# Homework 2
ASEN 5417: Numerical Methods

Ryan Skinner
Due 2015/09/17

## 1 Introduction

We solve the following problems to better understand numerical techniques for solving initial value problems governed by ordinary differential equations. As will be described in the methods section, our tools primarily consist of second- (RK2) and fourth-order (RK4) Runge-Kutta methods, as well as Euler stability analysis.

### *1.1 Problem 1*

Equations of motion for a rocket's vertical speed $v$ can be written as

$$(m_c + m_p)\frac{dv}{dt} = -(m_c + m_p)g + \dot{m}_p v_e - \tfrac{1}{2}\rho v\,|v|AC_D, \tag{1}$$

where $z$ is the vertical coordinate, $v = dz/dt$ is the vertical speed, and

| | |
|---|---|
| $m_c = 51.02$ kg | (rocket casing mass) |
| $g = 9.8$ m/s$^2$ | (gravitational acceleration) |
| $\rho = 1.23$ kg/m$^3$ | (air density) |
| $A = 0.1$ m$^2$ | (maximum cross-sectional area) |
| $v_e = 360$ m/s | (exhaust speed) |
| $C_D = 0.15$ | (drag coefficient) |
| $m_{p0} = 102.04$ kg | (initial propellant mass). |

Furthermore, the instantaneous propellant mass at time $t$ is given by

$$m_p(t) = m_{p0} - \int_0^t \dot{m}_p dt, \tag{2}$$

and the time-varying burn rate is

$$\dot{m}_p = \frac{m_{p0}}{4} \cdot \begin{cases} t & 0 \le t \le 1 \\ 1 & 1 \le t \le 4 \\ 5-t & 4 \le t \le 5 \\ 0 & 5 \le t. \end{cases} \tag{3}$$

Use a second-order Runge-Kutta (RK2) integrator with $\Delta t = 0.1$ s to plot $z(t)$ and $v(t)$. Use these plots to find the maximum speed, and the height and time at which it is reached; the maximum height, and the time at which it is reached; and the time and velocity when the rocket hits the ground. Check these results with those obtained from MATLAB's `ode45` solver.

This problem is fairly straight-forward. An RK2 scheme is relatively easy to implement, but we will need to be careful when accounting for the time-dependence of $m_p$. Furthermore, though it is not stated in the problem, when $m_p$ reaches zero, the thrust term involving the exhaust speed $v_e$ in (1) needs to "turn off." Lastly, it would be un-physical for the rocket to initially fall through the launch pad before thrust overcomes gravity, and we must account for this when coding the solution.

### *1.2 Problem 2*

Consider the stream function for a two-dimensional jet in self-similar form, which can be written as

$$f''(\eta)+f(\eta)f'(\eta)=0, \qquad f(0)=0, \qquad f'(0)=1, \qquad \Rightarrow f''(0)=0. \tag{4}$$

The velocities in the jet can be obtained via

$$\frac{U}{U_0}=f'(\eta), \qquad \frac{V}{U_0}=\eta f'(\eta)-\tfrac{1}{2}f(\eta). \tag{5}$$

Using numerical integration for $0 \le \eta \le 4$, with a step size of $\Delta\eta = 0.05$, plot the quantities $U/U_0$, $V/V_0$, and $f$ as functions of $\eta$. Perform this analysis with both the second- (custom) and fourth-order (`ode45`) Runge-Kutta methods, and compare their performance. Both solutions may be compared to those obtained from the best-fit expression

$$f'(\eta)=\exp(\gamma\eta^2), \tag{6}$$

where $\gamma = -0.0692$.

### *1.3 Problem 3*

Show that the equation

$$y''=-\frac{19}{4}y-10y', \qquad y(0)=-9, \qquad y'(0)=0, \qquad \Rightarrow y''(0)=\frac{171}{4} \tag{7}$$

is moderately stiff. Use Euler stability analysis to estimate the largest step size $h_{\max}$ for which the Runge-Kutta method will be stable. Then confirm this estimate by computing $y(10)$ using the fourth-order Runge-Kutta method with $h=\{\frac{1}{2}h_{\max}, 2h_{\max}\}$. Compare these solutions with the analytical solution.

## 2 Methodology

### *2.1 Problem 1*

We first re-write the system of equations governing rocket altitude as

$$\begin{aligned} \frac{dz}{dt} &= f(t,z,v) &&= v \\ \frac{dv}{dt} &= g(t,z,v) &&= -g + T\underbrace{\frac{\dot{m}_p v_e}{(m_c+m_p)}}_{\alpha(t)} + \underbrace{\frac{-\rho A C_D}{2(m_c+m_p)}}_{\beta(t)} v\,|v|, \end{aligned} \tag{8}$$

where $\alpha$ and $\beta$ are constants that directly depend only on time, and

$$T = \begin{cases} 1 & 0 \leq t < 5 \\ 0 & \text{all other } t \end{cases} \tag{9}$$

toggles the thrust term on and off based on whether the rocket engines are on. In the MATLAB code, the functions that calculate $\alpha(t)$ and $\beta(t)$ call subsequent functions to calculate $m_p(t)$ and $\dot{m}_p(t)$.

Given the initial values of our variables at $t = 0$, the 2nd-order Runge-Kutta method allows us to integrate the system of equations (8) with respect to time using a time step $h$. At a given time $t_n$, the velocity and position at the next time step, $v_{n+1}$ and $z_{n+1}$, can be approximated by, at each successive time step, calculating

$$\begin{aligned} k_1 &= hf(t_n, z_n, v_n) \\ l_1 &= hg(t_n, z_n, v_n) \\ k_2 &= hf(t_n + \tfrac{1}{2}h, z_n + \tfrac{1}{2}k_1, v_n + \tfrac{1}{2}l_1) \\ l_2 &= hg(t_n + \tfrac{1}{2}h, z_n + \tfrac{1}{2}k_1, v_n + \tfrac{1}{2}l_1) \end{aligned} \tag{10}$$

and then updating the next step's values of $z$ and $v$ as

$$\begin{aligned} z_{n+1} &= z_n + k_2 \\ v_{n+1} &= v_n + l_2. \end{aligned} \tag{11}$$

For early time steps, say $t < 3$, we ensure that the rocket does not fall through the launch pad by setting $v_{n+1}$ and $z_{n+1}$ to zero if they are negative after the RK2 iteration.

### 2.2 Problem 2

We are able to use an analogous method to Problem 1 for the RK2 integrator, and we choose MATLAB's `ode45` function as our higher-order integrator. Normalized velocities are calculated at each value of $\eta$ using the results of integration according to (5). The best-fit expression (6) has an analytical solution for $f$ given by

$$f(\eta) = \sqrt{\pi/4\gamma}\,\mathrm{erf}(\sqrt{\gamma}\eta) \tag{12}$$

where constant of integration for $f(\eta)$ must be zero to match the initial condition. We integrate the variables $f(\eta)$ and $f'(\eta)$, and at each time step use the definition from (4) to calculate $f''(\eta)$.

### 2.3 Problem 3

An analytical solution for (7) exists, and we find it by assuming that $y(t)$ has the form $\exp(\lambda t)$. The values of $\lambda$ allow us to determine how stiff the ODE is.

Euler stability analysis can be performed on the system of equations

$$\frac{d}{dt}\begin{bmatrix} y \\ y' \end{bmatrix} = \mathbf{A}\begin{bmatrix} y \\ y' \end{bmatrix}, \qquad \mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{19}{4} & -10 \end{bmatrix}, \tag{13}$$

by finding the eigenvalues $\lambda_i$ of $\mathbf{A}$. For a forward Euler method at least, the maximum step size for which a numerical solution is stable is given by

$$h_{\max} < \frac{2}{\max_i |\lambda_i|}. \tag{14}$$

For the 4th-order Runge-Kutta method, at each successive time step $t_n$ we calculate

$$\begin{aligned}
\mathbf{k}_1 &= hf(t_n, \mathbf{y}_n) \\
\mathbf{k}_2 &= hf(t_n + \tfrac{1}{2}h, \mathbf{y}_n + \tfrac{1}{2}\mathbf{k}_1) \\
\mathbf{k}_3 &= hf(t_n + \tfrac{1}{2}h, \mathbf{y}_n + \tfrac{1}{2}\mathbf{k}_2) \\
\mathbf{k}_4 &= hf(t_n + \tfrac{1}{2}h, \mathbf{y}_n + \mathbf{k}_3)
\end{aligned} \tag{15}$$

where $\mathbf{y}_n$ is a vector of solution variables evaluated at time $t_n$, and $\mathbf{k}_i$ is a vector of intermediate values for each variable represented by $\mathbf{y}_n$. Finally, we update the next step's values as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \tfrac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right). \tag{16}$$

## 3 Results

### *3.1 Problem 1*

Plots of the velocity and altitude time histories are presented in Figure 1. Statistics on notable flight events are shown in Table 1.

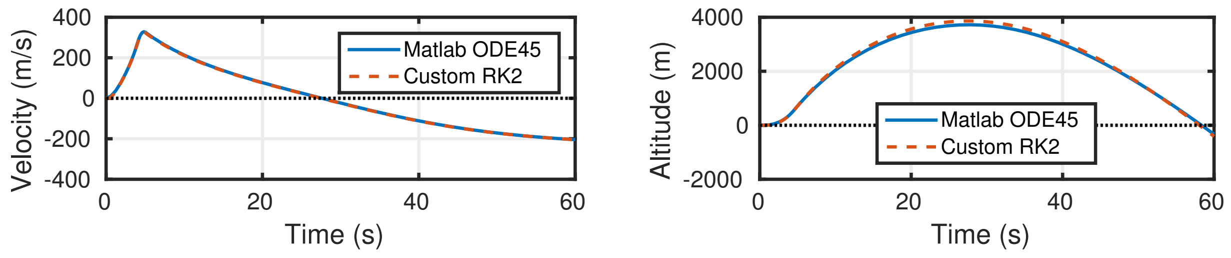

**Figure 1:** Comparison of our own 2nd-order Runge-Kutta scheme to MATLAB's `ode45` solver, for the rocket velocity and altitude.

| Integrator | Max Velocity | | | Max Altitude | | Crash | |
|---|---|---|---|---|---|---|---|
| | Time | Velocity | Altitude | Time | Altitude | Time | Velocity |
| Custom RK2 | 4.80 | 325.74 | 653.47 | 27.50 | 3861.61 | 58.06 | -199.79 |
| MATLAB `ode45` | 4.80 | 328.71 | 648.04 | 27.74 | 3723.21 | 58.34 | -200.19 |

**Table 1:** Comparison of notable flight events between integration methods.

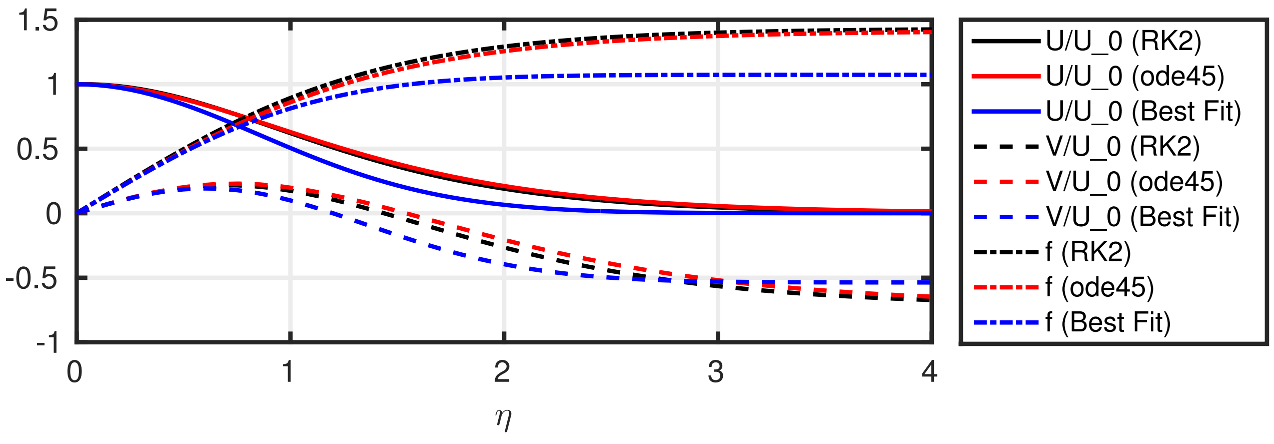

**Figure 2:** Plots of the stream function $f$ and the normalized velocities $U/U_0$ and $V/V_0$ as a function of the similarity variable $\eta$, using two different integration methods and the experimentally-determined best fit for $f'(\eta)$.

### *3.2 Problem 2*

Results of for the stream function and normalized velocities as a function of the similarity variable $\eta$ are presented in Figure 2.

### *3.3 Problem 3*

The analytical solution to (7) is

$$y = -\frac{19}{2}\exp\left(\frac{-x}{2}\right) + \frac{1}{2}\exp\left(\frac{-19x}{2}\right). \tag{17}$$

Since the first exponent has a characteristic time scale of 2, and the second has a characteristic time scale of 2/19, we conclude that the ODE is moderately stiff. Put simply, the second exponential term decays approximately 20× more rapidly than the first, and it will be somewhat difficult to capture both time scales accurately with a fixed step size.

Euler stability analysis yields eigenvalues of $\lambda = \{\frac{-1}{2}, \frac{-19}{2}\}$, and thus our estimate of the largest value of $h$ for which we expect the solution to be stable is

$$h_{\max} = \frac{4}{19}. \tag{18}$$

Values of $y(10)$ are shown in Table 2.

| Method | $y(10)$ |
|---|---|
| Analytical | -0.0640 |
| RK4 ($\frac{1}{2}h_{\max}$) | -0.0419 |
| RK4 ($2h_{\max}$) | $8.2059\times10^{13}$ |

**Table 2:** Comparison of $y(10)$ between the analytical solution and the 4th-order Runge-Kutta method with different step sizes.

## 4 Discussion

### *Problem 1*

Using both integration strategies, we physically see the rocket accelerate at an increasing rate as propellant is burned, and then decelerate as the rocket engine shuts down and the forces of gravity and air drag dominate its dynamics.

Agreement of time-histories of rocket velocity and altitude is strong between the $2^{\text{nd}}$-order Runge-Kutta method we implemented and MATLAB's own `ode45` solver. This holds true both visually and quantitatively, based on Figure 1 and Table 1 respectively. Errors between methods are within 3% for time, within 1% for velocity, and within 9% for altitude.

Any disagreement can be attributed to two phenomena. First, MATLAB's integrator uses an adaptive time-step coupled with a higher-order integration scheme. It is thus able to capture the rocket's take-off behavior more accurately. Since the propellant-burning phase heavily influences the maximum altitude achieved, any small inaccuracies from the RK2 method during that phase will throw off the altitude profile compared to the more-accurate `ode45` solution. Second, position errors are more pronounced than velocity errors, because errors from the velocity calculations are compounded when estimating the altitude at each time step.

### *4.1 Problem 2*

Differences between the Runge-Kutta methods used are qualitatively small. This is likely due to the stream function $f(\eta)$ having relatively small curvature over the domain of interest. If curvatures were more pronounced, the higher-order method would approximate the solution much more accurately.

The experimental best-fit expression for $f'(\eta)$ matches the velocities $U/U_0$ and $V/V_0$ fairly well, but since it is only an exponential function fit to experimental data, it struggles to capture the true behavior of the governing ODE (which is not *exactly* exponential). All things considered, it does a decent job of matching the normalized velocities, though agreement is not perfect.

### *4.2 Problem 3*

Since we are using an RK4 method instead of an Euler method, Euler stability analysis is only a guide. If we violate the $h_{\text{max}}$ criterion, numerical instabilities will persist. If not, however, there is still no guarantee the solution will be stable. This is borne out in our results (Table 2), as the solution with $h = 2h_{\text{max}}$ diverges rapidly. Fortunately, using a step size of $h = \frac{1}{2}h_{\text{max}}$ does not diverge, and produces a value of $y(10)$ within 35% of the analytical value. We would likely need to reduce the step size considerably to get below 1% error, since the governing equation is moderately stiff. This is a situation in which an adaptive step size may be appropriate.

## 5 References

No external references were used other than the course notes for this assignment.

## Appendix: MATLAB Code

The following code listings generate all figures presented in this homework assignment.

**Listing 1:** Problem_1.m

```
function [] = Problem_1()

    %%%%%%
    % Solves the rocket equation using a second-order Runge-Kutta method.
    %   Ryan Skinner, September 2015
    %%%

    Set_Default_Plot_Properties();

    % Define constants.
    c.mc    = 51.02;    % Rocket casing mass
    c.g     = 9.8;      % Gravitational accel
    c.rho   = 1.23;     % Air density
    c.A     = 0.1;      % Max cross-sectional area
    c.ve    = 360;      % Exhaust speed
    c.CD    = 0.15;     % Drag coefficient
    c.mp0   = 102.04;   % Initial propellant mass
    c.v0    = 0;        % Initial velocity
    c.z0    = 0;        % Initial altitude

    % Initialize times at which to evaluate solution.
    t0 = 0;
    tf = 60;
    dt = 0.1;
    t = t0:dt:tf;

    % Initialize velocities and positions.
    v = zeros(length(t),1);
    z = zeros(length(t),1);
    v(1) = c.v0;
    z(1) = c.z0;

    % Perform integration using an RK2 method.
    for n = 1:(length(t)-1)
        tn = t(n);
        vn = v(n);
        zn = z(n);
        k1 = dt * (alpha(c,tn)        + beta(c,tn)        *  vn          * abs(vn));
        l1 = dt * vn;
        k2 = dt * (alpha(c,tn + dt/2) + beta(c,tn + dt/2) * (vn + k1/2) * abs(vn + k1/2));
        l2 = dt * (vn + l1/2);
        v(n+1) = vn + k2;
        z(n+1) = zn + l2;
        % Prevent rocket from falling into the ground before thrust overcomes gravity.
        if n * dt < 3
            if v(n+1) < 0
                v(n+1) = 0;
            end
            if z(n+1) < 0
                z(n+1) = 0;
            end
        end
    end

    % Perform validation using Matlab's ODE45.
    t_span = [t0, tf];
    initials = [v(1), z(1)];
    [t_ml, sol_ml] = ode45(@(t,y) rocket(t,y,c), t_span, initials);
    v_ml = sol_ml(:,1);
    z_ml = sol_ml(:,2);

    % Plot velocity.
    figure();
    hold on;
    plot(t_ml,v_ml,'DisplayName','Matlab ODE45');
    plot(t,v,'--','DisplayName','Custom RK2');
    legend('show');
    plot(t_span,[0,0],'k:');
    xlim(t_span);
    xlabel('Time (s)');
    ylabel('Velocity (m/s)');

    % Plot altitude.
    figure();
    hold on;
    plot(t_ml,z_ml,'DisplayName','Matlab ODE45');
    plot(t,z,'--','DisplayName','Custom RK2');
    hleg = legend('show');
    set(hleg,'location','south');
    plot(t_span,[0,0],'k:');
    xlim(t_span);
    xlabel('Time (s)');
    ylabel('Altitude (m)');

    % Stats on rocket flight (Custom RK2)
    i = find(v == max(v));
    fprintf('RK2  : Max velocity is %.2f at time %.2f and height %.2f.\n',max(v),t(i),z(i));
    i = find(z == max(z));
    fprintf('RK2  : Max altitude is %.2f at time %.2f.\n',max(z),t(i));
    ignore = 100;
    t_crash = interp1(z(ignore:end),t(ignore:end),0);
    fprintf('RK2  : Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t,v,t_crash));

    % Stats on rocket flight (Matlab's ODE45)
    i = find(v_ml == max(v_ml));
    fprintf('ODE45: Max velocity is %.2f at time %.2f and height %.2f.\n',max(v_ml),t_ml(i),z_ml(i));
    i = find(z_ml == max(z_ml));
    fprintf('ODE45: Max altitude is %.2f at time %.2f.\n',max(z_ml),t_ml(i));
    ignore = 50;
    t_crash = interp1(z_ml(ignore:end),t_ml(ignore:end),0);
    fprintf('ODE45: Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t_ml,v_ml,t_crash));

end

function [ mp_dot ] = mp_dot ( c, t )
% Calculates time derivative of propellant mass.
% This is the exact value of mp(t) given in the problem statement.
    if 0 <= t && t < 1
        mp_dot = t;
    elseif 1 <= t && t < 4
        mp_dot = 1;
    elseif 4 <= t && t < 5
        mp_dot = 5-t;
    else
        mp_dot = 0;
    end
    mp_dot = mp_dot * c.mp0 / 4;
end

function [ mp ] = mp ( c, t )
% Calculates instantaneous propellant mass.
% This is the exact value of mp(t) given in the problem statement.
    if t < 0
        intg = 0;
    elseif 0 <= t && t < 1
        intg = 0.5 * t^2;
    elseif 1 <= t && t < 4
        intg = t - 0.5;
    elseif 4 <= t && t < 5
        intg = 3.5 + (t-4) - 0.5 * (t-4)^2;
    else
```

```
        intg = 4;
    end
    mp = c.mp0 - intg * c.mp0 / 4;
end

function [ a ] = alpha( c, t )
% Calculates the time-dependent constant alpha.
    engines_on = 0 <= t && t < 5;
    a = -c.g + (engines_on * c.ve) * mp_dot(c,t) / (mp(c,t) + c.mc);
end

function [ b ] = beta( c, t )
% Calculates the time-dependent constant beta.
    b = -0.5 * c.rho * c.A * c.CD / (mp(c,t) + c.mc);
end

function [ dy ] = rocket( t, y, c )
% Calculates values for ODE45-based integration of the rocket equation.
    dy = zeros(2,1);
    dy(1) = alpha(c,t) + beta(c,t) * y(1) * abs(y(1));
    dy(2) = y(1);
    % Prevent rocket from falling into the ground before thrust overcomes gravity.
    if t < 3
        if dy(1) < 0
            dy(1) = 0;
        end
        if dy(2) < 0
            dy(2) = 0;
        end
    end
end
```

**Listing 2:** Problem_2.m

```
function [] = Problem_2()

    %%%%%%
    % Solves the stream function for a 2D jet in self-similar form, using RK2 and ode45.
    %   Ryan Skinner, September 2015
    %%%

    Set_Default_Plot_Properties();

    % Initialize times at which to evaluate solution.
    eta0 = 0;
    etaf = 4;
    deta = 0.05;
    rk2.eta  = (eta0:deta:etaf)';

    % Initialize velocities and positions.
    rk2.f   = zeros(length(rk2.eta),1);
    rk2.fp  = zeros(length(rk2.eta),1);
    rk2.fpp = zeros(length(rk2.eta),1);
    rk2.f(1)   = 0;
    rk2.fp(1)  = 1;
    rk2.fpp(1) = 0;

    % Perform integration using an RK2 method.
    for n = 1:(length(rk2.eta)-1)
        f   = rk2.f(n);    % k
        fp  = rk2.fp(n);   % l
        fpp = rk2.fpp(n);
        k1 = deta * fp;
        l1 = deta * fpp;
        k2 = deta * (fp + k1/2);
        l2 = deta * (fpp + l1/2);
        rk2.f(n+1)   = f + k2;
        rk2.fp(n+1)  = fp + l2;
        rk2.fpp(n+1) = -rk2.f(n+1) * rk2.fp(n+1);
    end
    rk2.uou0 = rk2.fp;
    rk2.vou0 = rk2.eta .* rk2.fp - 0.5 * rk2.f;

    % Repeat integration using a higher-order method (ode45).
    [T, Y] = ode45(@stream, [0,4], [0,1]);
    o45.eta = T;
    o45.f   = Y(:,1);
    o45.fp  = Y(:,2);
    o45.fpp = -o45.f .* o45.fp;
    o45.uou0 = o45.fp;
    o45.vou0 = o45.eta .* o45.fp - 0.5 * o45.f;

    % Calculate solution using best-fit expression.
    bst.eta = rk2.eta;
    bst.f   = 1.07313 * erf(0.825833 * bst.eta);
    bst.fp  =                     exp(-0.682*bst.eta.^2);
    bst.uou0 = bst.fp;
    bst.vou0 = bst.eta .* bst.fp - 0.5 * bst.f;

    % Plot fields.
    figure();
    hold on;
    plot(rk2.eta, rk2.uou0,  'k-', 'DisplayName', 'U/U_0 (RK2)');
    plot(o45.eta, o45.uou0,  'r-', 'DisplayName', 'U/U_0 (ode45)');
    plot(bst.eta, bst.uou0,  'b-', 'DisplayName', 'U/U_0 (Best Fit)');
    plot(rk2.eta, rk2.vou0, 'k--', 'DisplayName', 'V/U_0 (RK2)');
    plot(o45.eta, o45.vou0, 'r--', 'DisplayName', 'V/U_0 (ode45)');
    plot(bst.eta, bst.vou0, 'b--', 'DisplayName', 'V/U_0 (Best Fit)');
    plot(rk2.eta, rk2.f,    'k-.', 'DisplayName', 'f (RK2)');
    plot(o45.eta, o45.f,    'r-.', 'DisplayName', 'f (ode45)');
    plot(bst.eta, bst.f,    'b-.', 'DisplayName', 'f (Best Fit)');
    hleg = legend('show');
    set(hleg, 'location', 'eastoutside');
    xlim([eta0,etaf]);
    ylim([-1,1.5]);
    xlabel('\eta');

end

function dy = stream(~, y)
    dy = zeros(2,1);
    dy(1) = y(2);
    dy(2) = -y(1)*y(2);
end
```

**Listing 3:** Problem_3.m

```
function [] = Problem_3()

    %%%%%%
    % Solves a stiff ODE using an RK4 method and two different step sizes.
    %   Ryan Skinner, September 2015
    %%%

    Set_Default_Plot_Properties();

    % Initialize times at which to evaluate solution (with dt = hmax*2).
    t0 = 0;
    tf = 15;
    dt = 8/19;
    t  = (t0:dt:tf)';

    % Initialize velocities and positions.
    y   = zeros(length(t),1);
    yp  = zeros(length(t),1);
    ypp = zeros(length(t),1);
    y(1)   = -9;
    yp(1)  = 0;
```

```matlab
    ypp(1) = 42.75;

    % Perform integration using an RK4 method.
    for n = 1:(length(t)-1)
        yn   = y(n);    % k
        ypn  = yp(n);   % l
        yppn = ypp(n);  % m
        k1 = dt * ypn;
        l1 = dt * yppn;
        k2 = dt * (ypn + k1/2);
        l2 = dt * (yppn + l1/2);
        k3 = dt * (ypn + k2/2);
        l3 = dt * (yppn + l2/2);
        k4 = dt * (ypn + k3);
        l4 = dt * (yppn + l3);
        y(n+1)   = yn   + k1/6 + k2/3 + k3/3 + k4/6;
        yp(n+1)  = ypn  + l1/6 + l2/3 + l3/3 + l4/6;
        ypp(n+1) = (-19/4) * y(n+1) - 10 * yp(n+1);
    end
    hmt2.t = t;
    hmt2.y = y;
    hmt2.y10 = interp1(hmt2.t,hmt2.y,10);

    % Initialize times at which to evaluate solution (with dt = hmax/2).
    t0 = 0;
    tf = 15;
    dt = 2/19;
    t  = (t0:dt:tf)';

    % Initialize velocities and positions.
    y   = zeros(length(t),1);
    yp  = zeros(length(t),1);
    ypp = zeros(length(t),1);
    y(1)   = -9;
    yp(1)  = 0;
    ypp(1) = 42.75;

    % Perform integration using an RK4 method.
    for n = 1:(length(t)-1)
        yn   = y(n);    % k
        ypn  = yp(n);   % l
        yppn = ypp(n);  % m
        k1 = dt * ypn;
        l1 = dt * yppn;
        k2 = dt * (ypn + k1/2);
        l2 = dt * (yppn + l1/2);
        k3 = dt * (ypn + k2/2);
        l3 = dt * (yppn + l2/2);
        k4 = dt * (ypn + k3);
        l4 = dt * (yppn + l3);
        y(n+1)   = yn   + k1/6 + k2/3 + k3/3 + k4/6;
        yp(n+1)  = ypn  + l1/6 + l2/3 + l3/3 + l4/6;
        ypp(n+1) = (-19/4) * y(n+1) - 10 * yp(n+1);
    end
    hmd2.t = t;
    hmd2.y = y;
    hmd2.y10 = interp1(hmd2.t,hmd2.y,10);

    % Calculate solution analytically.
    ya = (1/2) * exp(-19*hmd2.t/2) - (19/2) * exp(-hmd2.t/2);
    ya10 = (1/2) * exp(-19*10/2) - (19/2) * exp(-10/2);

    % Plot fields.
    figure();
    hold on;
    plot(hmd2.t,    ya,     'DisplayName','analytic');
    plot(hmt2.t,hmt2.y,'--','DisplayName','RK4 (hmax*2)');
    plot(hmd2.t,hmd2.y,'-.','DisplayName','RK4 (hmax/2)');
    hleg = legend('show');
    set(hleg, 'location', 'eastoutside');
    xlim([0,10]);
    ylim([-10,0]);
    xlabel('\eta');
    ylabel('y');

    % Print statistics.
    fprintf('Values of y(10) for each method:\n');
    fprintf('Actual      : %.4f\n',ya10);
    fprintf('RK4 (hmax/2): %.4f\n',hmd2.y10);
    fprintf('RK4 (hmax*2): %.4f\n',hmt2.y10);

end
```