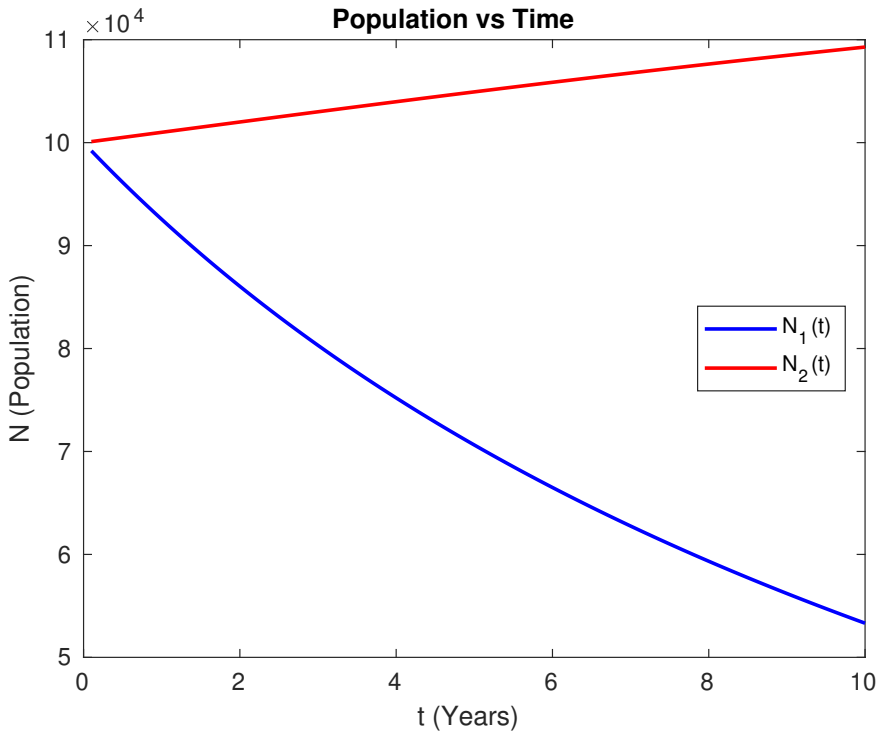
Q1. The solution obtained for a time-step of $h = 0.0001$ is assumed to be the "exact" solution. This assumption is made on the basis of following results.

1. For $h = 0.001$, $N = [N_1(10), N_2(10)] = [0.533177936172460, 1.092840108399847]x10^5$
2. For $h = 0.0001$, $N = [N_1(10), N_2(10)] = [0.533177936172449, 1.092840108399842]x10^5$
3. For $h = 0.00001$, $N = [N_1(10), N_2(10)] = [0.533177936172467, 1.092840108399861]x10^5$

The solution is plotted below for the time-step of $h = 0.0001$.



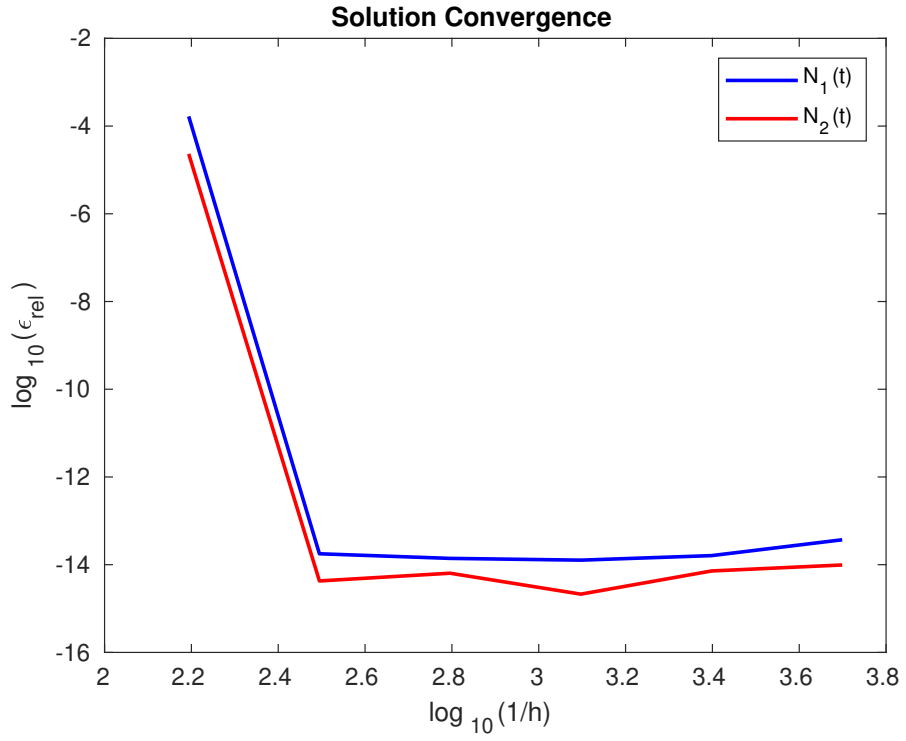
The solution makes sense because the death rate due to competition for food is greater for species 1 and hence N_1 declines with time whereas N_2 increases.

Now, considering the above solution to be the exact solution, the solution is computed again by increasing the time-step by a factor of 2, i.e for

$$h = 0.0001x[2, 4, 8, 16, 32, 64]$$

The logarithm of relative error, $log_{10}\epsilon_{rel}$ is plotted against the $log_{10}(1/h)$ and the results are shown in the figure below.

We see that the change in step-size do not have any significant effect on the errors except when the step-size is changed nearly by a factor of 100 in which case, the relative error improves almost by an order of 3.



Theoretically, the 4th order Runge-Kutta method has a truncation error of $O(h^4)$. If the time-step is changed by a factor of 100, we expect an improvement in error by a factor of 8, whereas the obtained improvement is a factor of 3. Also, when the time-step is increased by a factor of 10, the error should increase by a factor of 4, which does not happen here. Instead, it is more or less the same.

The characteristics of the solution obtained DOES NOT agree well with our theoretical expectations.

Q2. Let us consider the stability of the general equation

$$\frac{dy}{dt} = \lambda y$$

with $\lambda = -3$ as the specific value for the problem.

The exact solution to the differential equation is $y(t) = c\lambda e^{\lambda t}$, and the function $f(t, y) = \lambda y$ is continuous and satisfies a Lipschitz condition in the variable y with a Lipschitz constant $L = \lambda$(if $\lambda > 0$) or $L = 1$(if $\lambda \leq 0$) on the set $D = \{(t, y)|0 \leq t < \infty$ and $-\infty < y < \infty\}$.

RK2/MIDPOINT METHOD: Assuming $y(0) = \alpha$ so that the exact solution is $y(t) = \alpha e^{\lambda t}$, and the step-size to be h, such that $t_i = hi$ where $i = 0, 1, 2, ...$, the Midpoint method is defined by the following equation.

$$\begin{aligned} w_{i+1} &= w_i + hf\left(t_i + h/2, w_i + \frac{h}{2} f(t_i, w_i)\right) \\ &= w_i + h\lambda\left(w_i + \frac{\lambda h}{2} w_i\right) \\ &= w_i\left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right) \end{aligned}$$

which implies that

$$\begin{aligned} w_{i+1} &= w_0\left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^{i+1} \\ w_{i+1} &= \alpha\left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^{i+1} \end{aligned}$$

Then, the absolute error is

$$\begin{aligned} \epsilon_i &= |y(t_i) - w_i| \\ &= \left|e^{\lambda hi} - \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^{i+1}\right| |\alpha| \end{aligned}$$

For the solution to be accurate, the error must decay as i increases. The term $e^{\lambda hi}$ will decay if $\lambda < 0$ which is true for the given DE since $\lambda = -3$. The other term

$$\left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^{i+1}$$

will decay as i increases if and only if

$$\left|\left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)\right| < 1$$

which implies that

$$-2 < h\lambda + \frac{(h\lambda)^2}{2} < 0$$

or

$$-4 < 2h\lambda + (h\lambda)^2 < 0$$

The left inequality is always true since $\lambda^2 h^2 + 2h\lambda + 4 > 0$ for all h. The right inequality gives us

$$\begin{aligned} 2h\lambda + (h\lambda)^2 &< 0 \\ \implies h &< -2/\lambda \end{aligned}$$

Hence, the step-size limit is $h < 2/|\lambda|$, which in our case is

$$\boxed{h < \frac{2}{3}}$$

Suppose now we introduce a round-off error in the initial condition, i.e.

$$w_0 = \alpha + \delta_0$$

Then, at the i^{th} step, the round-off error is

$$\delta_i = \delta_0 \left(1 + h\lambda + \frac{(h\lambda)^2}{2}\right)^{i+1}$$

which again does not grow if $\lambda < 0$ and $h < 2/|\lambda|$.

Hence, the stability limit for the Midpoint method is

$$\boxed{h < \frac{2}{3}}$$

ADAMS-BASHFORTH TECHNIQUE: Once again, assuming $y(0) = \alpha$ so that the exact solution is $y(t) = \alpha e^{\lambda t}$, and the step-size to be h, such that $t_i = hi$ where $i = 0, 1, 2, 3, 4, ...$, the 2-step Adams-Bashforth technique is defined by the following equation.

$$w_{i+1} = w_i + \frac{h}{2}[3f(t_i, w_i) - f(t_{i-1}, w_{i-1})]$$

The difference equation is

$$w_{i+1} - \left(1 + \frac{3h\lambda}{2}\right) w_i + \left(\frac{h\lambda}{2}\right) w_{i-1} = 0$$

The characteristic polynomial for the difference equation is

$$z^2 - \left(1 + \frac{3h\lambda}{2}\right) z + \left(\frac{h\lambda}{2}\right) = 0$$

Substituting $\lambda = -3$, we get

$$z^2 + \left(\frac{9h}{2} - 1\right) z - \frac{3h}{2} = 0$$

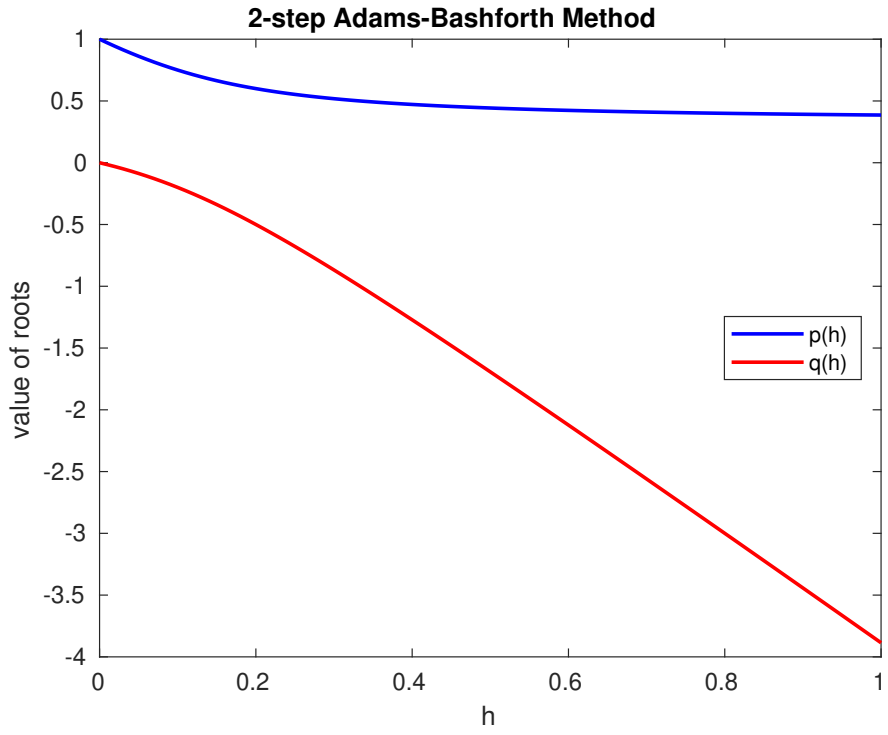
For stability, all zeros $z = p, q$ of this equation must satisfy $|z| < 1$.

The roots are

$$p = \frac{1}{2} - \frac{9h}{4} + \frac{\sqrt{\frac{81h^2}{4} - 3h + 1}}{2}$$

$$q = \frac{1}{2} - \frac{9h}{4} - \frac{\sqrt{\frac{81h^2}{4} - 3h + 1}}{2}$$

The roots are plotted as a function of h below.



It can be seen from the plots that the first root p has a magnitude less than 1 for all positive $h < 1$, but the root q has a magnitude less than 1 only for $0 < h < \frac{1}{3}$ (obtained by solving $q(h) = -1$). Hence, the 2-step Adams-Bashforth Method is only stable for

$$\boxed{h < \frac{1}{3}}$$

ADAMS-MOULTON TECHNIQUE: Once again, assuming $y(0) = \alpha$ so that the exact solution is $y(t) = \alpha e^{\lambda t}$, and the step-size to be h, such that $t_i = hi$ where $i = 0, 1, 2, 3, 4, ...$, the Adams-MOULTON technique is defined by the following equation.

$$\begin{aligned} w_{i+1} &= w_i + \frac{h}{12}[5f(t_{i+1}, w_{i+1}) + 8f(t_i, w_i) - f(t_{i-1}, w_{i-1})] \\ &= \frac{5h\lambda}{12} w_{i+1} + \left(1 + \frac{8h\lambda}{12}\right) w_i - \left(\frac{h\lambda}{12}\right) w_{i-1} \end{aligned}$$

The difference equation is

$$w_{i+1}\left(1 - \frac{5h\lambda}{12}\right) - \left(1 + \frac{8h\lambda}{12}\right) w_i + \left(\frac{h\lambda}{12}\right) w_{i-1} = 0$$

The characteristic polynomial for the difference equation is

$$z^2\left(1 - \frac{5h\lambda}{12}\right) - \left(1 + \frac{8h\lambda}{12}\right) z + \left(\frac{h\lambda}{12}\right) = 0$$

Substituting $\lambda = -3$, we get

$$z^2\left(1 + \frac{5h}{4}\right) - (1 - 2h)\, z - \frac{h}{4} = 0$$

For stability, all zeros $z = p, q$ of this equation must satisfy $|z| < 1$.

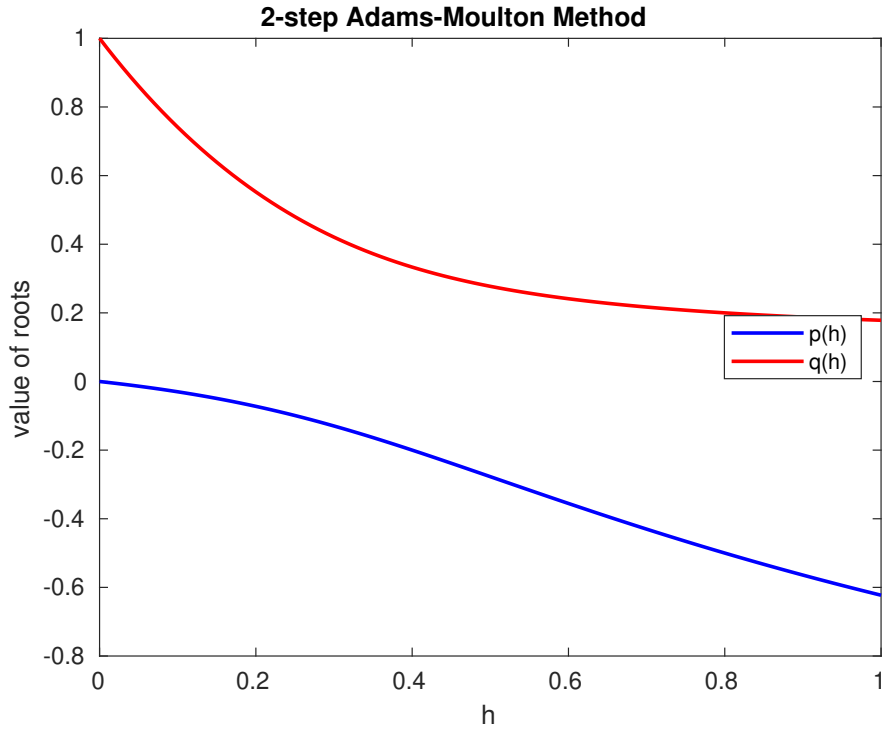
The roots are

$$\begin{aligned} p &= \frac{-2\left(\sqrt{\frac{21h^2}{4} - 3h + 1} + 2h - 1\right)}{5h + 4} \\ q &= \frac{2\left(\sqrt{\frac{21h^2}{4} - 3h + 1} - 2h + 1\right)}{5h + 4} \end{aligned}$$

The roots are plotted as a function of h below.

It can be seen from the plots that both the roots have a magnitude less than 1 for all $h < 1$. Hence, the 2-step Adams-Bashforth Method is stable for

$$\boxed{h < 1}$$



ADAMS PREDICTOR-CORRECTOR METHOD: Since this method relies on the Adams-Bashforth and Adams-Moulton methods for prediction and correction respectively, it is only as stable as the least stable part which is the Adams-Bashforth method, which is only stable as long as

$$\boxed{h < \frac{1}{3}}$$

Thus, the methods can be ranked in the order of stability as follows:

$$\boxed{\text{Adams-Moulton} > \text{Midpoint Method} > \text{Adams-Bashforth} = \text{Adams Predictor/Corrector}}$$

VERIFICATION: We verify these results by implementing a Predictor-Corrector algorithm to solve the differential equation with varying step-sizes. The selected values are

$$h = 1/4, 1/3.1, 1/3, 1/2.9 \text{ and } 2/5.$$

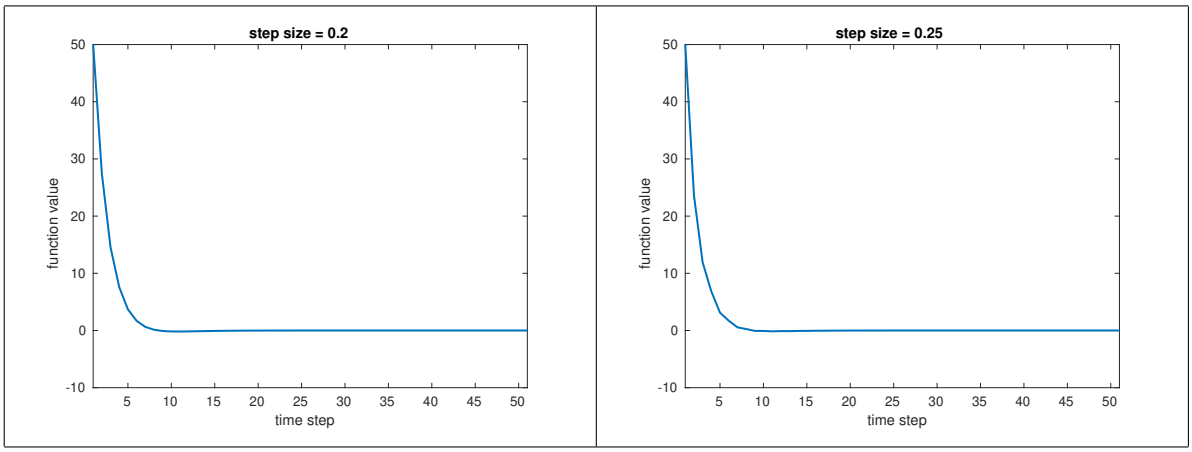
The results are shown below for 50 time steps.

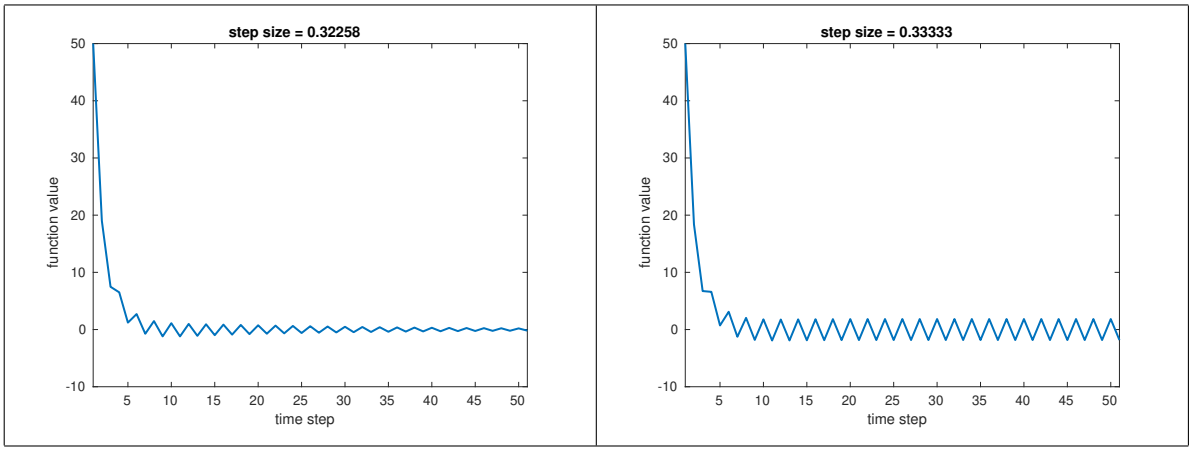
From the figures shown, it can be easily seen that

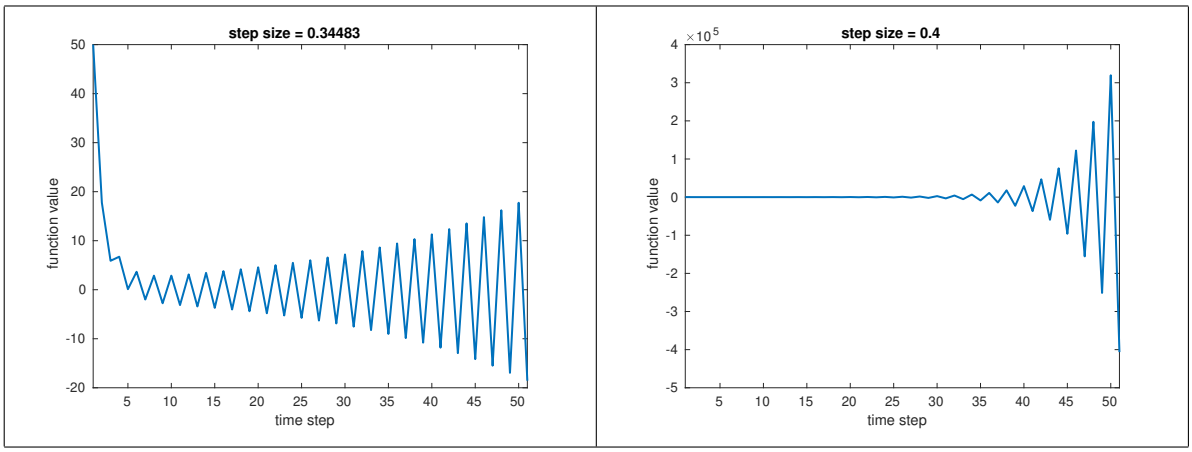
1. The algorithm is stable for any value of h such that $h < 1/3$.
2. At values slightly below $1/3$, the values at far away time steps oscillate about a mean value and eventually settle down. The process is stable but the solution might not be accurate.

3. At values slightly above $1/3$, the oscillations increase in magnitude and the results get skewed more and more from the true value.

4. At values above $1/3$, the algorithm breaks down completely with errors as large as $O(10^5)$.







MATLAB Code - Q1

filename - hw6p1.m

```
%
% DS 288 Numerical Methods
% Homework 6, Problem 1
% Nikhil Jayswal, SR - 16961
%

clc; close all
clear;

% intial conditions
N0 = [1e5 1e5];     % N0 = [N_1(0) N_2(0)]

% constants
A = [1e-1 1e-1];    % A = [A_1 A_2]
B = [8e-7 8e-7];    % B = [B_1 B_2]
C = [1e-6 1e-7];    % C = [C_1 C_2]

% time period
tbegin = 0;
tend = 10;

% solve the system of first order DEs
% [dN/dt] = [dN_1/dt dN_2/dt]
%         = A.*N - B.*N.*N - C*N(1)*N(2)*ones(1, 2)
% 4th order RK

% step size
% change this to find the time step for
% the "exact" values
dt = 1e-4;

% time vector
time = tbegin:dt:tend;

% output vector
global N
N = zeros(length(time), length(N0));
```

```
% call helper function that implements
% a 4th order RK method
% flag to distinguish between use cases
flag = 1;
hw6p1helper(tbegin, tend, dt, N0, A, B, C, flag);

% plot N_1(t) and N_2(t)
figure
num_data_points = 100;
x = zeros(num_data_points, 1);
y = zeros(num_data_points, length(N0));
divisor = (length(time)-1)/num_data_points;
index = 1;
for i = 1:length(time)
    if rem(i, divisor) == 0
        x(index) = time(i);
        y(index, :) = N(i, :);
        index = index + 1;
    end
end
plot(x, y(:, 1),'b', 'LineWidth', 1.5)
hold on
plot(x, y(:, 2), 'r', 'LineWidth', 1.5)
hold off
xlabel('t (Years)')
ylabel('N (Population)')
title('Population vs Time')
legend('N_1(t)', 'N_2(t)', 'Location', 'east')

%% error analysis

% time step for "exact" value
dtexact = dt;

dt = dtexact*[2 4 8 16 32 64];
logerror = zeros(length(dt), length(N0));
flag = 0;

for i = 1:length(dt)
    W = hw6p1helper(tbegin, tend, dt(i), N0, A, B, C, 0);
```

```
    logerror(i, :) = log10(abs(N(end, :) - W(end, :))./N(end, :));
end

% plot errors vs step size
figure
logdt = log10(1./dt);
plot(logdt, logerror(:, 1), 'b', 'LineWidth', 1.5)
hold on
plot(logdt, logerror(:, 2), 'r', 'LineWidth', 1.5)
xlabel('log_{10}(1/h)')
ylabel('log_{10}(\epsilon_{rel})')
title('Solution Convergence')
legend('N_1(t)', 'N_2(t)')
hold off
```

filename - hw6p1helper.m

```
function [W] = hw6p1helper(tbegin, tend, dt, N0, A, B, C, flag)
    % time vector
    time = tbegin:dt:tend;

    global N

    if flag == 1
        % initialize
        t = tbegin;
        N(1, :) = N0;

        % iterate to find N
        for i = 1:length(time)-1
            % NOTE - this formulation is specific to this problem
            W  = N(i, :);
            k1 = dt*(A.*W - B.*(W.^2) - C*prod(W));
            W  = N(i, :) + k1/2;
            k2 = dt*(A.*W - B.*(W.^2) - C*prod(W));
            W  = N(i, :) + k2/2;
            k3 = dt*(A.*W - B.*(W.^2) - C*prod(W));
            W  = N(i, :) + k3;
            k4 = dt*(A.*W - B.*(W.^2) - C*prod(W));
            W  = N(i, :) + (k1 + 2*k2 + 2*k3 + k4)/6;
            t  = t + i*dt;
```

```
            N(i+1, :) = W;
        end
    else
        % initialize
        t = tbegin;
        W = zeros(length(time), length(N0));
        W(1, :) = N0;

        % iterate to find N
        for i = 1:length(time)-1
            % NOTE - this formulation is specific to this problem
            U  = W(i, :);
            k1 = dt*(A.*U - B.*(U.^2) - C*prod(U));
            U  = W(i, :) + k1/2;
            k2 = dt*(A.*U - B.*(U.^2) - C*prod(U));
            U  = W(i, :) + k2/2;
            k3 = dt*(A.*U - B.*(U.^2) - C*prod(U));
            U  = W(i, :) + k3;
            k4 = dt*(A.*U - B.*(U.^2) - C*prod(U));
            U  = W(i, :) + (k1 + 2*k2 + 2*k3 + k4)/6;
            t  = t + i*dt;
            W(i+1, :) = U;
        end
    end
end
```

MATLAB Code - Q2

filename - hw6p2.m

```
%
% DS 288 Numerical Methods
% Homework 6, Problem 2
% Nikhil Jayswal, SR - 16961
%

clc; close all
clear;

% Predictor - Corrector Scheme
% Predictor - 2-step Adams-Bashforth
```

```
% Corrector - 2-step Adams-Moulton

% dy/dt = f(t, y)
% f(t, y) = -3y
f = @(z) -3*z;

% intial conditions
t0 = 0;
y0 = 50.0;

% exact solution
y = @(t) 50*exp(-3*t);

% step sizes
steps = [1/5 1/4 1/3.1 1/3 1/2.9 2/5];

% number of time steps
N = 50;

% intialise output
w = zeros(N+1, length(steps));
w(1, :) = y0;
w(2, :) = y(steps);

% Predictor
for i = 2:N
    for j = 1:length(steps)
        h = steps(j);
        w(i+1, j) = w(i, j) + (h/2)*(3*f(w(i, j)) - f(w(i-1, j)));
    end
end

% Corrector
for i = 2:N
    for j = 1:length(steps)
        h = steps(j);
        w(i+1, j) = w(i) + (h/12)*(5*f(w(i+1, j)) + 8*f(w(i)) - f(w(i-1)));
    end
end

% plot all solutions
```

```
for j = 1:length(steps)
    figure
    time = 1:N+1;
    plot(time, w(:, j), 'linewidth', 1.5)
    xlabel('time step')
    ylabel('function value')
    xlim([1, N+1])
    titlestring = ['step size = ', num2str(steps(j))];
    title(titlestring)
end
```