRAVITY, DRAG COEFFICIENT, AND MASS RESPECTIVELY, WHICH
m = 75;     % WERE DEFINED IN THE HOMEWORK ASSIGNTMENT

EulersMethodPlot(.3,15,g,c,m); % CREATES THREE SEPERATE PLOTS OF THE EXACT
RK2Plot(.3,15,g,c,m);          % SOLUTION VS A NUMERICAL SOLUTION USING A
RK4Plot(.3,15,g,c,m);          % NUMERICAL METHOD

% THE REMAINDER OF THE PROGRAM DEALS WITH CREATING TABLES WHICH DEMONSTRATE THE
% ACCURACY OF EULERS METHOD, SECOND ORDER RUNGE-KUTTA METHOD, AND FOURTH ORDER
% RUNGE-KUTTA METHOD WHEN APPROXIMATING A NUMERICAL SOLUTION TO THE ORDINARY
% DIFFERENTIAL EQUATION GIVEN IN THE HOMEWORK

% CREATES A MATRIX OF EXACT SOLUTIONS FOR COMPARISON VALUE USEING Analytical
Exact_Solution = [Analytical(15,g,c,m);Analytical(15,g,c,m);...
                Analytical(15,g,c,m);Analytical(15,g,c,m);Analytical(15,g,c,m)];

% CREATES A MATRIX OF TIME STEPS
Time_Step = [.3; .1; .05; .025; .0125];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CREATES A MATRIX OF NUMERICAL SOLUTIONS USING THE STEP SIZES IN THE Time_Step
% MATRIX USING EULERS METHOD
Eulers_Method=[EulersMethod(.3,15,g,c,m);EulersMethod(.1,15,g,c,m);...
     EulersMethod(.05,15,g,c,m);EulersMethod(.025,15,g,c,m);...
     EulersMethod(.0125,15,g,c,m)];

% CREATES A MATRIX WITH THE DIFFERENCES BETWEEN THE APPROXIMATED SOLUTION FOUND
% USING EULERS METHOD AND THE EXACT SOLUTION FOR THE TIME STEPS IN THE
% Time_Step MATRIX.
Difference1 = Eulers_Method - Exact_Solution;

% CREATES A MATRIX WITH THE PERCENT DIFFERENCE BETWEEN THE APPROXIMATED SOLUTION
% FOUND USING EULERS METHOD AND THE EXACT SOLUTION FOR THE TIME STEPS IN THE
% Time_Step MATRIX.
Percent_Difference1=((Eulers_Method - Exact_Solution)/Analytical(15,g,c,m))*100;

%MAKE TABLE FOR EULERS METHOD
T1=table(Time_Step,Eulers_Method,Exact_Solution,Difference1,...
         Percent_Difference1);
disp(T1); % DISPLAY TABLE
writetable(T1,'T1.xls','sheet',1,'Range','B2:F7');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CREATES A MATRIX OF NUMERICAL SOLUTIONS USING THE STEP SIZES IN THE Time_Step
% MATRIX USING THE SECOND ORDER RUNGE-KUTTA METHOD
RK_2=[RK2(.3,15,g,c,m);RK2(.1,15,g,c,m);RK2(.05,15,g,c,m);RK2(.025,15,g,c,m);...
     RK2(.0125,15,g,c,m)];

% CREATES A MATRIX WITH THE DIFFERENCES BETWEEN THE APPROXIMATED SOLUTION FOUND
% USING SECOND ORDER RUNGE-KUTTA METHOD AND THE EXACT SOLUTION FOR THE TIME
% STEPS IN THE Time_Step MATRIX.
Difference2 = RK_2 - Exact_Solution;

% CREATES A MATRIX WITH THE PERCENT DIFFERENCES BETWEEN THE APPROXIMATED
% SOLUTION FOUND USING SECOND ORDER RUNGE-KUTTA METHOD AND THE EXACT SOLUTION
% FOR THE TIME STEPS IN THE Time_Step MATRIX.
Percent_Difference2=((RK_2 - Exact_Solution)/Analytical(15,g,c,m))*100;

% MAKE TABLE FOR SECOND ORDER RUNGE-KUTTA METHOD (RK2)
T2=table(Time_Step,RK_2,Exact_Solution,Difference2,Percent_Difference2);
disp(T2); % DISPLAY TABLE
writetable(T2,'T2.xls','sheet',1,'Range','B2:F7');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% CREATES A MATRIX OF NUMERICAL SOLUTIONS USING THE STEP SIZES IN THE Time_Step
% MATRIX USING THE FOURTH ORDER RUNGE-KUTTA METHOD
RK_4=[RK4(.3,15,g,c,m);RK4(.1,15,g,c,m);RK4(.05,15,g,c,m);RK4(.025,15,g,c,m);...
     RK4(.0125,15,g,c,m)];

% CREATES A MATRIX WITH THE DIFFERENCES BETWEEN THE APPROXIMATED SOLUTION FOUND
% USING FOURTH ORDER RUNGE-KUTTA METHOD AND THE EXACT SOLUTION FOR THE TIME
% STEPS IN THE Time_Step MATRIX.
Difference3 = RK_4 - Exact_Solution;
```

```
% CREATES A MATRIX WITH THE PERCENT DIFFERENCES BETWEEN THE APPROXIMATED
% SOLUTION FOUND USING FOURTH ORDER RUNGE-KUTTA METHOD AND THE EXACT SOLUTION
% FOR THE TIME STEPS IN THE Time_Step MATRIX.
Percent_Difference3=((RK_4 - Exact_Solution)/Analytical(15,g,c,m))*100;

% MAKE TABLE FOR FOURTH ORDER RUNGE-KUTTA METHOD (RK4)
T3=table(Time_Step,RK_4,Exact_Solution,Difference3,Percent_Difference3);
disp(T3); % DISPLAY TABLE
writetable(T3,'T3.xls','sheet',1,'Range','B2:F7');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% DISPLAYS A TABLE THAT SHOWS SOLUTIONS FOUND USING TIME STEPS IN Time_Step
% MATRIX FOR ALL NUMERICAL SOLUTION METHODS AND THE EXACT SOLUTION
T4 = table(Time_Step, Eulers_Method, RK_2, RK_4, Exact_Solution);
disp(T4); % DISPLAY TABLE
writetable(T4,'T4.xls','sheet',1,'Range','B2:F7');
```

acceleration.m:

```
function dv_dt = acceleration(gravity, dragCoefficient, mass, velocity)

%acceleration SOLVES FOR THE ACCELERATION OF A GIVEN OBJECT AT A GIVEN TIME
%  THIS FUNCTION SOLVES THE ACCELERATION OF A MASS IN FREE FALL AFFECTED BY
%   DRAG. THIS FUNCTION TAKES IN FOUR PARAMETERS: gravity, dragCoefficient, mass
%   WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY, AND velocity WHICH DEFINES THE VELOCITY OF THE OBJECT AT THE TIME
%   ACCELERATION NEEDS TO BE CALCULATED FOR

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               %
    v = velocity;           %

    % CALCULATE ACCELERATION BASED ON PARAMETERS GIVEN
    dv_dt = g - (c/m) * v^2;
    % dv_dt NOW EQUALS THE ACCELERATION OF THE OBJECT AT VELCOITY velocity

end
```

Analytical.m:

```
function exact_solution = Analytical(time, gravity, dragCoefficient, mass)
```

```
%Analytical SOLVES THE EXACT SOLUTION OF A GIVEN OBJECTS VELOCITY AT A GIVEN
%TIME
%  THIS FUNCTION SOLVES THE REAL VELOCITY OF A MASS IN FREE FALL AFFECTED BY
%   DRAG. THIS FUNCTION TAKES IN FOUR PARAMETERS: time WHICH DEFINES THE TIME
%   AT WHICH VELOCITY SHOULD BE SOLVED FOR AND gravity, dragCoefficient, mass
%   WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               %
    t = time;               %

    % CALCULATE THE VELOCITY OF THE OBJACT AT TIME time SECONDS WHEN THE INITIAL
    % VELOCITY IS ZERO
    exact_solution = sqrt((g * m) / c) * tanh(sqrt((g * c) / m) * t);
    % exact_solution NOW EQUALS THE REAL VELOCITY OF THE OBJECT AT end_time
end
```

AnalyticalPlot.m:

```
function exact_solution = AnalyticalPlot(time_step, end_time, gravity,...
                                         dragCoefficient, mass)

%AnalyticalPlot PLOTS THE EXACT SOLUTION OF A GIVEN OBJECTS VELOCITY AT A GIVEN
%TIME
%  THIS FUNCTION PLOTS THE REAL VELOCITY OF A MASS IN FREE FALL AFFECTED BY DRAG
%   THIS FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME
%   STEP BETWEEN EACH PLOT OF VELOCITY, end_time WHICH DEFINES HOW
%   LONG THE FREE FALL MOTION SHHOULD BE OBSERVED FOR, gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               %

    % CALCULATE AND PLOT THE EXACT SOLUTION OF VELOCITY AT A
    % GIVEN TIME FROM 0->end_time SECONDS
    t = (0:time_step:end_time);
    y = sqrt((g * m) / c) * tanh(sqrt((g * c) / m) * t);
    plot(t,y);

    exact_solution = sqrt((g * m) / c) * tanh(sqrt((g * c) / m) * end_time);
    % exact_solution NOW EQUALS THE REAL VELOCITY OF THE OBJECT AT end_time
end
```

EulersMethod.m:

```
function velocity = EulersMethod(time_step, end_time, gravity,...
                                 dragCoefficient, mass)

%EulersMethod NUMERICALLY SOLVES THE VELOCITY OF AN OBJECT AFTER end_time
%SECONDS IN FREE FALL
%   THIS FUNCTION NUMERICALLY SOLVES THE VELOCITY OF A MASS IN FREE FALL
%   AFFECTED BY DRAG USING EULERS METHOD. THIS FUNCTION TAKES IN FIVE PARAMETERS
%   : time_step WHICH DEFINES THE TIME STEP BETWEEN EACH ITERATION OF EULERS
%   METHOD, end_time WHICH DEFINES HOW LONG THE FREE FALL MOTION SHHOULD BE
%   OBSERVED FOR, AND gravity, dragCoefficient, and mass WHICH ARE ALL
%   PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF DIFFERENT MASS WITH
%   DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % CALCULATE THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME (endt_time) USING EULERS METHOD FOR ALL ITERATIONS OF
    % EULERS METHOD. THE NUMBER OF ITERATIONS IS GIVEN BY end_time / time_step

    for time = time_step: time_step: end_time
        velocity = velocity + acceleration(g, c, m, velocity) * time_step;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING EULERS METHOD
end
```

EulersMethodPlot.m:

```
function velocity = EulersMethodPlot(time_step, end_time, gravity,...
                                 dragCoefficient, mass)

%EulersMethodPlot PLOTS ITERATIONS OF EULERS METHOD WTIH THE GIVEN STEP SIZE
%UNTIL THE END TIME
```

```
%   THIS FUNCTION PLOTS THE VELOCITY OF A MASS IN FREE FALL AFFECTED BY DRAG.
%   THE PLOT USING EULERS METHOD TO NUMERICALLY SOLVE FOR VELOCITY IS SHOWN IN
%   RED CIRCLES WHILE THE EXACT SOLUTION IS SHOWN AS A BLUE LINE ON THE PLOT.
%   THIS FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME
%   STEP BETWEEN EACH ITERATION OF EULERS METHOD, end_time WHICH DEFINES HOW
%   LONG THE FREE FALL MOTION SHHOULD BE OBSERVED FOR, gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % PLOT THE FIRST POINT OF EULERS METHOD AND CONFIGURE THE PLOT              %
    figure
    plot(time, velocity,'or'); hold on;

    title('Plot of Velocity Over Time Using Eulers Method'); % PLOT TITLE
    xlabel('Time in Seconds Mass Has Been in Free Fall (s)');% AXIS LABELS
    ylabel('Velcoity in Meters per Second of Mass in Free Fall (m/s)');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    str = {strjoin({'Gravity_________=',num2str(g),'m/s^2'}),strjoin(...       %
                   {'Drag Coefficient_=',num2str(c),'kg/m'}),strjoin(...       %
                   {'Mass of Object___=',num2str(m),'kg'}),strjoin(...         %
                   {'Initial Velocity_=',num2str(velocity),'m/s'}),strjoin(... %
                   {'Time Step________=',num2str(time_step),'s'})};            %
        % CONFIGURE INFORMATION DISPLAY THAT WILL GO ON THE PLOT %%%%%%%%%%%%%%%

    % CALCULATE AND PLOT THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME USING EULERS METHOD FOR ALL ITERATIONS OF EULERS METHOD.
    % THE NUMBER OF ITERATIONS IS GIVEN BY end_time / time_step
    for time = time_step: time_step: end_time
        velocity = velocity + acceleration(g,c,m,velocity) * time_step;
        plot(time, velocity,'or');hold on;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING EULERS METHOD

    % PLOT THE ACTUAL VELOCITY OF THE OBJECT OVER THE TIME INTERVAL 0->end_time
    AnalyticalPlot(.01, end_time, g, c, m);
    % THE REST OF THE CODE IS DEDICATED TO FORMATTING THE GRAPH
    h = zeros(2, 1);                           %
    h(1) = plot(0,0,'or', 'visible', 'on');    % FORMAT LEGEND
    h(2) = plot(0,0,'-b', 'visible', 'on');    %
    legend(h, 'Eulers Method','Exact Solution','location','southeast');%

    t = text(end_time - (9.8*end_time)/10,velocity,str,'interpreter','none');
    t.FontName = 'FixedWidth'; % FORMAT ADDITIONAL INFORMATION TEXT


end
```

RK2.m:

```
function velocity = RK2(time_step, end_time, gravity,...
                        dragCoefficient, mass)

%RK2 NUMERICALLY SOLVES THE VELOCITY OF AN OBJECT AFTER end_time
%SECONDS IN FREE FALL
%   THIS FUNCTION NUMERICALLY SOLVES THE VELOCITY OF A MASS IN FREE FALL
%   AFFECTED BY DRAG USING A SECOND ORDER RUNGE-KUTTA APPROXIMATION (RK2). THIS
%   FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME STEP
%   BETWEEN EACH ITERATION OF EULERS METHOD, end_time WHICH DEFINES HOW LONG THE
%   FREE FALL MOTION SHHOULD BE OBSERVED FOR, AND gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % CALCULATE THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME (endt_time) USING RK2 METHOD FOR ALL ITERATIONS OF
    % RK2. THE NUMBER OF ITERATIONS IS GIVEN BY end_time / time_step
    for time = time_step: time_step: end_time
        k1 = time_step * acceleration(g,c,m,velocity);
        k2 = time_step * acceleration(g,c,m,velocity + k1 / 2);
        velocity = velocity + k2;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING THE SECOND ORDER
    % RUNGE-KUTTA METHOD
end
```

RK2Plot.m:

```
function velocity = RK2Plot(time_step, end_time, gravity,...
                        dragCoefficient, mass)

%RK2Plot PLOTS ITERATIONS OF THE SECOND ORDER RUNGE-KUTTA (RK2) METHOD WTIH THE
%GIVEN STEP SIZE UNTIL THE END TIME
%   THIS FUNCTION PLOTS THE VELOCITY OF A MASS IN FREE FALL AFFECTED BY DRAG.
%   THE PLOT USING THE RK2 METHOD TO NUMERICALLY SOLVE FOR VELOCITY IS SHOWN IN
%   RED CIRCLES WHILE THE EXACT SOLUTION IS SHOWN AS A BLUE LINE ON THE PLOT.
%   THIS FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME
%   STEP BETWEEN EACH ITERATION OF EULERS METHOD, end_time WHICH DEFINES HOW
%   LONG THE FREE FALL MOTION SHHOULD BE OBSERVED FOR, gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % PLOT THE FIRST POINT OF RK2 AND CONFIGURE THE PLOT            %
    figure
    plot(time, velocity,'or'); hold on;

    title('Plot of Velocity Over Time Using Second Order Runge-Kutta Method');
    xlabel('Time in Seconds Mass Has Been in Free Fall (s)');% AXIS LABELS
    ylabel('Velcoity in Meters per Second of Mass in Free Fall (m/s)');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    str = {strjoin({'Gravity_________=',num2str(g),'m/s^2'}),strjoin(...      %
               {'Drag Coefficient_=',num2str(c),'kg/m'}),strjoin(...      %
               {'Mass of Object___=',num2str(m),'kg'}),strjoin(...        %
               {'Initial Velocity_=',num2str(velocity),'m/s'}),strjoin(... %
               {'Time Step_______=',num2str(time_step),'s'})};            %
        % CONFIGURE INFORMATION DISPLAY THAT WILL GO ON THE PLOT %%%%%%%%%%%%

    % CALCULATE AND PLOT THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME USING RK2 METHOD FOR ALL ITERATIONS OF RK2. THE NUMBER OF
    % ITERATIONS IS GIVEN BY end_time / time_step
    for time = time_step: time_step: end_time
        k1 = time_step * acceleration(g,c,m,velocity);
        k2 = time_step * acceleration(g,c,m,velocity + k1 / 2);
        velocity = velocity + k2;
        plot(time, velocity,'or');hold on;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING THE SECOND ORDER
    % RUNGE-KUTTA METHOD

    % PLOT THE ACTUAL VELOCITY OF THE OBJECT OVER THE TIME INTERVAL 0->end_time
    AnalyticalPlot(.01, end_time, g, c, m);

    % THE REST OF THE CODE IS DEDICATED TO FORMATTING THE GRAPH
    h = zeros(2, 1);                            %
    h(1) = plot(0,0,'or', 'visible', 'on');   % FORMAT LEGEND
    h(2) = plot(0,0,'-b', 'visible', 'on');   %
    legend(h, 'RK2','Exact Solution','location','southeast');

    t = text(end_time - (9.8*end_time)/10,velocity,str,'interpreter','none');
    t.FontName = 'FixedWidth'; % FORMAT ADDITIONAL INFORMATION TEXT

end
```

RK4.m:

```
function velocity = RK4(time_step, end_time, gravity,...
                        dragCoefficient, mass)

%RK4 NUMERICALLY SOLVES THE VELOCITY OF AN OBJECT AFTER end_time
%SECONDS IN FREE FALL
%   THIS FUNCTION NUMERICALLY SOLVES THE VELOCITY OF A MASS IN FREE FALL
%   AFFECTED BY DRAG USING A FOURTH ORDER RUNGE-KUTTA APPROXIMATION (RK4). THIS
%   FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME STEP
%   BETWEEN EACH ITERATION OF EULERS METHOD, end_time WHICH DEFINES HOW LONG THE
%   FREE FALL MOTION SHHOULD BE OBSERVED FOR, AND gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % CALCULATE THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME (endt_time) USING RK4 METHOD FOR ALL ITERATIONS OF
    % RK4. THE NUMBER OF ITERATIONS IS GIVEN BY end_time / time_step
    for time = time_step: time_step: end_time
        k1 = time_step * acceleration(g,c,m,velocity);
        k2 = time_step * acceleration(g,c,m,velocity + k1 / 2);
        k3 = time_step * acceleration(g,c,m,velocity + k2 / 2);
        k4 = time_step * acceleration(g,c,m,velocity + k3);
        velocity = velocity + (k1 + 2*k2 + 2*k3 + k4) / 6;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING THE FOURTH ORDER
    % RUNGE-KUTTA METHOD
end
```

RK4Plot.m:

```
function velocity = RK4Plot(time_step, end_time, gravity,...
                        dragCoefficient, mass)

%RK4Plot PLOTS ITERATIONS OF THE FOURTH ORDER RUNGE-KUTTA (RK2) METHOD WTIH THE
%GIVEN STEP SIZE UNTIL THE END TIME
%   THIS FUNCTION PLOTS THE VELOCITY OF A MASS IN FREE FALL AFFECTED BY DRAG.
%   THE PLOT USING THE RK4 METHOD TO NUMERICALLY SOLVE FOR VELOCITY IS SHOWN IN
%   RED CIRCLES WHILE THE EXACT SOLUTION IS SHOWN AS A BLUE LINE ON THE PLOT.
%   THIS FUNCTION TAKES IN FIVE PARAMETERS: time_step WHICH DEFINES THE TIME
%   STEP BETWEEN EACH ITERATION OF EULERS METHOD, end_time WHICH DEFINES HOW
%   LONG THE FREE FALL MOTION SHHOULD BE OBSERVED FOR, gravity, dragCoefficient,
%   and mass WHICH ARE ALL PARAMETERS THAT CAN BE PLAYED WITH FOR OBJECTS OF
%   DIFFERENT MASS WITH DIFFERENT DRAG COEFFICIENTS UNDER THE FORCE OF DIFFERENT
%   GRAVITY

    g = gravity;            %
    c = dragCoefficient;    % VARIABLE DEFINITIONS THAT MAKE TYPING EASIER
    m = mass;               % THE INITIAL TIME IS ALWAYS ZERO
    time = 0.0;             % THE INITIAL VELOCITY IS ALWAYS ZERO
    velocity = 0;           %

    % PLOT THE FIRST POINT OF RK4 AND CONFIGURE THE PLOT             %
    figure
    plot(time, velocity,'or'); hold on;

    title('Plot of Velocity Over Time Using Second Order Runge-Kutta Method');
    xlabel('Time in Seconds Mass Has Been in Free Fall (s)');% AXIS LABELS
    ylabel('Velcoity in Meters per Second of Mass in Free Fall (m/s)');
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    str = {strjoin({'Gravity__________=',num2str(g),'m/s^2'}),strjoin(...      %
                   {'Drag Coefficient_=',num2str(c),'kg/m'}),strjoin(...       %
                   {'Mass of Object___=',num2str(m),'kg'}),strjoin(...         %
                   {'Initial Velocity_=',num2str(velocity),'m/s'}),strjoin(... %
                   {'Time Step________=',num2str(time_step),'s'})};            %
        % CONFIGURE INFORMATION DISPLAY THAT WILL GO ON THE PLOT %%%%%%%%%%%%%

    % CALCULATE AND PLOT THE NUMERICAL APPROXIMATION OF VELOCITY AT A
    % GIVEN TIME USING RK4 METHOD FOR ALL ITERATIONS OF RK4. THE NUMBER OF
    % ITERATIONS IS GIVEN BY end_time / time_step
    for time = time_step: time_step: end_time
        k1 = time_step * acceleration(g,c,m,velocity);
        k2 = time_step * acceleration(g,c,m,velocity + k1 / 2);
        k3 = time_step * acceleration(g,c,m,velocity + k2 / 2);
        k4 = time_step * acceleration(g,c,m,velocity + k3);
        velocity = velocity + (k1 + 2*k2 + 2*k3 + k4) / 6;
        plot(time, velocity,'or');hold on;
    end
    % velocity NOW EQUALS THE NUMERICAL SOLUTION USING THE FOURTH ORDER
    % RUNGE-KUTTA METHOD

    % PLOT THE ACTUAL VELOCITY OF THE OBJECT OVER THE TIME INTERVAL 0->end_time
    AnalyticalPlot(.01, end_time, g, c, m);

    % THE REST OF THE CODE IS DEDICATED TO FORMATTING THE GRAPH
    h = zeros(2, 1);                            %
    h(1) = plot(0,0,'or', 'visible', 'on');     % FORMAT LEGEND
    h(2) = plot(0,0,'-b', 'visible', 'on');     %
    legend(h, 'RK4','Exact Solution','location','southeast');

    t = text(end_time - (9.8*end_time)/10,velocity,str,'interpreter','none');
    t.FontName = 'FixedWidth'; % FORMAT ADDITIONAL INFORMATION TEXT


end
```