Drexel University
Office of the Dean of the College of Engineering
ENGR 232 – Dynamic Engineering Systems

Lab 2: The Logistic Equation

The growth of a certain population $P(t)$ can be modeled using an autonomous, non-linear, first-order differential equation:

$$\frac{dP}{dt} = rP \cdot \left(1 - \frac{P}{K}\right)$$

This is called the **logistic equation**. Above, the constant r defines the rate of growth, while K is the carrying capacity.

Part A: Functions in MATLAB

Before studying this differential equation, let's review the **logistic function**, (or logistic curve), which is the common *S*-shaped curve or **sigmoid curve** defined by:

$$f(t) = \frac{L}{1 + e^{-k(t-t_0)}}$$

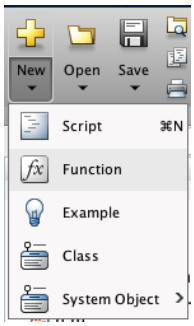
This function was named by Pierre François Verhulst, in his studies of population growth.

Above, L is the limiting maximum value of the sigmoidal curve, t_0 marks the midpoint and k is the steepness.
We'll show **three** different ways to enter this function into MATLAB. These skills can be used any time you need to define a function.

Method 1: In MATLAB, create a new **function** file.
The easiest way to do this is to click the **New** button at the top far left in MATLAB. Select the **Function** option.
This gives the following template for a Function script.

```
function [ output_args ] = untitled7( input_args )
%UNTITLED1 Summary of this function goes here
%   Detailed explanation goes here

end
```



a. Name the input arguments t, t0, L, k (in this exact order!)

b. Name the output argument f.

c. Name the function as **logistic**.

d. Write in the summary. This should be something like:

```
% The logistic function.
% L is the limiting maximum value of the sigmoidal curve,
% t0 marks the midpoint and k is the steepness.
```

e. Define the function by inserting the following line of code just before the **end** keyword. Then save the file as **logistic.m**

```
f = L ./ (1 + exp(-k*(t-t0)));
```

Notice the dot to assure pointwise operations.

Required to prevent data dumps into your command window.

The **standard** logistic function is defined by: $f(t) = \frac{1}{1+e^{-t}}$

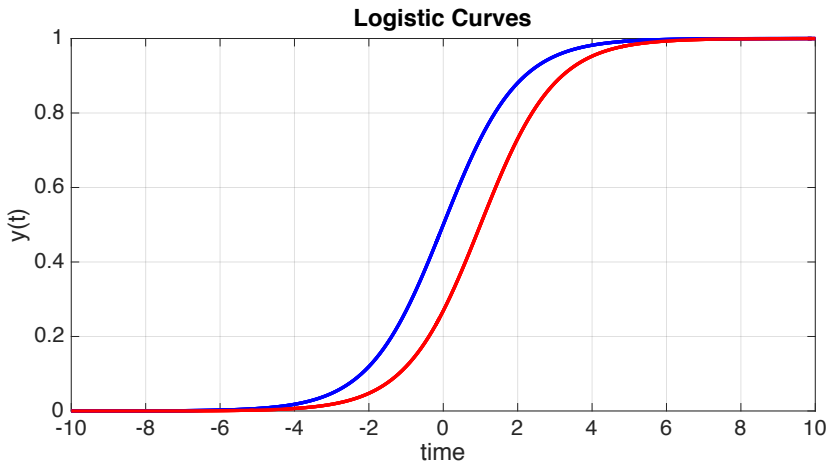
Thus the standard values are $t_0 = 0$, $L = 1$, and $k = 1$. Verify your logistic function is working by confirming the next line returns the midpoint value of 0.5

>> logistic(0, 0, 1, 1)

Here's some code to plot two logistic curves with $t_0 = 0$ and $t_0 = 1$, and the resulting graph. This should help you quickly review some standard plotting options. It assumes you have saved the function file logistic.m in your current working directory.

```
time_pts = -10 : 0.1 : 10;
y_pts = logistic(time_pts, 0, 1, 1);
plot(time_pts, y_pts, 'blue', 'LineWidth', 3) % The standard curve.
grid on
set(gca, 'FontSize', 20)
title('Logistic Curves')
xlabel('time');  ylabel('y(t)')
hold on
y_pts = logistic(time_pts, 1, 1, 1);  % New sigmoid curve with midpoint at t = 1.
plot(time_pts, y_pts, 'red', 'LineWidth', 3)
```

Resulting plot.



Questions 1-2: Adjust the code above to plot a total of 21 sigmoidal curves by varying the midpoint t_0 from -10 to +10 in steps of 1, using a **for** loop. Have the colors gradually transition from blue on the left to red on the right by specifying the color option using RGB triplets. You can see more about color options here: http://www.mathworks.com/help/matlab/ref/colorspec.html

The triplets must gradual change from **blue** = [0 0 1] to **red** = [1 0 0] as t0 varies from -10 to +10

So the colors in the middle will be shades of purple. Hint: Once you update my_color inside your **for** loop, you can use it to plot, by giving at as the option following the **'Color'** keyword.

```
plot(time_pts, y_pts, 'Color', my_color, 'LineWidth', 3)
```

Tip: Your color triplet [r, 0, b] will be zero in the middle and the values for r and b will be **lines** depending on t0, but always satisfying r+b=1.

Questions 1-2: Paste your graph with 21 sigmoidal curves here for 2 points credit. Even better, paste it into the Answer Template.

They should gradually transition from blue to red through shades of purple.

Challenge 1: No points, but important to master before the MATLAB Final. Don't skip!!

Let's create the same plot, but instead of using the file logistic.m, we'll invoke MATLAB's anonymous function syntax.

We'll just name our function f this time

```
f = @(t, t0, L, k)   L ./ (1 + exp(-k*(t-t0)));
```

Method 2 to define a function.
Declare an anonymous function.

We saw this before, but you can review anonymous functions here:

https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html

Recreate the same plot, but use your anonymous function f instead of logistic.m.

Method 3 to define a function.
Create a symbolic expression then use **matlabFunction**.

Challenge 2: No points, but important to master before the MATLAB Final. Don't skip!!

Let's create the same plot, but instead of using the file logistic.m, or the anonymous function f, we'll use matlabFunction.

Note: G = matlabFunction(F) generates a MATLAB anonymous function from the symbolic object F.

Learn more about **matlabFunction** here: https://www.mathworks.com/help/symbolic/matlabfunction.html

Generating a multi-argument function from a symbolic expression using **matlabFunction**.

```
syms t t0 L k
h(t,t0,L,k) =    L / (1 + exp(-k*(t-t0)));
g = matlabFunction(h)

g(0,0,1,1)  % Returns 0.5 just like before.
g(t,0,1,1)  % Returns the standard logistic function.
```

Note: We did **not** need the dot above to indicate pointwise operations, as **matlabFunction** will take care of that automatically for us.

Now create the same plot, but instead of using the file logistic.m, or the anonymous function f,
use the above function **g** which was defined via **matlabFunction**. Success? You are now a master at defining functions in MATLAB!

Part B: Finding solutions of the Logistic Differential Equation

After the above review of MATLAB **functions**, let's return to the logistic differential equation that describes the growth of a population:

$$\frac{dP}{dt} = rP \cdot \left(1 - \frac{P}{K}\right)$$

Above, the constant r defines the rate of growth, while K is the carrying capacity.
For this lab, we will assume the growth rate is $r = 2$ and the carrying capacity is $K = 10$ and the initial population is $P(0) = 1$.
The units (depending on the application) might be in thousands, millions or even billions.

$$\frac{dP}{dt} = 2P \cdot \left(1 - \frac{P}{10}\right), \qquad P(0) = 1$$

Find the **exact** solution using **dsolve**, and plot over the range from $t = 0$ to 10.
Use a **blue** curve with a 'LineWidth' of 3. Here is some starter code.

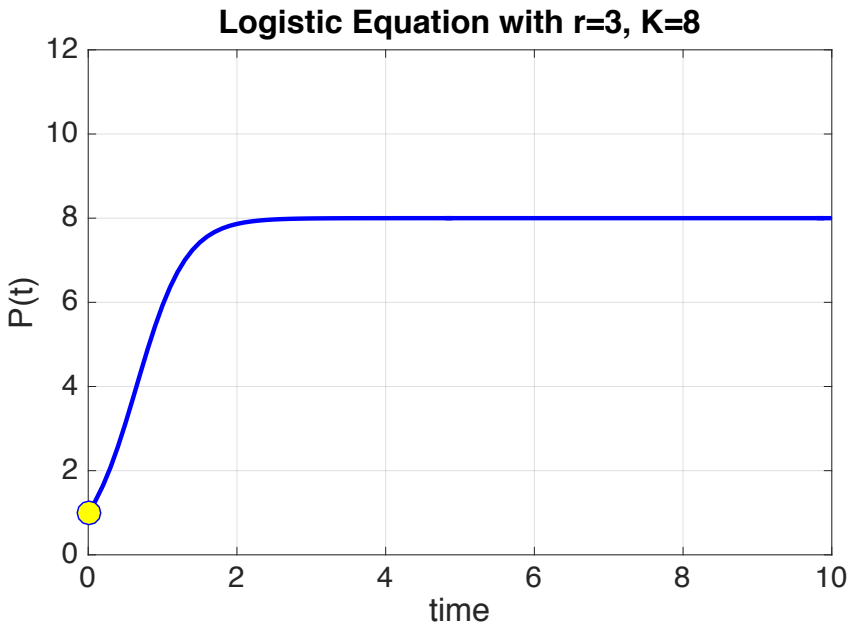
```
clear, clc
syms P(t)
r=2; K = 10;
DE = diff(P,t) == r*P * (1 - P/K)
sol = simplify( dsolve(DE, P(0)==1) )
p = matlabFunction(sol)  % Use little p for the solution.
```

Question 3: Simply enter p(t) in the command window to obtain the exact solution for the DE satisfying P(0) = 1.

Question 3: The exact solution satisfying P(0) = 1 is $p(t)$ = _____

Question 4: Plot the exact solution (found with **dsolve**) over the interval from $t = 0$ to $t = 10$. Use a **blue** curve with a thickness of 3, turn the **grid on**, set the **title** to 'Logistic Equation with r=2, K=10', set the **xlabel** to 'time' and the **ylabel** to 'P(t)'. Turn **hold on**, so all the later graphs will appear in the same figure. Place a **yellow** dot to indicate the initial point.

Replace the graph below, with your own graph which will be different.
This is the solution using different values for *r* and *K*. Your graph will be less steep, and will rise up to 10.



Next, we'll compare the exact solution, to several numerical approximations. First to an **Euler approximation** with a fairly large step size, and then to a better Euler approximation with smaller steps. Finally, we'll get a very good approximation using **ode45** and combine them all on the same graph.

Question 5: Slope Function for the DE

To help us explore this differential equation numerically, we will need to define its slope as a function.

$$\frac{dP}{dt} = f(t,P) = rP \cdot \left(1 - \frac{P}{K}\right), \qquad P(0) = 1 \quad \text{and } r = 2, K = 10$$

We saw there are at least three different methods to do that, but we will just use an **anonymous function** for $f(t,P)$.
Enter this now in a new section of your MATLAB script for Lab 2.

```
r = 2; K = 10;
f = @(t,P)  r*P * (1 - P/K)
```

Note we declared *t* as an input even though it *f* does not depend on *t*. That will be useful later when we invoke **ode45**.

Let's verify your function is working. Find the initial value of the **slope**. This is just f(0,1).

Q5. The initial value of the slope is: _ _ _ _ _ _

The last five points are obtained by completing the remaining steps and submitting your completed graph. (5 points)

Recall you already have a figure with the exact solution, and you should have turned hold on, so all these additional graphs are automatically added to the same figure.

Euler's Method with Two Different Step Sizes

Q6. Add a plot of the approximate solution using Euler's method with a step size of $dt = 1$ (This is a very coarse step size. The purpose is so we can actually see the error.) Plot each point as a **red** asterisk and connect the points with a dotted line. Hint: You will use the function named f above, in a for loop to find the Euler points. Review the Pre-Lab for code ideas.

Here's just a bit of code to get you started.

```
% Initialize the variables. Here we are using y for the population.
dt = 1;                         % Step size. Also called h in the notes.
tStart = 0;      yStart = 1;    % Population starts at P(0) = 1.
tEnd = 10;                      % Stopping time.

% Define time points and solution vector
t_points = tStart: dt: tEnd;
y_points = zeros(size(t_points));  % Use zeros as place holders for now.

% Initialize the solution at the initial condition.
y_points(1) = yStart;
```

Now implement Euler's method using a **for** loop. Plot the points in the same figure as your exact solution.
You already have the horizontal coordinates for each point inside the row vector named **t_points**.
To find the vertical coordinate `y_points(i)` for the i^{th} point, first find the slope at the previous point, then use Euler's formula.

Q7. Clone your Euler code, and repeat the calculations but now use a finer step-size of $dt = 1/4$. The fit should be much better.

Add another plot of the approximate solutions using this finer version for Euler's method. Plot each point as a **black** asterisk, and connect the points with a dotted line.

See sample plot at the end (using different values for *r* and *K*), to see roughly how your graph should look at this stage.

Q8. Now find the approximation using **ode45**.

Here's the main command you need, provided you have defined all the variables given as its arguments.

```
[t_out, y_out] = ode45(f, [tStart tEnd], yStart);
```

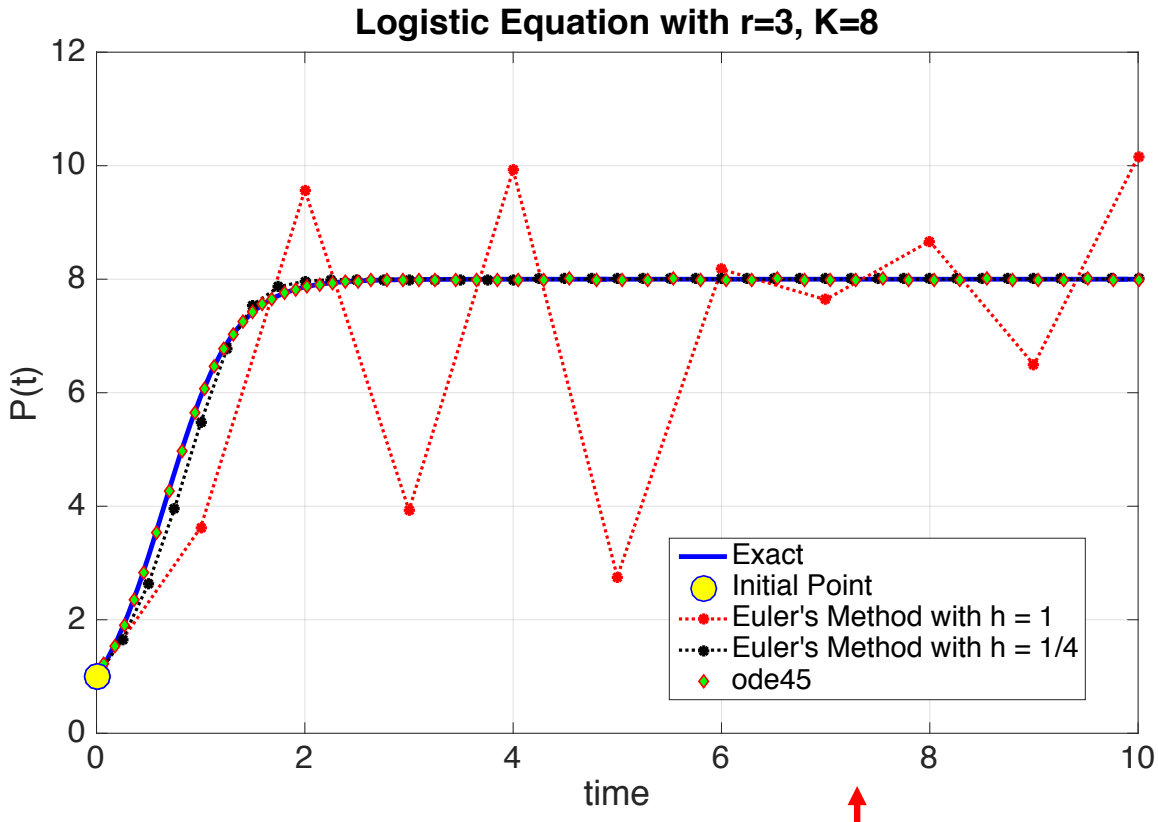
Show the solution points as **green** diamonds with **red** outlines. **Hint:** d is for diamonds.
Set the face color to green using the `'MarkerFaceColor'` keyword. The outline is specified as usual.

Q9. Add a **legend** to your figure, similar to that shown in the sample plot below.

Q10. Locate your legend to avoid the data points as much as possible. Adding these two arguments at the end of your **legend** command, will position the legend to avoid as many data points as possible.

`'Location', 'Best'` ← Place as the last two arguments inside the legend command.

Replace the graph below, with your own graph which will be different. (5 points)
This is the solution using different values for *r* and *K*. Your graphs will be less steep, and will rise up to 10.



The legend has avoided obstructing the data points thanks to setting the `'Location'` to `'Best'`.