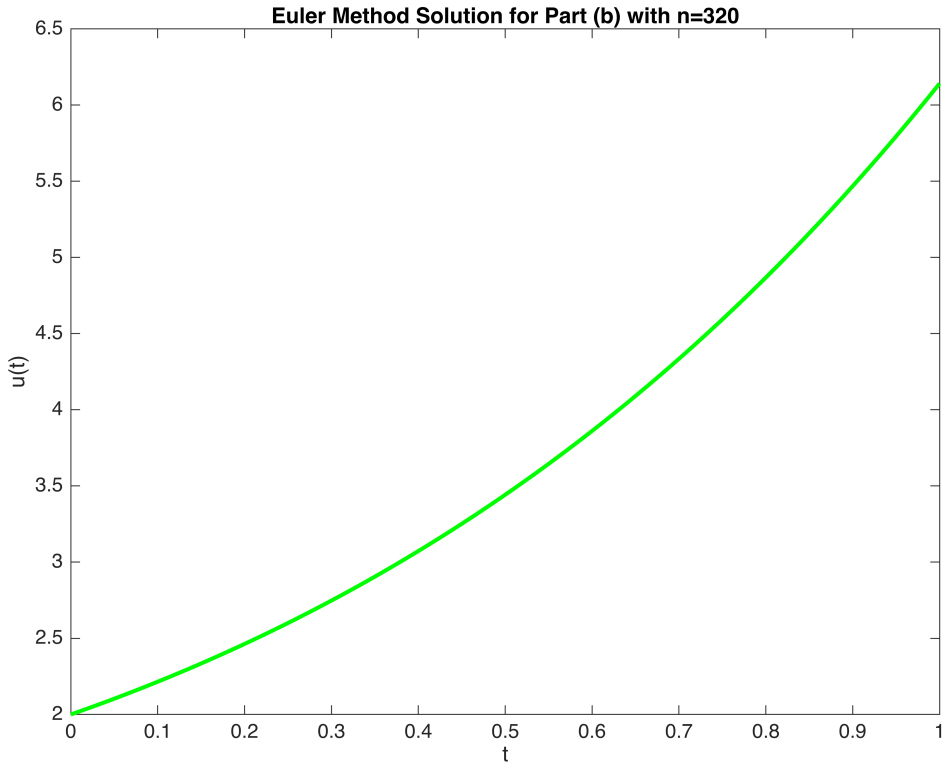
Muhideen Ogunlowo Math 426 HW10

Exercise 6.2.2 b i

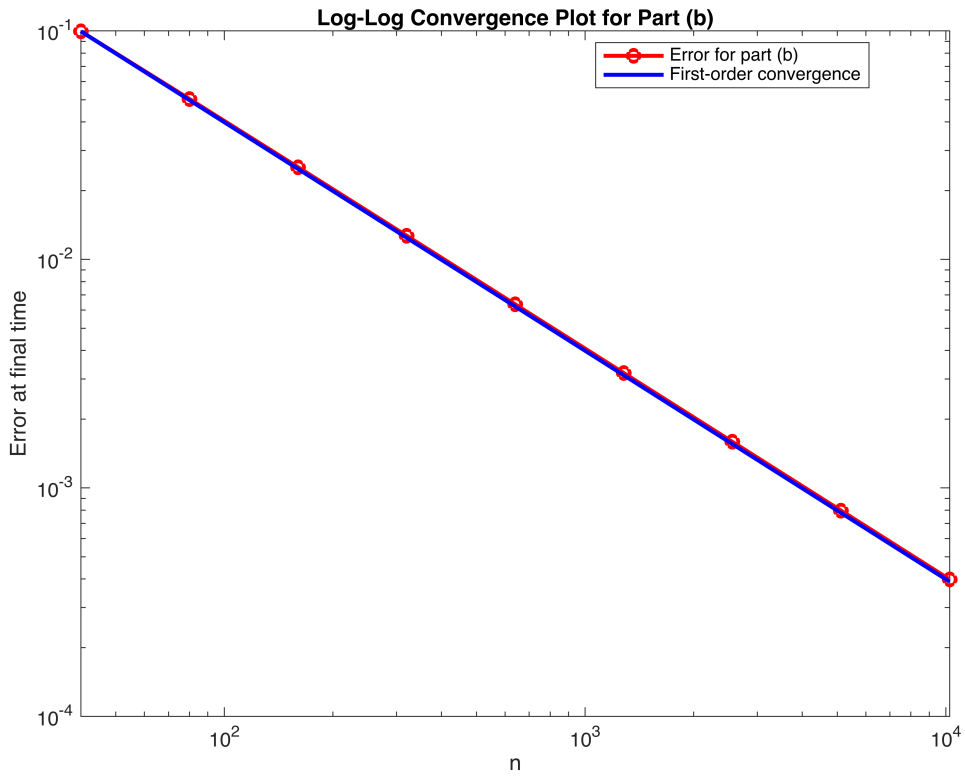
```
% Euler's Method for Solving IVPs
% This live script solves the IVP given in question 6.2.2(b) using Euler's
method.

%% Define the ODE for part (b)
dudt_b = @(t, u) u + t;

%% Define the initial condition and time span
u0_b = 2;
tspan_b = [0, 1];

%% Number of steps for the plot
n_b = 320;

%% Solve IVP for part (b) with n=320
[t_b, u_b] = eulerivp(dudt_b, tspan_b, u0_b, n_b);

%% Plot the solution
figure;
plot(t_b, u_b, 'g-', 'linewidth',2);
title('Euler Method Solution for Part (b) with n=320');
xlabel('t');
ylabel('u(t)');
grid off;
```



Exercise 6.2.2 b ii

```
%% Error analysis for different n values
% Define ns for the error analysis
ns = 10 * 2.^(2:10);
errors_b = zeros(size(ns)); % Preallocate error array

%% Exact solution for error comparison
exact_solution_b = @(t) -1 - t + 3*exp(t);

% Calculate errors at final time for different n
for i = 1:length(ns)
    n = ns(i);
    [t, u] = eulerivp(dudt_b, tspan_b, u0_b, n);
    exact_u_b = exact_solution_b(t(end));
    errors_b(i) = abs(u(end) - exact_u_b);
end

%% Create a log-log convergence plot
figure;
loglog(ns, errors_b, 'r-o', 'linewidth',2);
hold on;

% Reference line for first-order convergence
ref_line = (ns/ns(1)).^(-1) * errors_b(1);
```

```
loglog(ns, ref_line, 'b-','linewidth',2);

hold off;
title('Log-Log Convergence Plot for Part (b)');
xlabel('n');
ylabel('Error at final time');
legend('Error for part (b)', 'First-order convergence', 'Location', 'best');
grid off;
```



Exercise 6.3.1 a

$y''' - 3y'' + 3y' - y = t, y(0) = 1, y'(0) = 2, y''(0) = 3$

Assume $u_1 = y, u_2 = y'$, and $u_3 = y''$.

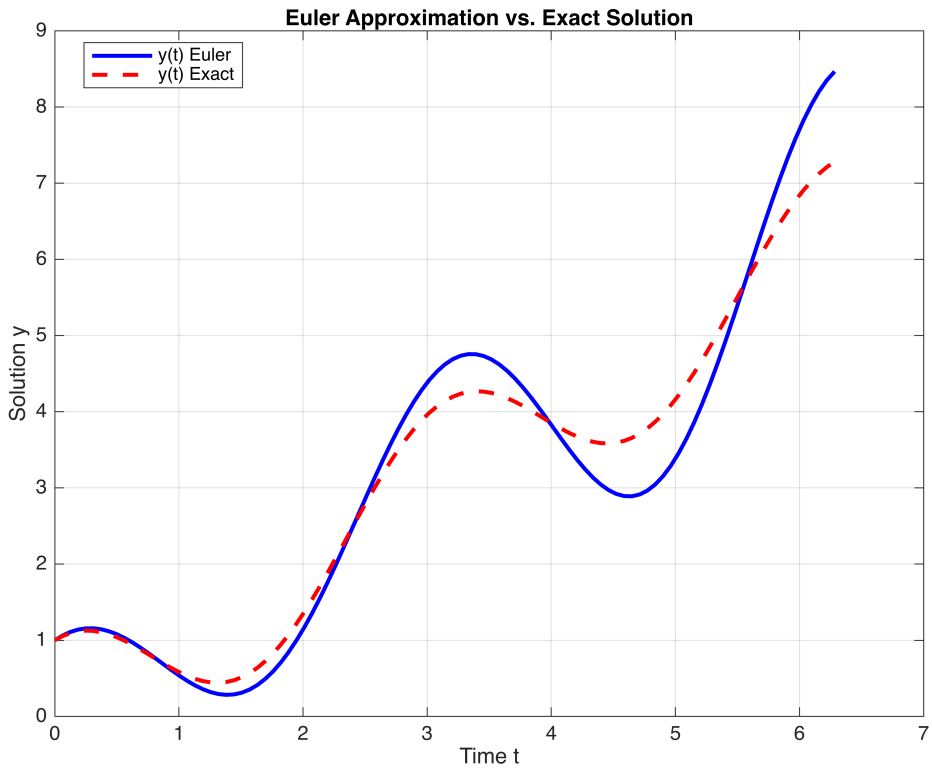
Therefore, $y''' - 3y'' + 3y' - y = t$ transfroms to :

$u_3' = t + 3u_3 - 3u_2 + u_1$

$u_1(0) = 1, u_2(0) = 2, u_3(0) = 3$

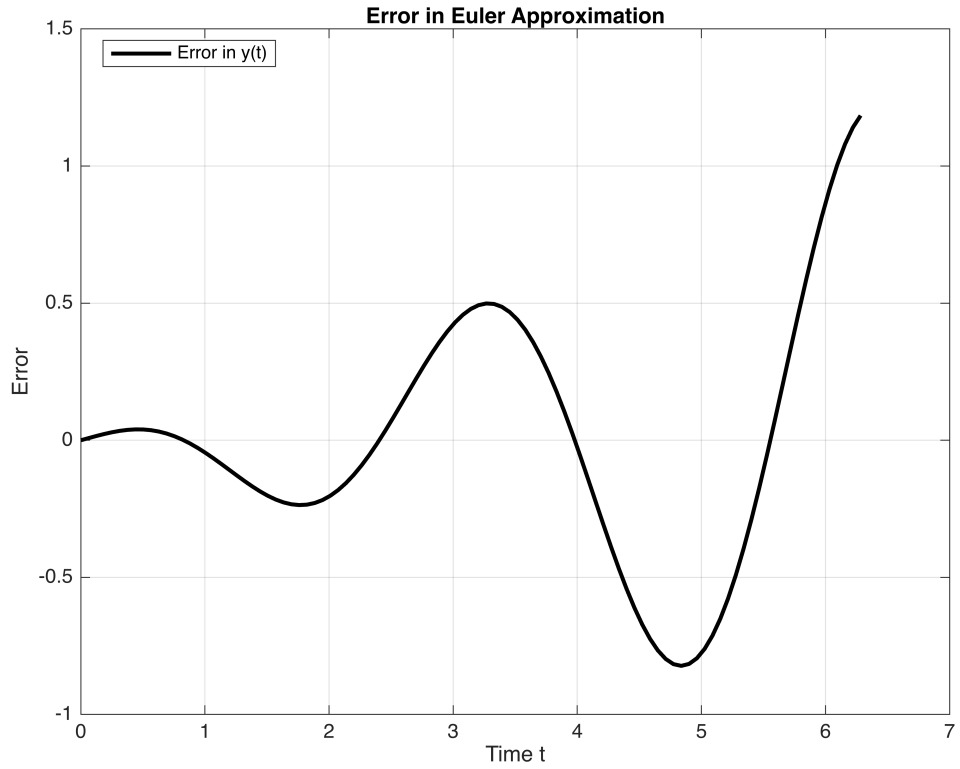
```
% Initial conditions
u0 = [1; 1];

% Time span
tspan = [0, 2*pi];

% Number of steps
```

```
n = 100;

% Solve the system using Euler's method
[t, u] = eulersys(@odefcn, tspan, u0, n);

% Exact solution
exact_solution = @(t) t + cos(2*t);
y_exact = arrayfun(exact_solution, t);

% Error calculation
error = u(1,:) - y_exact;

% Plot y(t) and y'(t) together with the exact solution
figure;
plot(t, u(1,:), 'b-', t, y_exact, 'r--', 'linewidth', 2);
title('Euler Approximation vs. Exact Solution');
xlabel('Time t');
ylabel('Solution y');
legend('y(t) Euler', 'y(t) Exact', 'Location', 'best');
grid on;
```




```
% Plot the error
figure;
plot(t, error, 'k-', 'linewidth', 2);
title('Error in Euler Approximation');
xlabel('Time t');
```

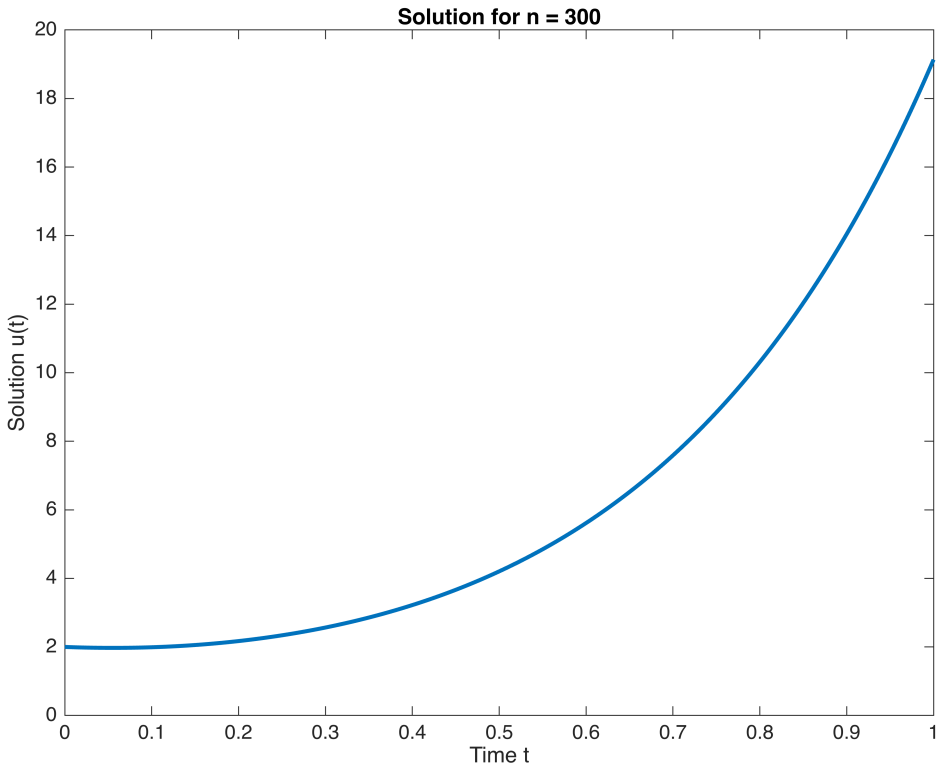
```
ylabel('Error');
legend('Error in y(t)', 'Location', 'best');
grid on;
```



Exercise 6.4.7c i

```
% Function handle for the ODE system
dudt = @(t, y) [y(2); 9*y(1) + 9*t];

% Initial conditions
u0 = [2, -1];

% Time span
tspan = [0, 1];

% Number of time steps
n = 300;

% Solve the ODE using the RK4 method
[t, u] = rk4(dudt, tspan, u0, n);

% Plot the solution for n = 300
figure;
plot(t, u(:,1), 'linewidth', 2);
```

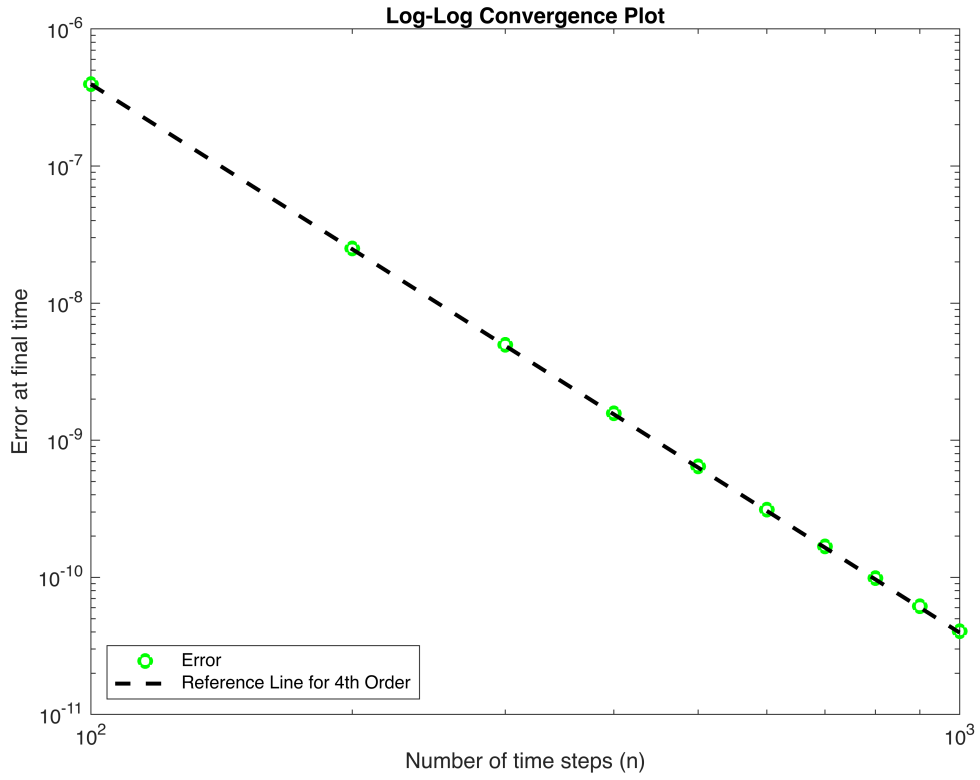
```
title('Solution for n = 300');
xlabel('Time t');
ylabel('Solution u(t)');
grid off;
```



Exercise 6.4.7c ii

```
% Analytical solution function
u_analytical = @(t) exp(3*t) + exp(-3*t) - t;

% Error computation and log-log convergence plot
n_values = 100:100:1000;
errors = zeros(size(n_values));

for i = 1:length(n_values)
    [t, u] = rk4(dudt, tspan, u0, n_values(i));
    errors(i) = abs(u(end, 1) - u_analytical(t(end)));
end

% Log-log convergence plot
figure;
loglog(n_values, errors, 'go', 'LineWidth', 2);
hold on;

% Reference line for fourth-order convergence
```

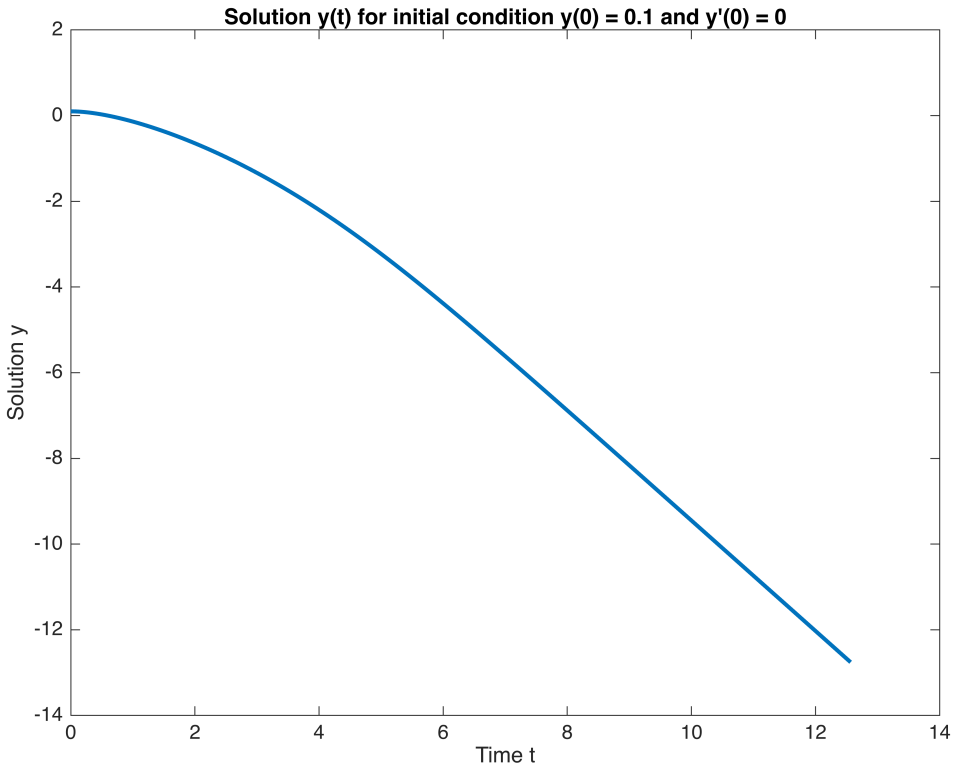
```
ref_line = (n_values/n_values(1)).^(-4) * errors(1);
loglog(n_values, ref_line, 'k--', 'LineWidth', 2);
hold off;

title('Log-Log Convergence Plot');
xlabel('Number of time steps (n)');
ylabel('Error at final time');
legend('Error', 'Reference Line for 4th Order', 'Location', 'SouthWest');
grid off;
```

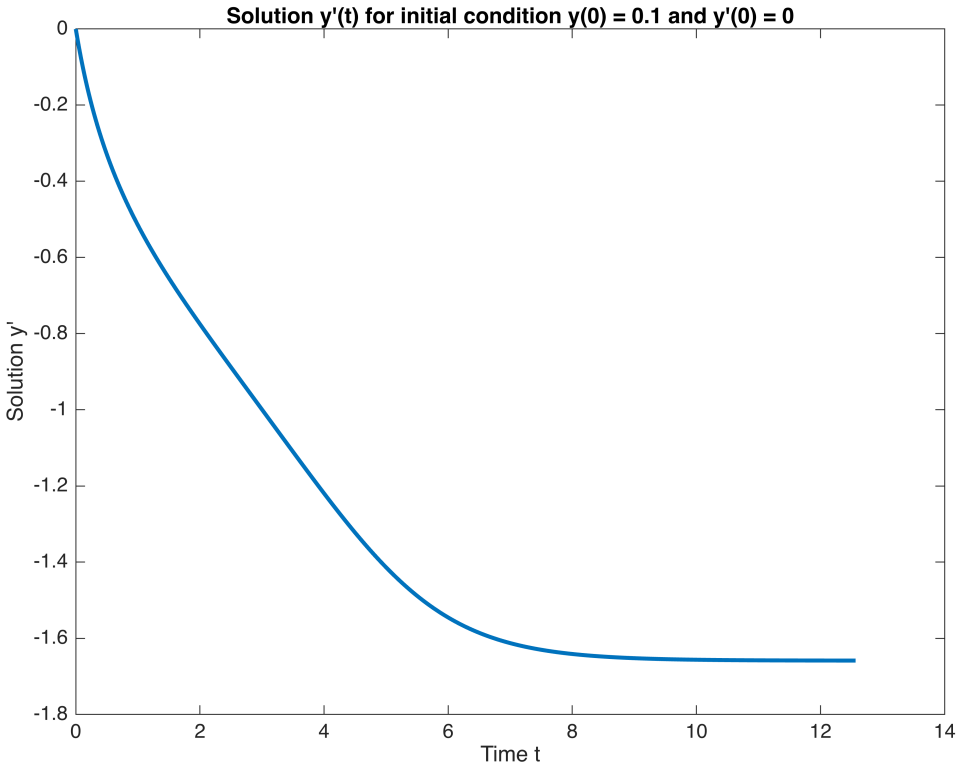


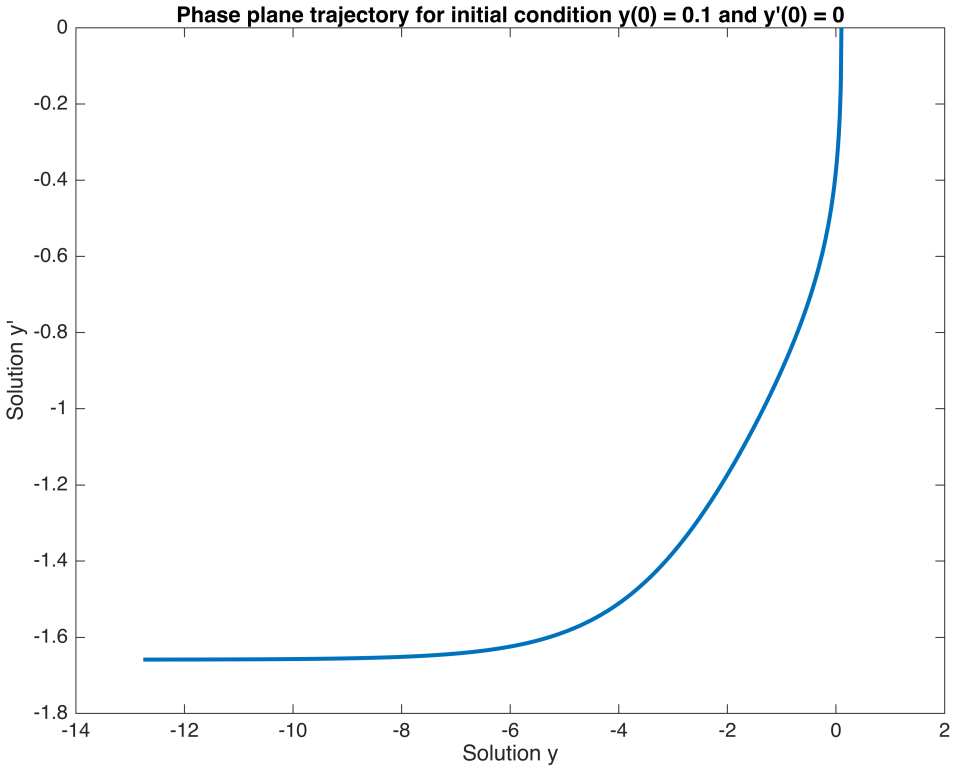
Exercise 6.5.1

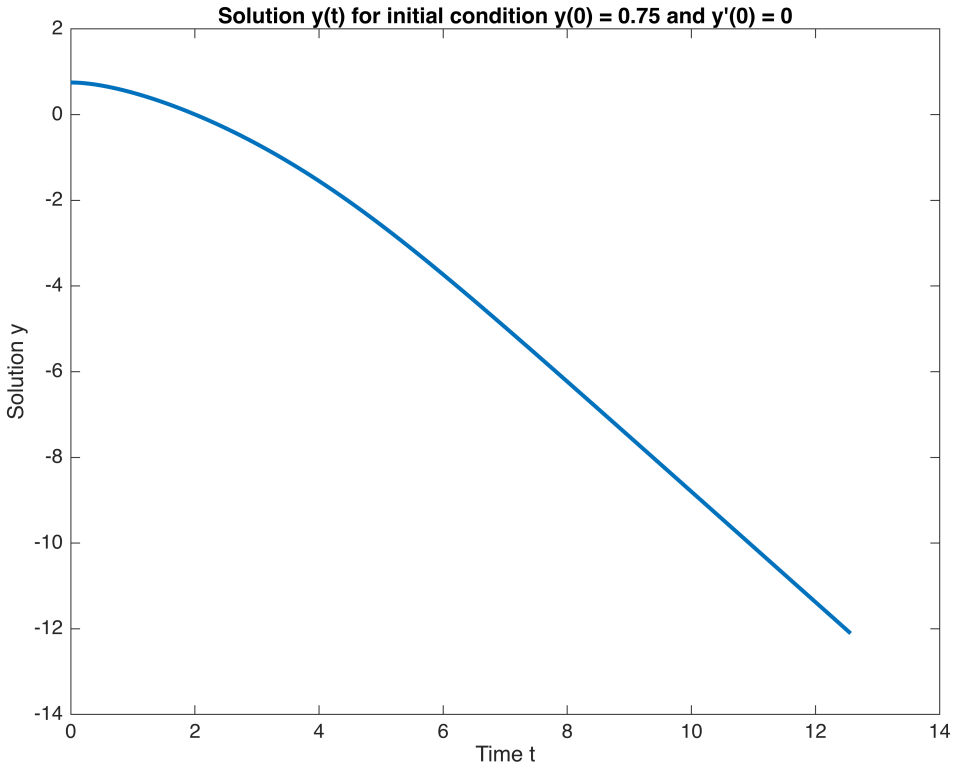
```
% Define the time span for the solution
tspan = [0, 4*pi];

% Define the tolerance for the solver
tol = 1e-6;

% Define the initial conditions and solve the ODE for each
initial_conditions = [0.1, 0; 0.5, 0; 0.75, 0; 0.95, 0];
for i = 1:size(initial_conditions, 1)
    y0 = initial_conditions(i, :);
    [t, u] = rk23(@ode_func, tspan, y0, tol);

    % Plot y(t)
    figure;
```

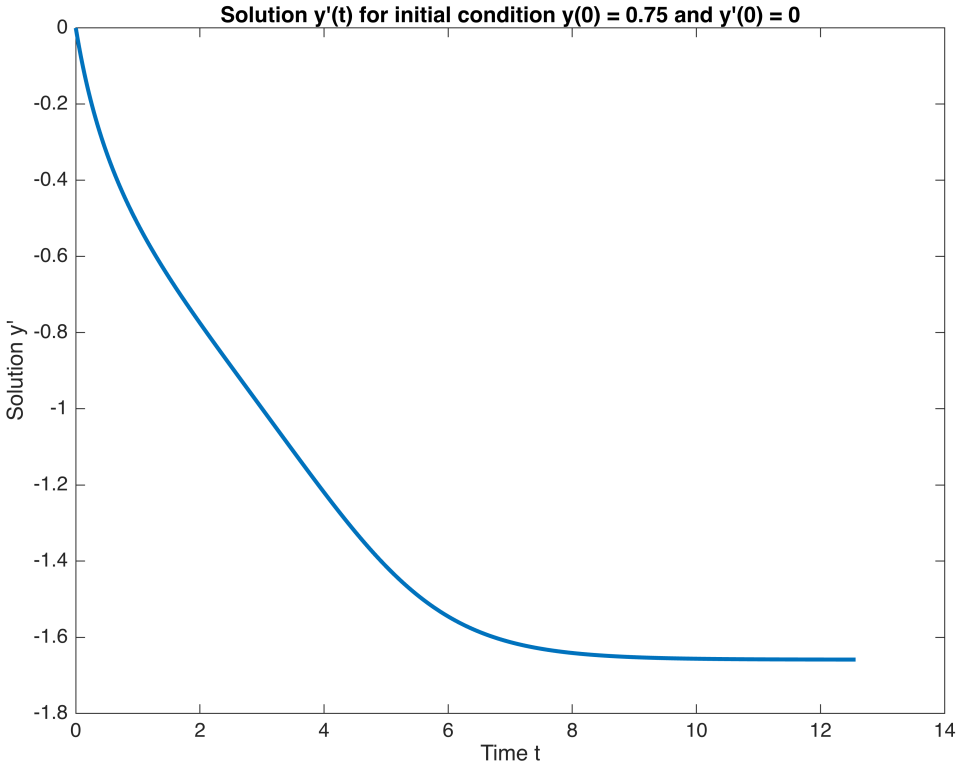
```
    plot(t, u(:, 1), 'linewidth', 2);
    title(['Solution y(t) for initial condition y(0) = ', num2str(y0(1)), '
and y''(0) = ', num2str(y0(2))]);
    xlabel('Time t');
    ylabel('Solution y');

    % Plot y'(t)
    figure;
    plot(t, u(:, 2), 'linewidth', 2);
    title(['Solution y''(t) for initial condition y(0) = ', num2str(y0(1)),
' and y''(0) = ', num2str(y0(2))]);
    xlabel('Time t');
    ylabel('Solution y''');

    % Plot the phase plane trajectory
    figure;
    plot(u(:, 1), u(:, 2), 'linewidth', 2);
    title(['Phase plane trajectory for initial condition y(0) = ',
num2str(y0(1)), ' and y''(0) = ', num2str(y0(2))]);
    xlabel('Solution y');
    ylabel('Solution y''');
end
```

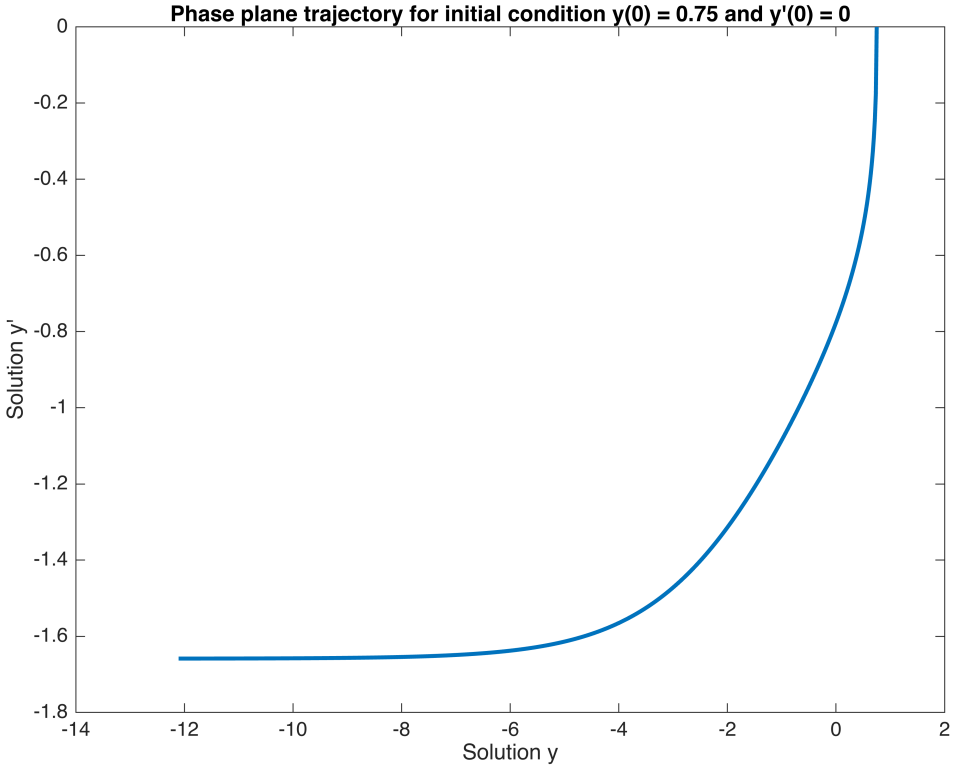


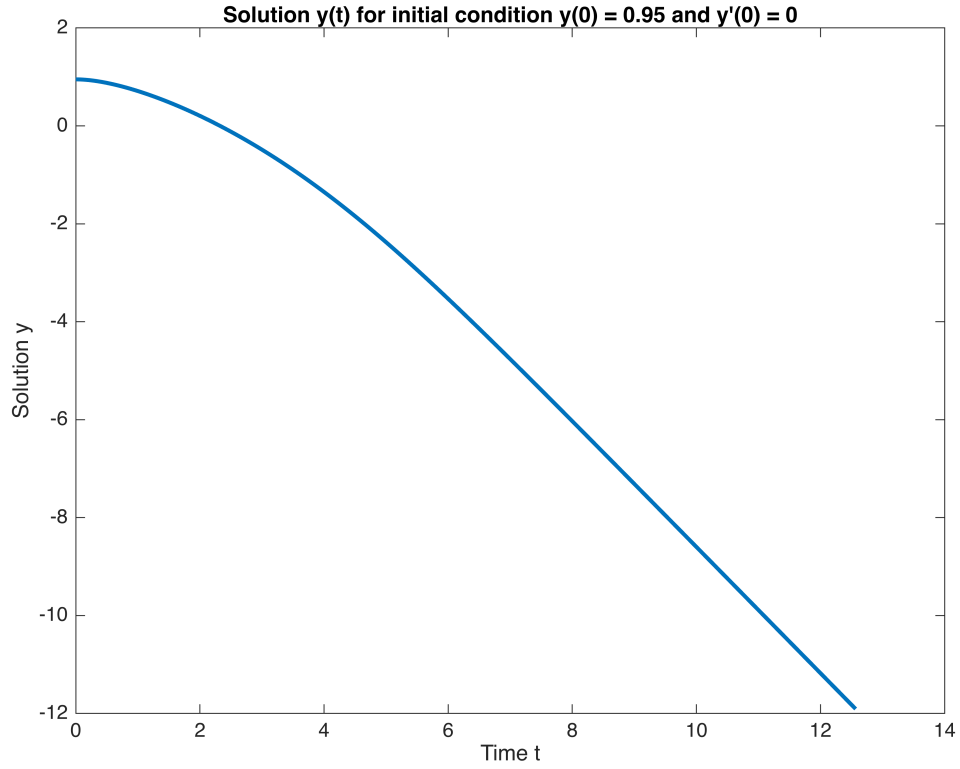

Solution y'(t) for initial condition y(0) = 0.1 and y'(0) = 0
Solution y'
Time t
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
0
2
4
6
8
10
12
14

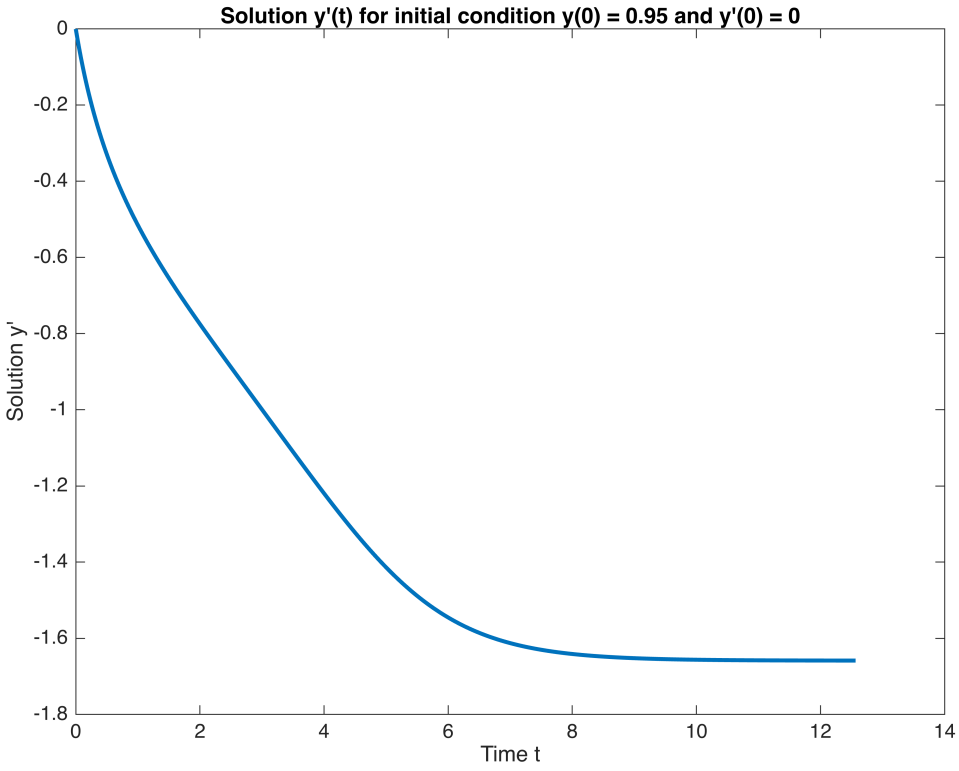

Phase plane trajectory for initial condition y(0) = 0.1 and y'(0) = 0
Solution y'
Solution y
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-14
-12
-10
-8
-6
-4
-2
0
2

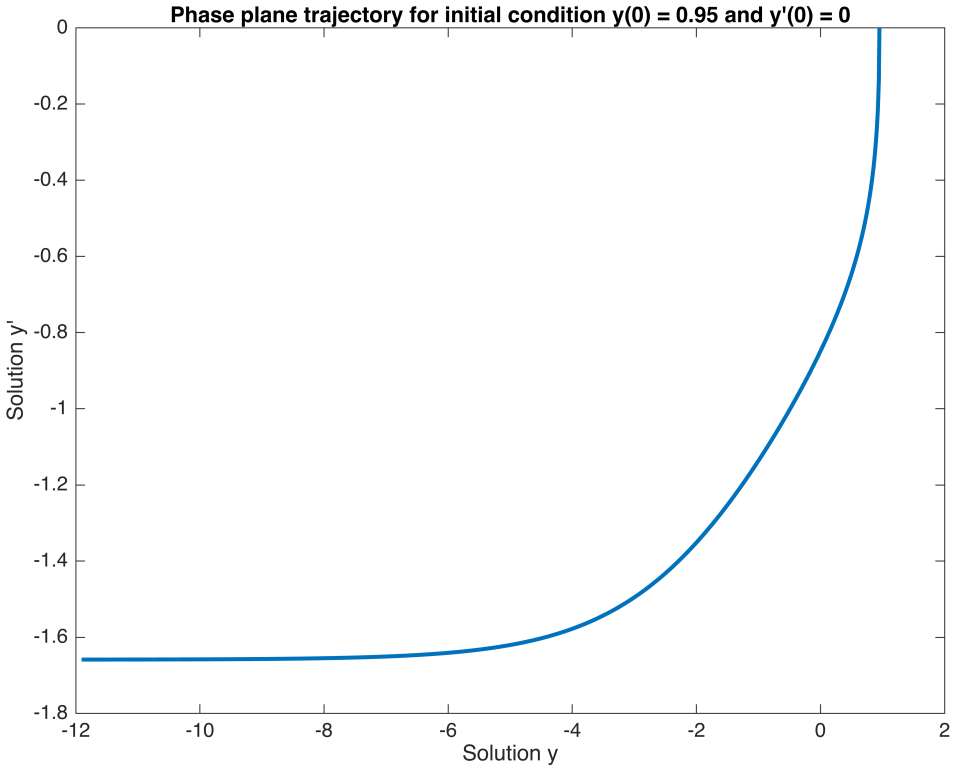

Solution y(t) for initial condition y(0) = 0.75 and y'(0) = 0
Solution y
Time t
2
0
-2
-4
-6
-8
-10
-12
-14
0
2
4
6
8
10
12
14


Solution y'(t) for initial condition y(0) = 0.75 and y'(0) = 0
Solution y'
Time t
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
0
2
4
6
8
10
12
14


Phase plane trajectory for initial condition y(0) = 0.75 and y'(0) = 0
Solution y'
Solution y
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-1.4
-1.6
-1.8
-14
-12
-10
-8
-6
-4
-2
0
2


Solution y(t) for initial condition y(0) = 0.95 and y'(0) = 0
Solution y
Time t
2
0
-2
-4
-6
-8
-10
-12
0
2
4
6
8
10
12
14





Functions Used

```
function dy = ode_func(t, y)
    % This function defines the ODE system to be solved.
    dy = zeros(2,1); % since we have two equations y and y'
    dy(1) = y(2); % y' = y(2)
    dy(2) = -(1 + y(2))^3; % y'' = -(1 + y')^3
end

function [t,u] = rk23(dudt, tspan, u0, tol)
    % RK23 Adaptive IVP solver based on embedded RK formulas.
    % Input:
    %   dudt: defines f in u'(t)=f(t,u) (function)
    %   tspan: endpoints of time interval (2-vector)
    %   u0: initial value (vector, length m)
    %   tol: global error target (positive scalar)
    % Output:
    %   t: selected nodes (vector, length n+1)
    %   u: solution values (array, size m by n+1)

    % Initialize for the first time step.
    t = tspan(1);
    u(:,1) = u0(:);  % make sure u0 is a column vector
    i = 1;
    h = 0.5 * tol^(1/3);  % initial step size
    s1 = dudt(t(1), u(:,1));

    % Time stepping.
    while t(i) < tspan(2)
        % Detect underflow of the step size.
        if t(i) + h == t(i)
            warning('Stepsize too small near t=%.6g.', t(i));
            break;  % quit time stepping loop
        end

        % New RK stages.
        s2 = dudt(t(i) + h/2, u(:,i) + (h/2) * s1);
        s3 = dudt(t(i) + 3*h/4, u(:,i) + (3*h/4) * s2);
        unew2 = u(:,i) + h * (2*s1 + 3*s2 + 4*s3) / 9;  % 2nd order solution
        s4 = dudt(t(i) + h, unew2);

        % Error estimation
        err = h * (-5*s1/72 + s2/12 + s3/9 - s4/8);  % 2nd/3rd order
difference
        E = norm(err, inf);  % error estimate
        maxerr = tol * (1 + norm(u(:,i), inf));  % relative/absolute blend

        % Accept the proposed step?
        if E < maxerr  % if error is within tolerance, accept the step
            t(i+1) = t(i) + h;
            u(:,i+1) = unew2;
            i = i + 1;  % increment the step counter
```

```
        end

        % Adjust step size.
        q = 0.8 * (maxerr/E)^(1/3);  % conservative optimal step factor
        q = min(q, 4);  % limit step size growth
        h = min(q * h, tspan(2) - t(i));  % don't step past the end
    end

    t = t.';  % transpose for output
    u = u.';  % transpose for output
end


function[t,u]=rk4(dudt,tspan, u0, n)
%RK4 Fourth-Order Runge Kutta for an IVP
 % Input:
  % dudt - function handle for the system u'(t) = f(t,u)
  % tspan - vector with start and end time [t0 tf]
  % u0 - initial value (vector, length m)
  % n - number of time steps (integer)
  % Output:
  % t - selected nodes (vector, length n+1)
  % u - solution values (array, (n+1) by m

  %Define Time discretization.
  a = tspan(1); b= tspan(2);
  h=(b-a)/n;
  t= a + (0:n)'*h;

  u = zeros(length(u0), n+1);
  u(:,1) = u0(:);

  %Time stepping
  for i= 1:n
      k1=h*dudt(t(i),    u(:,i));
      k2=h*dudt(t(i)+h/2,    u(:,i)+k1/2);
      k3=h*dudt(t(i)+h/2,    u(:,i)+k2/2);
      k4=h*dudt(t(i)+h,    u(:,i)+k3);
      u(:, i+1) = u(:,i)+ (k1 + 2*(k2+k3)+k4)/6;
  end
  u= u.'; %conform to MATLAB output convention
end

function [t,u] = eulersys(dudt,tspan, u0, n)
  % EULERSYS Euler's method for a system of first order IVP.
  % Input:
  % dudt - function handle for the system u'(t) = f(t,u)
  % tspan - vector with start and end time [t0 tf]
  % u0 - initial condition vector
  % n - number of steps
```

```
  % Output:
  % t - vector of time points
  % u - matrix of solutions, columns correspond to time points

  a = tspan(1); b = tspan(2);
  h = (b - a) / n; % step size
  t = a:h:b; % time vector
  u = zeros(length(u0), n+1); % solution matrix
  u(:,1) = u0; % set initial condition

  for i = 1:n
    f = dudt(t(i), u(:,i)); % evaluate derivative
    u(:,i+1) = u(:,i) + h*f; % Euler update
  end
end

% System of ODEs function
function du = odefcn(t, u)
    du = zeros(size(u));
    du(1) = u(2);
    du(2) = 4*t - 4*u(1);
end
```

```
function[t,u]= eulerivp(dudt,tspan,u0,n)
% EULERIVP Euler's method for a scalar initial-value problem.
%Input:
% dudt defines f in u'(t)= f(t,u). (function)
% tspan endpoints of time interval (2 vector)
% u0 initial value (scalar)
% n number of time steps (integer)
% Output:
% t selected nodes (vector, length n+1)
% u solution values (vector, length n+1)

a = tspan(1); b= tspan(2);
h=(b-a)/n;
t= a + (0:n)'*h;

u= zeros(n+1,1);
u(1)=u0;

for i = 1:n
    u(i+1)= u(i)+ h*dudt(t(i),u(i));
end

end
```