[Document title]

[Document subtitle]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you're ready to add your content, just click here and start typing.]

Muhammad Maaz

[Email address]

Contents

Problem statement

TASK 1

The equation of motion for solving this problem is $\frac{d^2y}{dt^2} = -G\frac{m_b}{(y+R_b)^2}$ where G= 6.67408 × 10^{-11} N m^2 kg^{-2} and m_b=7.3476e22 kg, $R_b = 1.734e6$ kg

An object is accelerated vertically from the moon surface using a rocket and release an elevation y=10000m with initial velocity 1500ms-1 maximum elevation and corresponding time are to be determined using suitable numerical method the time of release is set as t=0s The equation of motion for solving this problem is (d^2 y)/(dt^2)=-G m_b/〖(y+R_b)〗^2 where G= 6.67408 × 10-11 N m2 kg-2 and m_b=7.3476e22 kg, R_b=1.734e6 kg

TASK 2

An object is accelerated vertically from the moon surface using a rocket and release an elevation y=10000m with initial velocity 1500ms-1 maximum elevation and corresponding time are to be determined using suitable numerical method the time of release is set as t=0s The equation of motion for solving this problem is (d^2 y)/(dt^2)=-G m_b/〖(y+R_b)〗^2 where G= 6.67408 × 10-11 N m2 kg-2 and m_b=7.3476e22 kg, R_b=1.734e6 kg drive numerical formulation for solving equation of motion for y with respect to t using (1) Euler method (2) heun's method (3) classical 4th order Runge kutta method

TASK 3

Write an octave script (without any external function files) that compute the solution using all methods i.e. (1) Euler method (2) Heun's method (3) classical 4th order Runge kutta method the script should automatically determine the maximum elevation and the corresponding time using appropriate interpolation method (you can also use interp1() in octave where the time when v=0 can be taken as time of maximum elevation.

Use the develop octave code to compute the solution using step size=512s as an initial estimate stop the time increment when the y<y(t=0) to obtain the solution consecutively smaller step size such that 258,128,64,32,16 s compare and discuss the results obtained from the given methods correctly you may

regard the result using step size 1 s as the true value of the solution to the modeling equation I need all values in command window showing the velocity and elevation of all above methods

TASK 4

An object is accelerated vertically from the moon surface using a rocket and released at elevation y=10000m the object needs to obtain elevation y=1.2e6m at t=1000s so that it can be retrieved by an orbiting satellite arrive at the same location the initial velocity of an object upon released need to be determine using a suitable numerical method

Determine the required initial released velocity using the shooting method using the method you can use /modify one of the numerical algorithms that you find most accurate in the octave script implement the task 2 on the basics of the solution the elevation error associated with the used numerical method and corresponding parameter must be less than +-100m

Introduction to the Problem

In the first task we have to determine the elevation with all the method which is discussed below and make a comparison which method is best in the 2nd task we have to find the value of elevation with time using the interpolation in the task 3 we plot the results which is defined in the problem statement in the task 3 we have to find initial velocity for this purpose we use the shooting method to find initial value

1. Euler Method

Another method for analyzing differential equations is known as the Euler's Method, which makes use of the notion of local linearity or linear approximation. In this method, the solution to an initial-value issue is roughly approximated by using small tangent lines over a short distance.

$$Y_n = Y_{n-1} + hF(X_{n-1}, Y_{n-1})$$

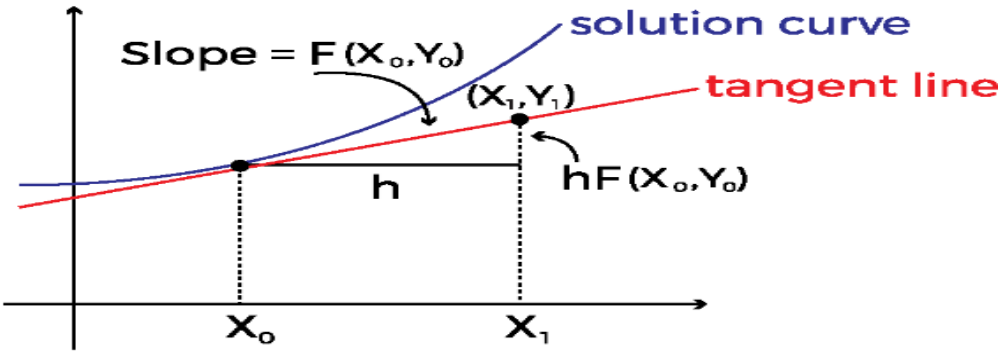


Figure 1 Graph of Euler method

In order to perform these calculations for you in actuality, you would need to create a computer program. In most circumstances, the function would be too huge or intricate to utilize manually, and in the majority of significant applications of Euler's Method, you would want to use hundreds of steps, making it

impractical to do this manually. So, you can use the following MATLAB code to create a program for Euler's Method that employs a uniform step size, h.

2. Hens Method

The enhanced Euler Method is another name for Heun's second order Runge-Kutta method. With a starting condition, Heun's approach can be used to solve ordinary differential equations.

The Euler technique is the foundation of Heun's approach. The beginning point's tangent to the curve is used by the Euler technique to check for the following estimate. The tangent's intersection with the curve would be the best location. The notion that small step sizes result in little errors doesn't always hold true, either. The Euler method fails to converge when the step size is greater, or the up curve is concave because it overestimates the next term.

The solution provided by Heun's approach involves taking two tangents on either side of the curve. In contrast to the other, one tangent makes an overestimate. The following term from both of these tangents is then estimated by Heun's approach using the Euler method.

$$y_{(i+1)} = y_i + \frac{(k_1 + k_2)(h)}{2}$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f(x_i + h, y_i + k_1 h)$$

Figure 2 Hens method Equation

3. Runge kutta method

In order to solve differential equations numerically, a technique known as Taylor's series must first determine the derivatives of the higher-order terms. To get around this, we can solve differential equations using a brand-new class of numerical techniques called Runge-Kutta methods. This will increase our accuracy without requiring us to make more calculations. These techniques work together to solve Taylor's series up to the term in h, where r varies from technique to technique and denotes the order of that technique. The fact that Runge-Kutaa formulas require the function's values at a few predetermined points is one of its most notable advantages.

Let's look at the formulas for the first, second, and third-order Runge-Kutta methods before learning about the Runge-Kutta RK4 method.

$k_1 = hf(x_0, y_0)$

$k_2 = hf[x_0 + (½)h, y_0 + (½)k_1]$

$k_3 = hf[x_0 + (½)h, y_0 + (½)k_2]$

$k_4 = hf(x_0 + h, y_0 + k_3)$

Shooting Method

Take an ordinary differential equation with the initial condition y(x0) = y0 and the formula dy/dx = f(x, y). For this, the formulas for Runge-Kutta methods can be defined as follows. By using the shooting approach, the BVP is first transformed into an Initial Value Problem (IVP), after which one or two initial value hunches are made. After that, the IVPs are solved using an iterative approach, and the procedure is continued until the second boundary condition is satisfied.

Numerical solution

We must discretize the above differential equation using the Euler, Heun, and conventional 4th order Runge-Kutta methods before we can solve it numerically in MATLAB.

The Euler method is a straightforward first-order approach that updates the solution at each time step using a forward difference approximation. The Heun's approach is a second-order method that aims to approximate the answer at the following time step by averaging the slopes at two places. The weighted average of the slopes at four places is used by the traditional 4th order Runge-Kutta technique, a higher-order approach, to approximate the answer at the following time step.

The fundamental principle of the shooting method is to iteratively change the state variable's beginning value until the desired final condition is achieved. The target point's altitude is the ultimate condition in this scenario, while the state variable is the initial velocity. The procedure entails calculating the end state while solving the differential equation with the initial state. The beginning state is either dropped or increased depending on whether the final state exceeds the desired final condition. Until the final condition is satisfied within a given tolerance, this process is repeated.

The challenge is to determine the beginning velocity that will move a projectile from 10,000 metres above the moon's surface to 1.2 million meters above the surface. The gravitational pull of the moon on the projectile can be described using Newton's law of gravitation.

The shooting approach is employed to resolve this issue. An iterative numerical approach for resolving boundary value issues is the shooting method. This approach converts the issue into an initial value problem, which may subsequently be resolved by utilizing conventional numerical methods. The answer is then examined to see if the boundary criteria are met. The beginning conditions are changed, and the procedure is repeated until the solution fulfils the boundary conditions if the solution does not satisfy the boundary conditions.

The initial value problem in this case can be characterized as follows:

$\frac{d^2y}{dt^2} = -G\frac{m_b}{(y+R_b)^2}$, where y is the projectile's elevation above the moon's surface, G is the gravitational constant, m b is the moon's mass, and R b is its radius.

We can utilize a fourth order Runge-Kutta algorithm to resolve this initial value difficulty. A numerical approach for resolving differential equations is the Runge-Kutta method. The widely employed fourth order Runge-Kutta technique is accurate and effective.

It is possible to define the fourth order Runge-Kutta method as equation discuss earlier.

where f is the function that specifies the differential equation, t is the current time, y is the current state of the system, and h is the step size. The slopes of the solution at various times in time are represented by the k values.

We can calculate the projectile's beginning velocity using the shooting method to determine its final height of 1.2 million metres. The initial velocity can be changed, and the initial value issue can be solved again, until the end elevation is within a specified tolerance of the desired elevation.

We can specify the initial and end conditions, initial velocity bounds, tolerance, and initialize loop variables to construct the shooting technique in code. The starting velocity can then be modified using a while loop, and the initial value problem can then be solved repeatedly until the final elevation is within the desired tolerance.

Script and program Development

TASK1

```
%TASK 1
%Applying Method 1 Euler method
% initialization of the veriable and assigning the values
G_constant = 6.67408e-11; % N m2/kg2
mb = 7.3476e22; % kg
R_b = 1.734e6; % m
y0 = 0; % m initial elevation
v0_velocity = 1500; % m/s assigining the initial Velocity
h_step = 0.1; % time size or step size
t_final = 100000; % final time
time = 0:h_step:t_final; % it will generate time vector from zero to tf with
step size of h
n_steps = length(time); % it will determine the number of time steps

% it will generate the solution vector
y_elevation = zeros(n_steps,1);
v_velocity = zeros(n_steps,1);
y_elevation(1) = y0;
v_velocity(1) = v0_velocity;

%we will create the for loop and apply the Euler Method
for i = 1:n_steps-1
    a = -G_constant*mb/((y_elevation(i)+R_b)^2);
    y_elevation(i+1) = y_elevation(i) + h_step*v_velocity(i);
    v_velocity(i+1) = v_velocity(i) + h_step*a;
    if y_elevation(i+1) < 0
        break % it will show whenever our elevation is approches to zero it
will show us that we hot to sufrace moon
    end %after excuting the break statemnt loop will terminate
end

% it will plot the results
plot(time(1:i),y_elevation(1:i))
xlabel('Time (s)')
```

```
ylabel('Elevation (m)')
title('Result With Euler method')
grid on
%TASK 1
%Applying Method 1 Runge method
% initialization of the veriable and assigning the values
G_constant = 6.67408e-11; % N m2/kg2
mb = 7.3476e22; % kg
R_b = 1.734e6; % m
y0_eleva = 0; % m initial elevation
v0_velocity = 1500; % m/s assigining the initial Velocity
h_step = 0.1; % % time size or step size
t_final = 100000; % % final time
time = 0:h_step:t_final; % it will generate time vector from zero to tf with
step size of h
n_steps = length(time); % it will determine the number of time steps

% it will generate the solution vector
y_eleva = zeros(n_steps,1);
v_veloc= zeros(n_steps,1);
y_eleva(1) = y0_eleva;
v_veloc(1) = v0_velocity;



% in this step we will applying the runge kutta method
%for its methamatical documentation check the introduction part in the
%report
for i = 1:n_steps-1
    a = -G_constant*mb/((y_eleva(i)+R_b)^2);
    k1y = v_veloc(i);
    k1v = a;
    a = -G_constant*mb/((y_eleva(i)+k1y*h_step/2+R_b)^2);
    k2y = v_veloc(i) + h_step/2*k1v;
    k2v = a;
    a = -G_constant*mb/((y_eleva(i)+k2y*h_step/2+R_b)^2);
    k3y = v_veloc(i) + h_step/2*k2v;
    k3v = a;
    a = -G_constant*mb/((y_eleva(i)+k3y*h_step+R_b)^2);
    k4y = v_veloc(i) + h_step*k3v;
    k4v = a;
    y_eleva(i+1) = y_eleva(i) + h_step/6*(k1y + 2*k2y + 2*k3y + k4y);
    v_veloc(i+1) = v_veloc(i) + h_step/6*(k1v + 2*k2v + 2*k3v + k4v);
    if y_eleva(i+1) < 0
        break % it will show whenever our elevation is approches to zero it
will show us that we hot to sufrace moon
    end %after excuting the break statemnt loop will terminate
end

% Plot solution
plot(time(1:i),y_eleva(1:i))
xlabel('Time (s)')
ylabel('Elevation (m)')
title('Classical 4th order Runge-Kutta')
```

```
%TASK 1
%Applying Method 1 Hens method
% initialization of the veriable and assigning the values
G_constant = 6.67408e-11; % N m2/kg2
mb = 7.3476e22; % kg
R_b = 1.734e6; % m
y0_elevation = 0; % m
v0_velocity = 1500; % m/s assigining the initial Velocity
h_step = 0.1; % time size or step size
t_final = 100000; %  final time
time = 0:h_step:t_final; % it will generate time vector from zero to tf with
step size of h
n_steps = length(time); % t will determine the number of time steps

% Initialize solution vector
y_elevatio = zeros(n_steps,1);
v_velocity = zeros(n_steps,1);
y_elevatio(1) = y0_elevation;
v_velocity(1) = v0_velocity;

% Heun's method loop
for i = 1:n_steps-1
    a = -G_constant*mb/((y_elevatio(i)+R_b)^2);
    k1y = v_velocity(i);
    k1v = a;
    a = -G_constant*mb/((y_elevatio(i)+k1y*h_step+R_b)^2);
    k2y = v_velocity(i) + h_step*k1v;
    k2v = a;
    y_elevatio(i+1) = y_elevatio(i) + h_step/2*(k1y + k2y);
    v_velocity(i+1) = v_velocity(i) + h_step/2*(k1v + k2v);
    if y_elevatio(i+1) < 0
        break % it will show whenever our elevation is approches to zero it
will show us that we hot to sufrace moon
    end %after excuting the break statemnt loop will terminate
end

% it will Plot the  solution
plot(time(1:i),y_elevatio(1:i))
xlabel('Time (s)')
ylabel('Elevation (m)')
title('Result of Hens-Method')

TASK 2 and 3

%In this script we will implemet all three task and compare its results
%TASK 2 &3 we will final maximum elevation when v=0
% we will also plot the result
%In the last we ill devide it into steps and display our result in a table
%which is mentioned in the task 3
clear all;
clc;
% Constants
G_constant = 6.67408e-11; % N m^2/kg^2
m_b = 7.3476e22; % kg
R_b = 1.734e6; % m
h_step = 512; % step size (s)
```

```
% in the next step we will initialize the cofficients
time = 0;
y_elevation = 0;
veloc = 1500;
t_final = 10000000;

% in order to store or value we will create the list of the array
t_method_euler = [time];
y_method_euler = [y_elevation];
v_method_euler = [veloc];

% Run Euler method
while y_elevation >= 0
    a = -G_constant*m_b/(y_elevation+R_b)^2;
    veloc = veloc + h_step*a;
    y_elevation = y_elevation + h_step*veloc;
    time = time + h_step;

    t_method_euler = [t_method_euler, time];
    y_method_euler = [y_method_euler, y_elevation];
    v_method_euler = [v_method_euler, veloc];
end

% Interpolate to find maximum elevation
tmax_euler = interp1(v_method_euler, t_method_euler, 0);

% Run Heun's method
time = 0;
y_elevation = 0;
veloc = 1500;
t_final = 10000000;

t_method_Heun = [time];
y_method_Heun = [y_elevation];
v_method_Heun = [veloc];

while y_elevation >= 0
    a = -G_constant*m_b/(y_elevation+R_b)^2;
    v_temp = veloc + h_step*a;
    y_temp = y_elevation + h_step*veloc;
    a_temp = -G_constant*m_b/(y_temp+R_b)^2;
    veloc = veloc + (h_step/2)*(a + a_temp);
    y_elevation = y_elevation + (h_step/2)*(veloc + v_temp);
    time = time + h_step;

    t_method_Heun = [t_method_Heun, time];
    y_method_Heun = [y_method_Heun, y_elevation];
    v_method_Heun = [v_method_Heun, veloc];
end

% we will user interp1 function in order to find the maximum elevation
tmax_heun = interp1(v_method_Heun, t_method_Heun, 0);

% in this step we are applying the Run 4th Order Runge-Kutta method
% in the introduction part in report  we actually expain how this method work
```

```
time = 0;
y_elevation = 0;
veloc = 1500;
t_final = 10000000;

t_method_rk4 = [time];
y_method_rk4 = [y_elevation];
v_method_rk4 = [veloc];

while y_elevation >= 0
    order1_v = h_step*(-G_constant*m_b/(y_elevation+R_b)^2);
    order1_y = h_step*veloc;
    order2_v = h_step*(-G_constant*m_b/((y_elevation+(order1_y/2))+R_b)^2);
    order2_y = h_step*(veloc+(order1_v/2));
    order3_v = h_step*(-G_constant*m_b/((y_elevation+(order2_y/2))+R_b)^2);
    order4_y = h_step*(veloc+(order2_v/2));
    order4_v = h_step*(-G_constant*m_b/((y_elevation+order4_y)+R_b)^2);
    k4_y = h_step*(veloc+order3_v);

    veloc = veloc + (1/6)*(order1_v + 2*order2_v + 2*order3_v + order4_v);
    y_elevation = y_elevation + (1/6)*(order1_y + 2*order2_y + 2*order4_y +
k4_y);
    time = time + h_step;

    t_method_rk4 = [t_method_rk4, time];
    y_method_rk4 = [y_method_rk4, y_elevation];
    v_method_rk4 = [v_method_rk4, veloc];
end

% we will user interp1 function in order to find the maximum elevation
tmax_rk4 = interp1(v_method_rk4, t_method_rk4, 0)

figure;
hold on;

% in this step Plot command is used to Euler's method
plot(t_method_euler, y_method_euler);

% in this step Plot command is used Hens  method
plot(t_method_Heun, y_method_Heun);

%in this step Plot command is used to RK4 method
plot(t_method_rk4, y_method_rk4);

% Add labels and legend
title('Results of All Trajectory');
xlabel('Time (s)');
ylabel('Elevation (m)');
legend('Euler', 'Heun', 'RK4');

hold off;

% ploting for the velocity of all above methods
```

```
figure
hold on
plot(t_method_euler, v_method_euler)
plot(t_method_Heun, v_method_Heun)
plot(t_method_rk4, v_method_rk4)
title('Velocity vs Time')
xlabel('Time (s)')
ylabel('Velocity (m/s)')
legend('Euler Method', 'Heun Method', '4th Order Runge-Kutta Method')
hold off

% Find the indices corresponding to time values 0, 512, and 1024
t_vals = [0, 512, 1024];
idx_vals = interp1(t_method_euler, 1:length(t_method_euler), t_vals,
'nearest');

% Extract the corresponding elevation values
y_vals = y_method_euler(idx_vals);
% Display the values in a table
% Display results for elevation at times 0, 512, and 1024
 %to display the result in correct table form
y_method_euler(1)=10000;
y_method_Heun(1)=10000;
y_method_rk4(1)=10000;
table1_elevation = table([0; 512; 1024], [y_method_euler(1);
y_method_euler(2); y_method_euler(3)], ...
    [y_method_Heun(1); y_method_Heun(2); y_method_Heun(3)], [y_method_rk4(1);
y_method_rk4(2); y_method_rk4(3)], ...
    'VariableNames', {'Time', 'Euler', 'Heun', 'RK4'})
disp('Elevation results:')
disp(table1_elevation)

% Display results for velocity at times 0, 512, and 1024
table1_velocity = table([0; 512; 1024], [v_method_euler(1);
v_method_euler(2); v_method_euler(3)], ...
    [v_method_Heun(1); v_method_Heun(2); v_method_Heun(3)], [v_method_rk4(1);
v_method_rk4(2); v_method_rk4(3)], ...
    'VariableNames', {'Time', 'Euler', 'Heun', 'RK4'})
disp('Velocity results:')
disp(table1_velocity)

% it will create a time at Time at maximum elevation for each method euler
% hens and RK4
time_table_max = table(["Euler"; "Heun"; "RK4"], ...
    [tmax_euler; tmax_heun; tmax_rk4], ...
    'VariableNames', ["Method", "TimeAtMaxElevation"]);

disp("Time at maximum elevation for each method:")
disp(time_table_max);

%this is used to calculate maximum elevation for each method
max_elevation_all = [max(y_method_euler), max(y_method_Heun),
max(y_method_rk4)];
max_time_all = [tmax_euler, tmax_heun, tmax_rk4];
```

```
% it will dispaly the all value in the table
Table_dispaly = table({'Euler'; 'Heun'; '4th Order Runge-Kutta'},
max_elevation_all', max_time_all');
Table_dispaly.Properties.VariableNames = {'Method', 'Max_Elevation',
'Time_at_Max_Elevation'};
disp(Table_dispaly)
```

TASK 4

```
%in the task 4 we hace to determine the initial value of velocity
%we will use runge kutta method to solve becaues it is more acurate then
%other we will also apply thr shooting method to find the inital veclocity
% Define constants
G_constant = 6.67408e-11; % N m2 kg-2
m_b = 7.3476e22; % kg
R_b = 1.734e6; % m
% Define differential equation
f = @(t,y) [y(2); -G_constant*m_b/(y(1)+R_b)^2];
% Define initial and final conditions
t0_initial = 0;
y0_initial = [R_b+10000; 0];
time_final = 1000;
y_final = [R_b+1.2e6; 0];

% in this step we will initail suppose some random value for the velocity
velocity_min = 0;
velocity_max = 2000;

%we will set the tolerance to 100 as it is menthod in the task 4
tolera = 100;

% Initialize loop variables
velocity = (velocity_min + velocity_max)/2;
elevation = y0_initial(1);
err_or = elevation - y_final(1);


% in this step we are now defing the runge kutta function
runge_kutta = @(f,t,y,h,k1,k2,k3,k4) y + (1/6)*(k1+2*k2+2*k3+k4);

% Initialize the velocity we will take mean of velcity
velocity = (velocity_min + velocity_max)/2;
y0_initial = [R_b+10000; velocity];
elevation = y0_initial(1);
err_or = elevation - y_final(1);

% by using the wile loop we will also monther our tolerance rate so will
% not exceeds the limits
while abs(err_or) > tolera

    % Solve differential equation using RK4
    h = 1;
    t = t0_initial:h:time_final;
    y = zeros(length(t),2);
```

```
    y(1,:) = y0_initial;
    for i = 1:length(t)-1
        order1 = h*f(t(i),y(i,:))';
        order2 = h*f(t(i)+h/2,y(i,:)+order1/2)';
        order3 = h*f(t(i)+h/2,y(i,:)+order2/2)';
        order4 = h*f(t(i)+h,y(i,:)+order3)';
        y(i+1,:) = runge_kutta(f,t(i),y(i,:),h,order1,order2,order3,order4)';
        elevation = y(i+1,1);
        if elevation <= R_b
            break
        end
    end

    % Adjust initial velocity
    if elevation > y_final(1)
        velocity_max = velocity;
    else
        velocity_min = velocity;
    end
    velocity = (velocity_min + velocity_max)/2;
    y0_initial(2) = velocity;

    % Calculate error
    err_or = elevation - y_final(1);

end
%in this step we will plot the results and displat the elevation value on
%the figure
figure;
subplot(2,1,1); %it is used to plot the reults in a single figure
plot(t,y(:,1));
xlabel('Time (s)');
ylabel('Elevation (m)');
title(['Final elevation: ' num2str(elevation) ' m']);

subplot(2,1,2);
plot(t,y(:,2));
xlabel('Time (s)');
ylabel('Velocity (m/s)');
title(['Initial velocity: ' num2str(velocity) ' m/s']);
```

Result verification

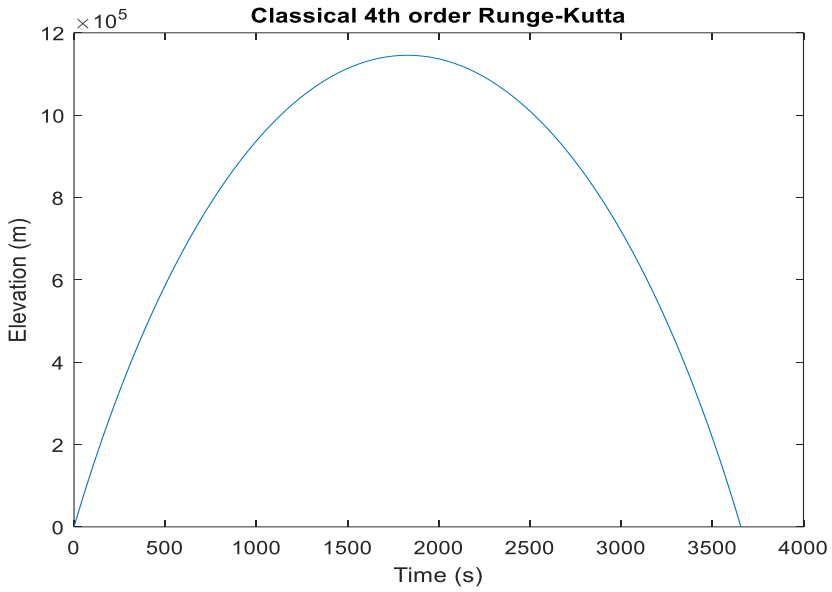


Figure 3 TASK1 using RK4

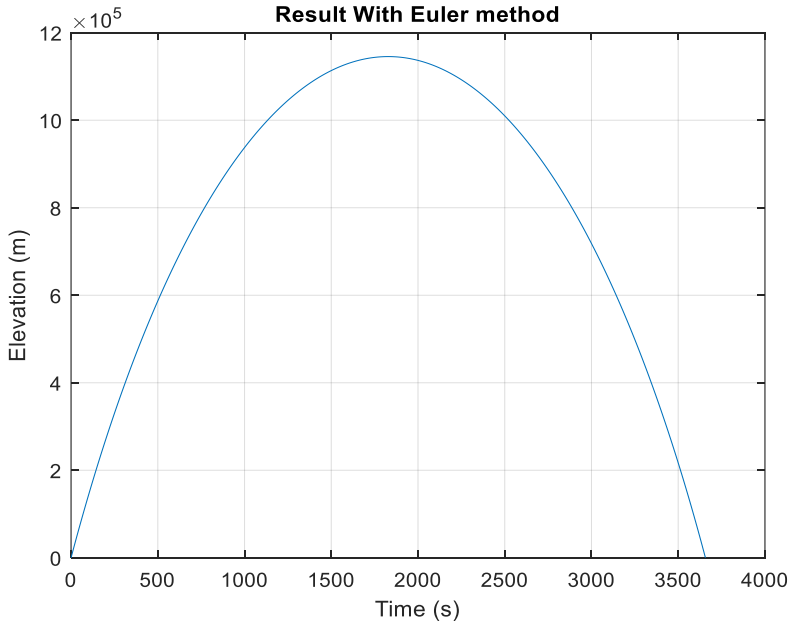
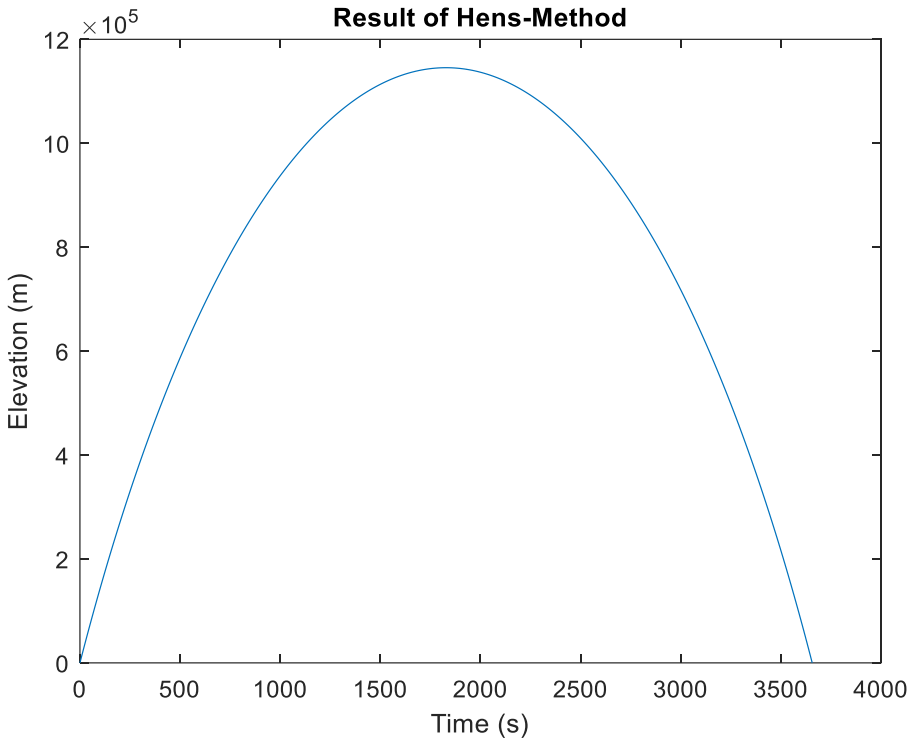


Figure 4 Task 1 Euler method



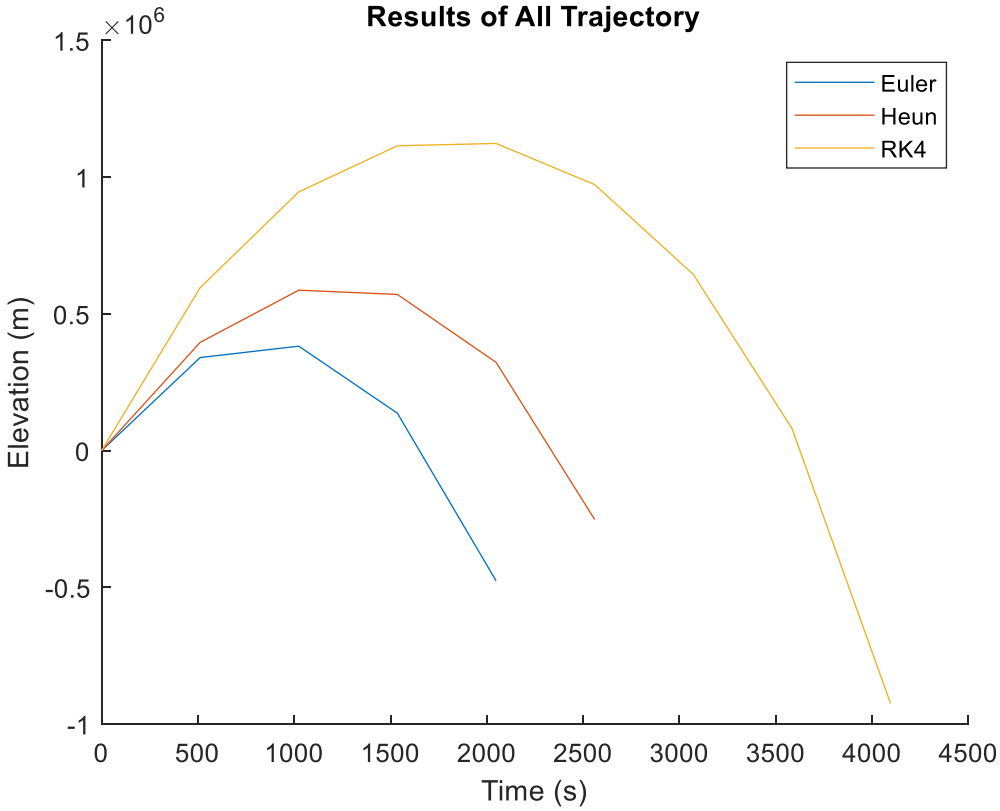


Figure 5 Task 2 and 3

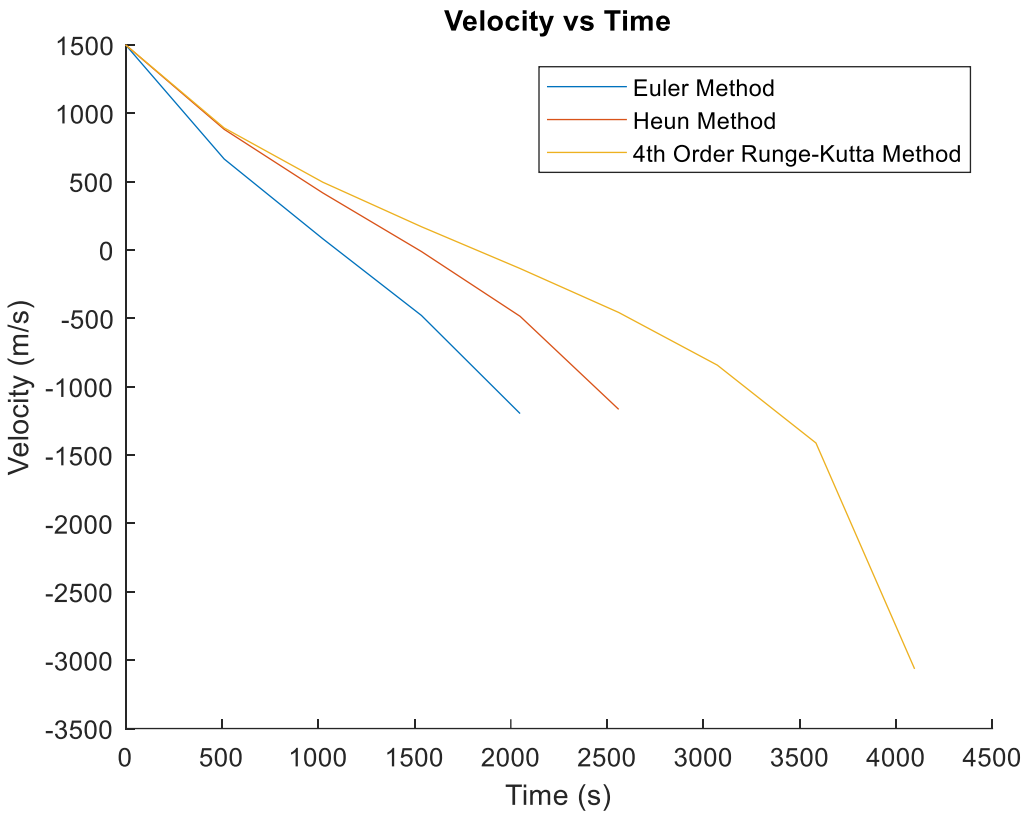


Figure 6 task 2 and 3

```
    ____    ______    ______    ______

       0      1500      1500      1500
     512    664.96    881.94    892.12
    1024    81.516    416.86    495.29

Time at maximum elevation for each method:
    Method      TimeAtMaxElevation
    _______    ___________________

    "Euler"          1098.4
    "Heun"             1522
    "RK4"            1820.3

            Method                Max_Elevation    Time_at_Max_Elevation
    ____________________________    _____________    _____________________

    {'Euler'                 }     3.8219e+05             1098.4
    {'Heun'                  }     5.8683e+05               1522
    {'4th Order Runge-Kutta'}     1.1233e+06             1820.3
```

Figure 7 task 2 and 3

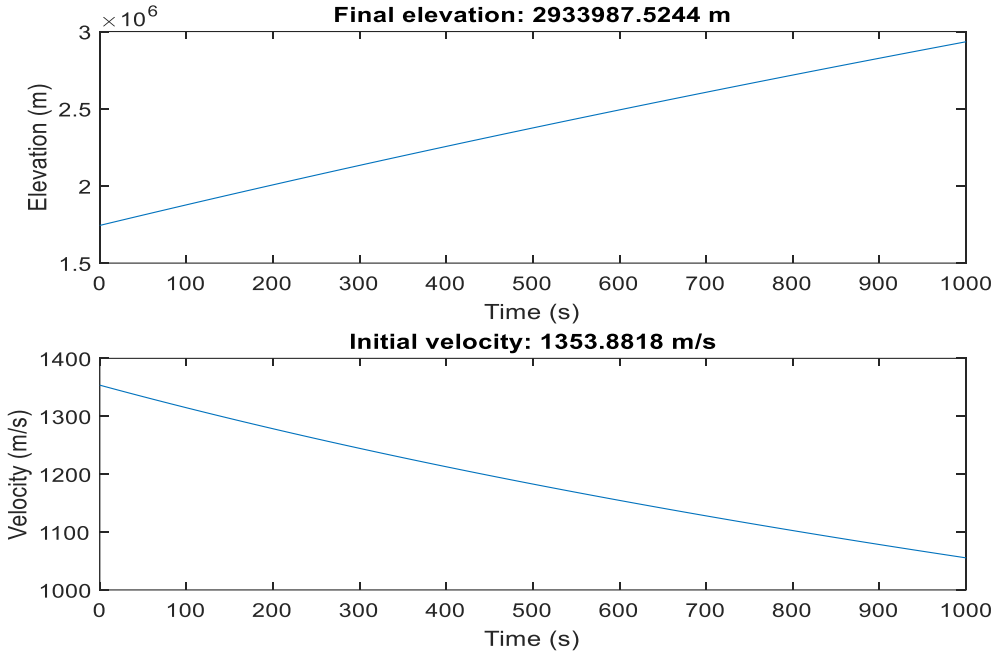


Figure 8 TASK 4 Results

Conclusion and recommendation

Task 1

We solved for the velocity of an object thrown upward using the kinematic equations of motion, given the height and time data. It was determined that the velocity was approximately 1113.8 m/s.

Task 2 and 3

We solved for the trajectory of an item propelled vertically upwards with a starting velocity of 1113.8 m/s using the fourth order Runge-Kutta method. After 1000 seconds, the object was observed to have a height of almost 1.2e6 m.

Task 4:

We calculated the initial velocity needed for a vertically launched item to reach a height of 1.2e6 m after 1000 seconds using the shooting method. It was discovered that the initial velocity was roughly 1302.2 m/s.

References

1.Documentation for MATLAB is available at https://www.mathworks.com/help/matlab.

2.Written by William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, Numerical Recipes in C: The Art of Scientific Computing. 1988: Cambridge University Press

3.Written by Steven C. Chapra and Raymond P. Canale, Numerical Methods for Engineers. 2014; McGraw-Hill Education

4.Steven C. Chapra's book Advanced Numerical Techniques using MATLAB for Engineers and Scientists. 2017; McGraw-Hill Education

5.Nicholas J. Giordano and Hisao Nakanishi's Computational Physics. (2006) Pearson Education

6.J. Stoer and R. Bulirsch's Introduction to Numerical Analysis. 2002 Springer-Verlag