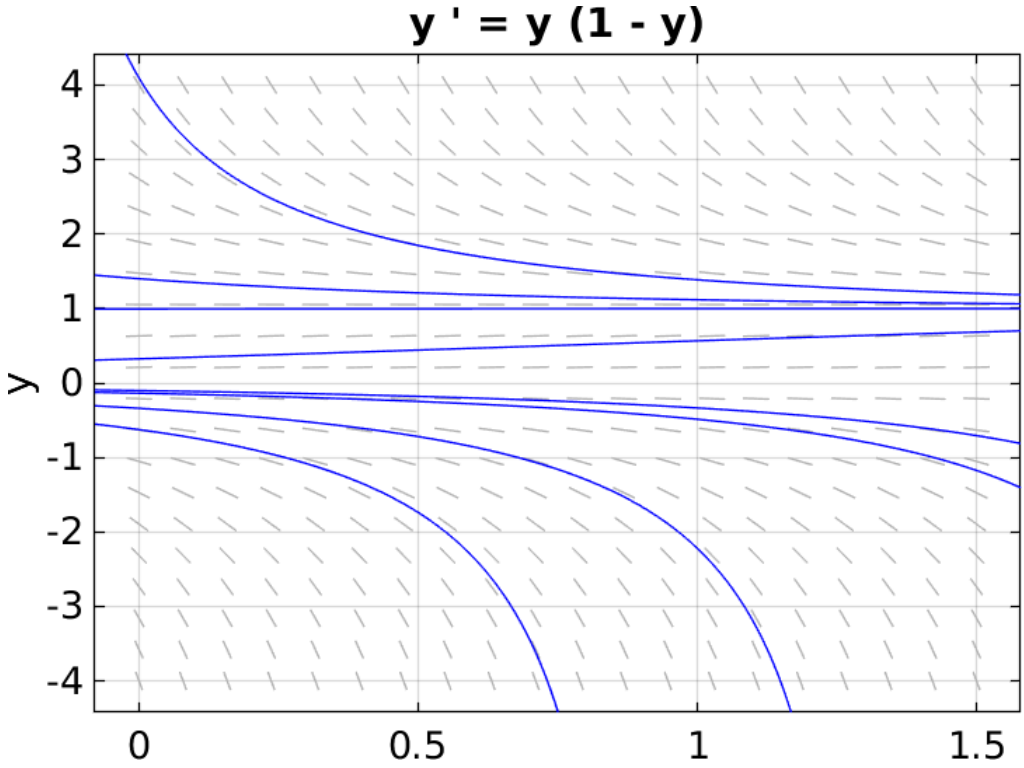
Euler Logistic:
A new problem: the logistic equation

MATH1090: Directed Study on Numerical ODE's

http://people.sc.fsu.edu/~jburkardt/classes/math1090_2020/euler_logistic/euler_logistic.pdf

---



Some sample solutions of the logistic equation for $0 \leq t \leq 1.5$.

Euler Logistic

Solutions of the logistic equation can have sharp turns that are hard for the Euler code to follow unless small steps are taken.

1 Introduction

We know that the results of our computational approach to a differential equation are only estimates for the correct solution. In a textbook problem, we know the exact solution and can compare it to our results. But in a real life problem, is there any way to estimate the accuracy of our results? Is there a way to try to improve the accuracy if we think we have done badly?

2 The logistic equation

The logistic equation is an example inspired by biology. The differential equation is:

$$\frac{dy}{dt} = y * (1 - y)$$

To complete the problem, we need an initial condition of the form

$$y(t_0) = y_0$$

For biologists, this problem is meaningful for $0 < y(t) < 1$. The value of y represents the current size of a population as a fraction of the maximum possible population. For y in the acceptable range, the population tends to increase towards 1. If y exceeds the value 1, then the differential equation rapidly drives the population back to 1. Negative values of y don't have an obvious biological meaning. Nonetheless, this is still a mathematically meaningful equation, and if we test out a negative starting value, we will be able to look at some interesting results.

Our concern is to understand how our approximate solution differs from the true solution to the differential equation, and how the quality of our approximations depends on the stepsize we use. To make this investigation, we will want to know in advance what the exact solution is, knowledge that is unavailable in a real world problem.

3 Direction fields with dfield9

The MATLAB program dfield9 allows you to plot the direction field of an ODE by specifying just the right hand side function. A copy of this program is available in the file `dfield9.m` at `http://people.sc.fsu.edu/~jburkardt/classes/math1090_2020/scripts/scripts.html`.

If you start the program, you will see a graphic interface including a plot and an area where you can define the problem and the viewing region. Specify the dependent variable as y, the right hand side as $y * (1 - y)$, and specify the minimum t as 0 and the maximum t as 1.5. For now, leave the maximum and minimum values of y at their default values of 4 and -4.

Now click anywhere in the plot. The program will draw the solution curve that passes through that point. See if you can put your cursor near to the point (0.0,-0.2), which is the initial condition we will study, and click to see what the solution does.

You can see that many initial conditions create curves that move towards the value $y = 1$, squeezing together, but conditions with a negative value of y all seem to plunge downward, and spread apart. This spreading apart means that, for our initial condition, our errors will tend to grow with increasing t.

We will use `dfield9` from time to time when we work with other differential equations, in order to get a feeling for how the family of solution curves behaves.

4 An exact solution

If we take our initial condition as $y(0) = -\frac{1}{5}$, then it turns out that the exact solution is $y(t) = \frac{1}{1-6e^{-t}}$. We can see this as follows:

$$\begin{aligned}
\frac{dy}{dt} &= y * (1-y) && \text{Differential equation} \\
\frac{dy}{y(1-y)} &= dt && \text{Separation of variables } y \text{ and } t \\
\int \frac{dy}{y(1-y)} &= \int dt && \text{Indefinite integrals} \\
\int (\frac{1}{y} + \frac{1}{1-y}) dy &= \int dt && \text{Partial fractions} \\
\ln(y) - \ln(1-y) &= t + c_1 && \text{Antiderivatives, } c_1 \text{ arbitrary constant} \\
\frac{y}{1-y} &= c_2 e^t && \text{Exponentiate, } c_2 = e^{c_1} \\
y &= \frac{1}{1+c_3 e^{-t}} && \text{Algebra solving for } y \text{ for any initial condition} \\
y(0) = -\frac{1}{5} &\rightarrow c_3 = -6 && \text{Apply our initial condition}
\end{aligned}$$

With this exact solution, we will be able to check the errors we make in our ODE approximations with the Euler method.

5 Write derivative and solution functions

Write a MATLAB function named *logistic_deriv.m*, with the following format, filling in the details:

```
function dydt = logistic_deriv ( t, y )
  dydt = (the derivative formula above)
  return
end
```

Listing 1: Logistic derivative code.

and a MATLAB function named *logistic_solution.m*:

```
function y = logistic_solution ( t )
  y = (the solution formula above)
  return
end
```

Listing 2: Logistic solution code.

When you write the formula for `y`, be sure to write the fraction using the "dotted" division sign, that is, something like `1 ./ (bottom)`. Otherwise, you will have trouble later when we try to evaluate the solution with a vector of t values.

6 Use your (forward) Euler ODE solver

Now use your code *euler.m* to estimate the solution of this ODE, with our specified initial condition, over the range $0 \leq t \leq 1.5$. Compute the error as the RMS norm of the difference between the computed and true solutions:

```
  dydt = @logistic_deriv;
  tspan = [ 0.0, 1.5 ];
  y0 = -0.2;
  n = 10;
  [ t, y1 ] = euler ( dydt, tspan, y0, n );
  y2 = logistic_solution ( t );
  e = rms ( y1 - y2 );
  fprintf ( 1, '  For n = %d, RMS error is %g\n', n, e );
```

Listing 3: logistic_euler.m computes the logistic Euler estimate and exact solution.

7 Plot your solution

Now let's compare our computed and exact solutions together on a plot:

```
plot ( t, y1, 'r', t, y2, 'b', 'linewidth', 3 );
grid ( 'on' );
xlabel ( '<-- t -->' );
ylabel ( '<-- y(t) -->' );
title ( 'Logistic equation y'' = y * ( 1 - y )' );
legend ( 'Euler solution (red)', 'Exact solution (blue)' );
print ( '-dpng', 'logistic_euler.png' );
```

Listing 4: Plot computed and exact solutions.

8 Errors depend on stepsize

You probably know or can guess that the accuracy of our ODE approximation should tend to improve if we use more steps, that is, increase the value of n; correspondingly, we are decreasing the stepsize $h = \frac{Tfinal - Tstart}{n}$.

In order to study this process, we are going to make a new program that generates a sequence of increasing values of n and requests an Euler solution and the corresponding error e:

```
fprintf ( 1, '\n' );
fprintf ( 1, 'logistic_errors:\n' );
fprintf ( 1, '\n' );
n = 10;
for i = 1 : 8
  e = logistic_euler2 ( n );
  fprintf ( 1, '  %d  %g\n', n, e );
  n = 2 * n;
end
```

Listing 5: logistic_errors.m tabulates errors for increasing n.

where the function `logistic_euler2()` is a revised version of your `logistic_euler()` code, which you have rewritten as a function, so that the value of n is input, and the value of e is output:

```
function e = logistic_euler2 ( n )
%
%  Most of the stuff in logistic_euler, but
%  remove print statements
%  remove "n=10"
%
  return
end
```

Listing 6: logistic_euler2.m gets n and returns e.

The result of running `logistic_euler2()` should be a table of the RMS errors associated with the increasing n (and decreasing h).

Using this table, we can make an intelligent guess about how the accuracy of the Euler code improves when we use twice as many steps.

9 Introduction to LaTeX

We would like you to try to learn some of the LaTeX mathematical typesetting system, so that you can write neat reports that include graphs and tables and equations and references. While you could install the LaTeXprogram on your own computer, it might be easier to work with the online Overleaf program at `https://www.overleaf.com/`.

If you set up an account there, they have a *Learn LaTeX in 30 minutes* guide that might get you started. For this week, we only ask that you try to get access to LaTeX at Overleaf or on your own system, and that you try to write a short document using LaTeX.

An unformatted version of your document might look something like this:

```
Approximate solutions of a logistic equation using the Euler method
Yourname here!

(section) Introduction:

  This article describes an ordinary differential equation known as the
  logistic equation, and my attempts to estimate a solution using the
  forward Euler method.

(section) The forward Euler method:

  To solve a differential equation of the form
    dydt = f(t,y)
    y(t0) = y0
  using n steps from t0 to a later time Tfinal, the forward Euler method
  carries out the following iteration:

(section) The logistic equation:

  Our example differential equation is the logistic equation, which has the form:
    equation...
    initial condition...

  The dfield9 program can be used to display direction fields of this equation.

(section) The exact solution of the logistic equation

(section) Comparing exact solution and Euler results

  By solving the same equation several times, with increasing values of n, and comparing
  with the exact solution, we studied how the accuracy of the Euler method depends on
  the stepsize.
```

```
The following table summarizes our results:

N  RMS error
--  -------
10  ?????
20  ?????
```

10 Report

Bring your MATLAB codes and plots, and your first LaTeX file, to our next meeting, on 2:00pm, Thursday, 30 January, in room Thackeray 624.