```
% function handle for logisticPop.m
f = @logisticPop;

% pass logisticPop to eulersMethod with step size 0.5, 0.1, and 0.01
popLow = eulersMethod(f, 0.5, 0, 30, 6);
popMed = eulersMethod(f, 0.1, 0, 30, 6);
popHigh = eulersMethod(f, 0.01, 0, 30, 6);

% calculate analytical solution
actual_population = arrayfun(@ivp, linspace(0,30,61));

% calcualte errors for the three step sizes used to calculate this problem
errorpopLow = absoluteError(popLow(2, :), actual_population);
errorpopMed = absoluteError(popMed(2, :), arrayfun(@ivp, ...
    linspace(0,30,301)));
errorpopHigh = absoluteError(popHigh(2, :), arrayfun(@ivp, ...
    linspace(0,30,3001)));

% plot figure that models mountain lion population
figure
plot(popLow(1, :), popLow(2, :), popMed(1, :), popMed(2, :), ':', ...
    popHigh(1, :), popHigh(2, :), '--', linspace(0,30,61), ...
    actual_population, '-.')
title('Mountain Lion Population')
xlabel('Time (years)')
ylabel('Population (Dozens)')
legend('Euler: h= 0.5', 'Euler: h= 0.1', 'Euler: h = 0.01', ...
    'Exact Solution')

%plot figure that puts the abolsute error of the numerical solutions on
%the y-axis
figure
semilogy(linspace(0,30,61), errorpopLow, linspace(0,30,301), errorpopMed, ...
    linspace(0,30,3001), errorpopHigh)
title('Absolute Error of Numerical Solutions')
xlabel('Time (years)')
ylabel('Log(Absolute Error)')
legend('Error: h= 0.5', 'Error: h= 0.1', 'Error: h = 0.01')

%function handle for the logistic harvesting equation
f2 = @logisticHarvesting;

% find the threee equilibrium solutions
z0 = fzero(f2, 0.8);
z1 = fzero(f2, 1.85);
z2 = fzero(f2, 5.44);

% pass logistic harvestigng to eulers method with four different population
% sizes
pop84 = eulersMethod(f2, 0.1, 0, 30, 84);
pop24 = eulersMethod(f2, 0.1, 0, 30, 24);
pop18 = eulersMethod(f2, 0.1, 0, 30, 18);
```

```
pop6 = eulersMethod(f2, 0.1, 0, 30, 6);

% plot figure that has the directional field, the four solutions to the
% logistic harvesting function, and the equilibirum solutions
figure
hold on
plot(pop84(1, :), pop84(2, :), pop24(1, :), pop24(2, :), ':', pop18(1, :), ...
    pop18(2, :), '--', pop6(1, :), pop6(2, :), '-.')
dirfield(f2, 0:1:30, 0:0.5:10, 'Mule Deer Population')
yline(z0)
yline(z1)
yline(z2)
xlabel('Time (years)')
ylabel('Population')
legend('Euler: h= 0.5', 'Euler: h= 0.1', 'Euler: h = 0.01', ...
    'Actual Solution')

% function handle for harvesting equation
h = @harvesting;

% simulate harvesting equation as population grows exponentially large
harvesting_simulation = arrayfun(h, linspace(1, 10000, 1000));

% plot simualtion of harvesting function and p value
figure
plot(linspace(0, 1.0, 1000), harvesting_simulation);
yline(1.2)
xlabel('Prey Population Size (Dozens)')
ylabel('Prey Harvested (Dozens)')
title('Harvesting Function as Prey Population Increases')
axis([-0.005 0.12 1.195 1.201])
```

plot equilibrium solutions for lotka volterra system in question 3.2.2

```
figure
hold on
equX = [(2.5/1.4)];
equY = [(1.5/1.1)];
scatter(equX, equY,'g', 'filled')
scatter(0,0, 'g')
flow
xline(0, 'red')
yline(0, 'black')
yline((1.5/1.1), 'red')
xline((2.5/1.4), 'black')

% fucntion handle for lotkla volterra
f3 = @lotkaVolterra;

% pass lotka volterra to ODE45, varibale is a 2 by x array with time series
% and popualtion information
[t, P] = ode45(f3, [0,30], [0.5, 1]);

% plot phase portrait
```

```
figure
flow
hold on
plot(P(:,1), P(:,2))

% plot the components in time
figure
plot(t, P(:,1), t, P(:,2));
xlabel('Time (years)')
ylabel('Population')
```

lotka volterra logistic function handle

```
f4 = @lotkavolterraLogistic;

% pass lotka volterra logisic equation to ode45 with two sets of parameters
[t1, P1] = ode45(f4, [0,30], [5,1]);
[t2, P2] = ode45(f4, [0,30], [1,5]);

% plot the phase portrait of the lotka volterra logistic system with
% horizontal and vertical nullclines on a slope field
figure
flow
hold on
xline(0, 'black')
yline(0,'red')
yline((1.5/1.1), 'black');
legend('Slope Field', 'h-nullcline', 'v-nullcline')
% anonymous function for one of the clines
f5 = @(x2) ((2.5-(2.5*0.5*x2))/1.4);
hnull = arrayfun(f5, linspace(-1, 6, 100));
plot(P1(:,1), P1(:,2), P2(:,1), P2(:,2), linspace(-1, 6, 100), hnull, 'red');


% plot components in time
figure
plot(t1, P1(:,1), t1, P1(:,2));
xlabel('Time (years)')
ylabel('Population')
legend("Population 1", "Population 2")

% plot components in time for second set of initial conditions
figure
plot(t2, P2(:,1), t2, P2(:,2))
xlabel('Time (years)')
ylabel('Population')
legend("Population 1", "Population 2")
```

Published with MATLAB® R2021b

```
function dydt = eulersMethod(f, h, a, b, y0)
% implementation of eulers method
% defined as f(x,t)_n = f(x,t)_n-1 + h*f(x,t)_n-1
% input:
% f: a differential equation fucntion handle
% h: step size
% a: left time point
% b: right time point
% y0: initial condition
n_steps = (b-a)/h;
y = zeros(n_steps+1, 1);
x = (a:h:b);
y(1) = y0;
    for i=1:n_steps
        y(i+1)=y(i)+h*f(y(i));
    end
dydt = [x; y'];

end
```

Published with MATLAB® R2021b

```
function dpdt = logisticPop(x)
% this function describes a logistic model of population growth that will
% be passed iteratively to a eulersMethod.m
% input:
% x : population size at some time point t
r = 0.65;
L = 5.4;
dpdt = r*(1-(x/L))*x;
end
```

Published with MATLAB® R2021b

```
function population = ivp(t)
% exact solution to IVP
% input
% t: time point
     population = (-54)*exp(0.65*t)/(1-10*exp(0.65*t));
end
```

Published with MATLAB® R2021b

```
function dpdt = logisticPop2(x)
% this function describes a logistic model of population growth that will
% be passed iteratively to a eulersMethod.m
% input:
% x : population size at some time point t
r = 0.65;
L = 8.1;
dpdt = r*(1-(x/L))*x;
end
```

Published with MATLAB® R2021b

```
function harvested = harvesting(x)
% harvesting function
% input
% x: population size at time point t
p=1.2;
q=1;
harvested = (p*(x^2))/(q+(x^2));
end
```

Published with MATLAB® R2021b

```
function dpdt = logisticHarvesting(p)
% combination of logistic equation2 and the harvesting function
% see logisticPop2.m and ahrvesting.m for further explanation
% p: population
    dpdt = logisticPop2(p) - harvesting(p);
end
```

Published with MATLAB® R2021b

```
function error = absoluteError(observed, actual)
% absolute error function
% input:
% observed: observed value
% actual: actual solution
    error = abs(actual - observed);
end
```

Published with MATLAB® R2021b

```
function dxdt= lotkaVolterra(~, p)
% this is an implementation of a lotka-volterra model of
    % differential equations
    %input:
    % ~: dummy variable for time that is needed to pass to ODE45
    % P: population row vector of size 2X1
    a=1.5;
    b=1.1;
    y=2.5;
    d=1.4;
    dxdt = zeros(2,1);
    dxdt(1) = (-a*p(1))+(b*p(1)*p(2));
    dxdt(2) = (y*p(2))-(d*p(1)*p(2));
end
```

Published with MATLAB® R2021b

```
function dxdt = lotkavolterraLogistic(~, P)
    % this is an implementation of a lotka-volterra logistic model of
    % differential equations
    %input:
    % ~: dummy variable for time that is needed to pass to ODE45
    % P: population row vector of size 2X1
    a = 1.5;
    b = 1.1;
    y = 2.5;
    d = 1.4;
    k = 0.5;
    dxdt = zeros(2,1);
    dxdt(1) = (-a*P(1))+(b*P(1)*P(2));
    dxdt(2) = (y*(1-k*P(2))*P(2))-(d*P(1)*P(2));
end
```

Published with MATLAB® R2021b

```
function dirfield(f,tval,yval,plot_title)
% dirfield(f, t1:dt:t2, y1:dy:y2)
%
%   plot direction field for first order ODE y' = f(t,y)
%   using t-values from t1 to t2 with spacing of dt
%   using y-values from y1 to t2 with spacing of dy
%
%   f is an @ function, or an inline function,
%     or the name of an m-file with quotes.
%
% Example: y' = -y^2 + t
%   Show direction field for t in [-1,3], y in [-2,2], use
%   spacing of .2 for both t and y:
%
%   f = @(t,y) -y^2+t
%   dirfield(f, -1:.2:3, -2:.2:2)

[tm,ym]=meshgrid(tval,yval);
dt = tval(2) - tval(1);
dy = yval(2) - yval(1);
fv = f;
if isa(f,'function_handle')
    fv = arrayfun(fv, ym);
end
yp=fv;
s = 1./max(1/dt,abs(yp)./dy)*0.35;
h = ishold;
quiver(tval,yval,s,s.*yp,0,'.r'); hold on;
quiver(tval,yval,-s,-s.*yp,0,'.r');
if h
  hold on
else
  hold off
end
axis([tval(1)-dt/2,tval(end)+dt/2,yval(1)-dy/2,yval(end)+dy/2])

title(plot_title);
xlabel('t values');
ylabel('y values');
```

Published with MATLAB® R2021b

```
% close all; clear all;
% This Matlab code generates a vector field for the system of ODEs
% dx1/dt = f(x1,x2), dx2/dt = g(x1,x2)

% This code currently will find the vector field for the EXAMPLE problem
%            dx1/dt = a*x2
%            dx2/dt = -x1
%---------------------------------------------------------------------------
%         THESE ARE NOT THE PROBLEMS YOU ARE SOLVING FOR PROJECT 1!
% (To have this code generate the vector fields for the Project 1 systems
% of equations, make any necessary adjustments in the sections of code
% labeled with "Step i" where i = 1, 2, 3, 4, or 5)
%---------------------------------------------------------------------------


% Step 1: Set the axis limits so that you plot the vector field over the
%          intervals x1min < x1 < x1max, x2min < x2 < x2max
    x1min = -1; x1max = 6; x2min = -1; x2max = 6;

% Step 2: pick step sizes for x1 and x2;
    x1step = 0.25; x2step = 0.25;

% generate mesh for plotting
    [x1, x2] = meshgrid(x1min:x1step:x1max, x2min:x2step:x2max);

% Step 3: define all needed parameter values
    a = 1.5;
    b = 1.1;
    y = 2.5;
    d = 1.4;
    k = 0.5;

% Step 4: define the system of equations you are using
%     dx1 = (-a.*x1)+(b.*x1.*x2);
%     dx2 = (y.*x2)-(d.*x1.*x2);
%
    dx1 = (-a.*x1)+(b.*x1.*x2);
    dx2 = y.*(1-k.*x2).*x2-(d.*x1.*x2);

% normalize vectors (to help plotting)
    dx2 = dx2./sqrt(dx1.^2 + dx2.^2);
    dx1 = dx1./sqrt(dx1.^2 + dx2.^2);


% generate the vector field
    quiver(x1, x2, dx1, dx2, 'blue', 'AutoScaleFactor',0.5)

% specify the plotting axes
    axis([x1min x1max x2min x2max])

% Step 5: label the axes, include a title
    xlabel('$x1$','Interpreter','latex')
```

```
ylabel('$x2$','Interpreter','latex')
title('Vector field example','Interpreter','latex')
```

Published with MATLAB® R2021b