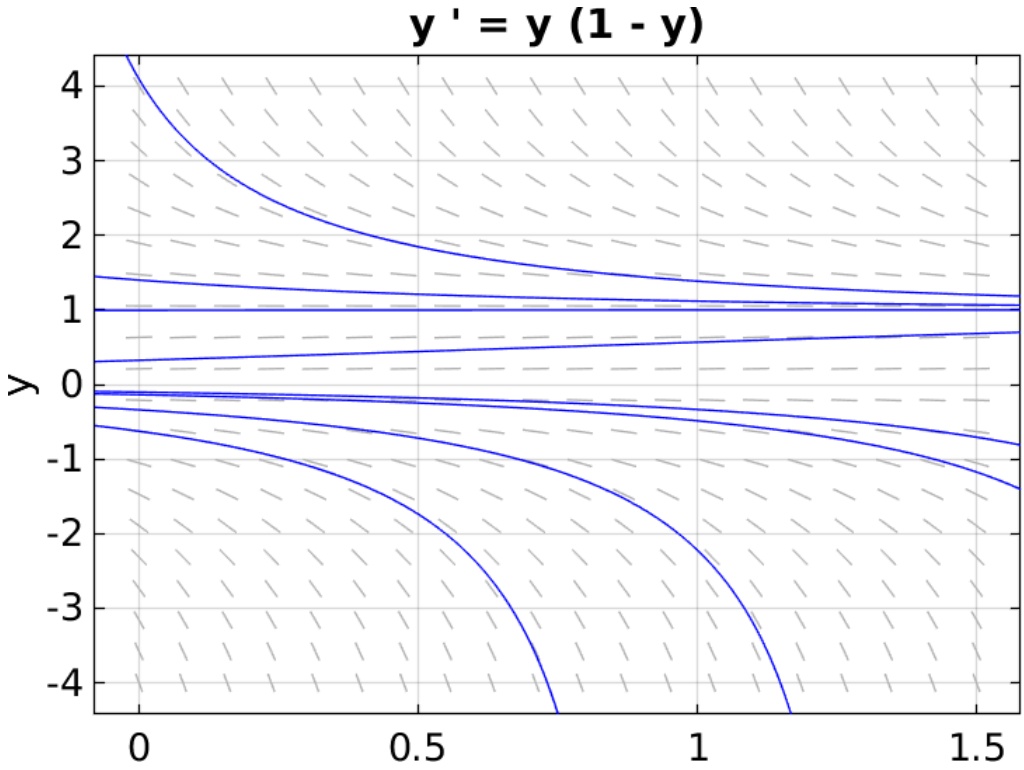
Logistic:
Solving the logistic equation

MATH1902: Numerical Solution of Differential Equations

http://people.sc.fsu.edu/~jburkardt/classes/math1902_2020/logistic/logistic.pdf

---



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

The Logistic Equation

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 $0 \leq y(t)$. They consider an environment that can support up to M creatures, but no more. To standardize the discussion, we "normalize" the problem, so that the value of y represents the size of the current population as a proportion of M. Thus in this example, $y = 0.2$ and $y = 1.2$ would correspond to populations of 100 and 600 respectively.

Using the normalized quantity y, we can note that, if at the initial time t_0, the value of $y(t_0)$ (which we will also call y_0) is strictly positive, then if population growth follows the logistic equaiton, it will tend towards 1 as time increases. Of course, an initial value of 0 stays zero. While negative initial values of y don't have an obvious biological meaning, the mathematical equation allows them, and they have their own kind of behavior.

Knowing only the differential equation, and the initial value, we can use the Euler method to draw a partial. approximate picture of the solution over time. It's a *partial* picture, because we limit our study to some range of time $t_0 \leq t \leq tstop$, and it's *approximate* because the Euler method is only able to produce an estimate of the solution behavior at selected points in that time interval.

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 number of steps n, or equivalently the stepsize $h = \frac{tstop - t_0}{n}$ that we use.

In a real world problem, we will never know the actual error we are making. For our example, however, we will have an exact formula to compare, and thus we can see whether our approximation process is doing a good job.

3 *dfield9.m* can display ODE direction fields

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/math1902_2020/logistic/logistic.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. Using this interface:

- Specify the dependent variable as y;
- Enter the right hand side as $y * (1 - y)$;
- Specify the minimum t as 0 and the maximum t as 3;

Now click anywhere in the plot. The program will draw the solution curve that passes through that point. We will assume our starting time is $t_0 = 0$ so from now on, we will try to click on initial values along the y axis. See if you can put your cursor near to the point $(t, y) = (0.0, -0.2)$, which is the first initial condition we will study, and click to see what the solution does. This is an example of a negative initial condition, and you can see that the solution disappears on its way to $-\infty$. Now try the initial condition $(t, y) = (0.0, 0.2)$. You should see that the solution is much better behaved.

Try a sample of positive initial conditions, such as $(t, y) = (0.0, 0.4)$, $(t, y) = (0.0, 0.8)$, $(t, y) = (0.0, 1.2)$. 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 behavior suggests that, for positive starting values, all the solutions will tend towards 1; this squeezing process means that even if we make small errors in the initial condition value, or in our approximation process, we will nonetheless tend towards the correct value in the long run.

If, for some reason, we were interested in a problem with a negative initial condition, then the spreading behavior of the solution curves means that small errors in the initial condition, or in our Euler method, will mean that our approximation will pull away from the exact solution at a faster and faster rate.

The `dfield9` program may help you to visualize the differential equation problem. The right hand side defines a family of solutions, something like a series of railroad tracks. Our initial condition picks one railroad track, which is the exact solution. We would like to start at the initial condition and follow that exact solution. However, we only have approximate methods, and so at each step, we are likely to shift to a nearby railroad track. Our approximation process then becomes a sequence of errors, so that we are liable to drift away from the exact solution as we proceed.

4 *logistic_deriv.m* evaluates *dydt*, the logistic ODE right hand side

We will want to try out our Euler method with this logistic ODE. To do so, we will need a derivative function. Write a MATLAB function named *logistic_deriv.m*, with the following format, filling in the details:

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

Listing 1: Pseudocode for the logistic ODE derivative.

5 *logistic_solution.m* evaluates the exact solution

It is possible to determine the exact solution of the logistic ODE once we know the initial condition information. For instance, if we take our initial condition for the logistic equation as $y(0) = \frac{1}{5}$, then it turns out that the exact solution is $y(t) = \frac{1}{1+4e^{-t}}$.

This is a rare chance to do some mathematics, so let's work through the details:

$$\frac{dy}{dt} = y * (1 - y) \quad \text{Differential equation}$$

$$\frac{dy}{y(1-y)} = dt \quad \text{Separation of variables } y \text{ and } t$$

$$\int \frac{dy}{y(1-y)} = \int dt \quad \text{Indefinite integrals}$$

$$\int (\frac{1}{y} + \frac{1}{1-y}) dy = \int dt \quad \text{Partial fractions}$$

$$\ln(y) - \ln(1-y) = t + c_1 \quad \text{Antiderivatives, } c_1 \text{ arbitrary constant}$$

$$\frac{y}{1-y} = c_2 e^t \quad \text{Exponentiate, } c_2 = e^{c_1}$$

$$y = \frac{1}{1 + c_2 e^{-t}} \quad \text{Algebra solving for } y \text{ for any initial condition}$$

$$y(0) = \frac{1}{5} \rightarrow c_2 = +4 \quad \text{Apply our initial condition at time } t = 0$$

This solution applies if we have chosen the initial condition $t_0 = 0, y_0 = \frac{1}{5}$. You should see that the corresponding exact solution function $y(t)$ is positive for all values of time t (even backwards in time).

How would our exact solution change if we had chosen instead the initial condition $t_0 = 0, y_0 = -\frac{1}{5}$? Can you see that, for this choice of initial condition, the solution, starting at time $t = 0$, will always be negative, and decrease to infinity?

What about an initial condition $t_0 = 0, y_0 = 10$?

Since we may be interested in various initial conditions, we can write a formula for the exact solution that allows us to choose any values for t_0, y_0:

$$y(t) = \frac{1}{1 + (\frac{1}{y_0} - 1)\, e^{t_0 - t}}$$

Since we happen to know the exact solution of the logistic ODE, any time we compute an approximate solution, we can compare it with the exact solution and decide how well we have done.

Although it is usual to specify the initial condition at time $t_0 = 0$, let's use the formula above which allows us to specify any initial value y_0 at any initial time t_0. This means our solution function will need to have these two values as extra inputs:

```
1 function y = logistic_solution ( t, t0, y0 )
2   y = (the solution formula above)
3   return
4 end
```

Listing 2: Pseudocode for logistic solution.

Note that in MATLAB, the mathematical expression e^x is written `exp ( x )`.

6 *rms.m* reports the approximation error

How can we compare an approximate solution to the exact solution? The approximate solution is a vector or list of n values, perhaps at regularly spaced times. The exact solution is a formula that can be evaluated

for any time. Suppose our ODE solver returns the approxiate solution as the pair of n pairs of values (t_i, y_i). To make a comparison, we evaluate our exact formula at these same times. Now we want to decide how well the first list approximates the second. We can do this with the RMS norm.

The RMS norm (RMS for *"root-mean-square"*) is a weighted average of a list of values. Given n numbers $x_1, x_2, ..., x_n$ in a vector x, we define the RMS norm of x as

$$e = \sqrt{\frac{x_1^2 + x_2^2 + ... + x_n^2}{n}}$$

Thus, the RMS of the 10 values $1, 2, ..., 10$ is 6.2048.

The RMS norm is very useful when comparing the behavior of vectors of different lengths. We can sketch a MATLAB function that carries out this task as follows:

```
function value = rms ( x )
get n, the length of x
start value at 0
add the square of each entry of x.
divide value by n
take the square root of value
return
end
```

Listing 3: Pseudocode for the RMS function.

We will need an RMS function for our next calculation. Write an RMS function based on the sketch given above. Note that in MATLAB, x^2 is written `x^2`, and $\sqrt{value}$ is `sqrt ( value )`.

Verify that your function computes the correct RMS of the vector $x = [1, 2, ..., 10]$.

7 *logistic_euler.m* solves the equation and returns the RMS error

Now use your code *euler.m* to estimate the solution of the logistic ODE, with initial conditions $t_0 = 0, y_0 = 0.2$, over the range $0 \leq t \leq 10$. Compute the error `e` as the RMS norm of the difference between the computed and true solutions. The function *logistic_euler(n)* manages this process:

```
function e = logistic_euler ( n )

  dydt = @ logistic_deriv;
  t0 = 0.0;
  tstop = 10.0;
  tspan = [ t0, tstop ];
  y0 = 0.2;
  [ t, y1 ] = euler ( dydt, tspan, y0, n );

  y2 = logistic_solution ( t, t0, y0 );

  e = rms ( y1 - y2 );

  return
end
```

Listing 4: logistic_euler.m computes the logistic Euler estimate and exact solution.

Try in succession the values $n = 10, 20, 40$ and print out the resulting RMS value e in each case. What does the error do as we increase n?

8 *logistic_euler_plot.m* compares exact and approximate solutions

We'd like to compare our computed and exact solutions. We can do this by starting with a copy of *logistic_euler.m*, dropping some statements, and adding a plot command. The approximate solution is only evaluated at n points. We want the exact solution to be plotted at a lot of points, so that its curve looks smooth. So for the plot, we evaluate the exact solution at 101 points `t2`, rather than simply using the same time values used by the Euler method.

```
function logistic_euler_plot ( n )

  dydt = @ logistic_deriv;
  t0 = 0.0;
  tstop = 10.0;
  tspan = [ t0, tstop ];
  y0 = 0.2;

  [ t1, y1 ] = euler ( dydt, tspan, y0, n );

  t2 = linspace ( t0, tstop, 101 );
  y2 = logistic_solution ( t, t0, y0 );

  plot ( t1, y1, 'ro', t2, y2, 'b', 'linewidth', 3 );
%
%  The following commands make the plot nicer, and save a copy in a file.
%  You can omit them if you prefer.
%
  grid ( 'on' );
  xlabel ( '<-- t -->' );
  ylabel ( '<-- y(t) -->' );
  title ( 'Logistic equation y'' = y * ( 1 - y )' );
  legend ( 'Euler', 'Exact' );
  print ( '-dpng', 'logistic_euler_plot.png' );

  return
end
```

Listing 5: logistic_euler_plot.m makes a plot.

Now repeat your previous computation, using the command `euler_plot ( 10 )`. The error should be zero initially, and we already suggested that it should be small at the end. Thus, the plot should reveal that the large errors occur somewhere in the middle.

9 *logistic_errors.m* varies the number of steps

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 $h = \frac{tstop-t0}{n}$. If the error decreases as we increase n, we say that our approximate solution seems to be *converging* to the exact solution.

In order to gather evidence of convergence, 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:

```
function logistic_errors ( )

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

```
9
10    return
11  end
```

Listing 6: logistic_errors.m tabulates errors for increasing n.

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

Now we know our approximations are not perfect. Because we have the exact answer for the logistic ODE, we can observe the error that we make with each value of n that we choose. A larger n means more work, but does it also mean better accuracy? Using the table of n versus RMS errors, can you estimate what happens to the error each time we double the value of n?

10 Homework #2

Put your table of n versus RMS error into a text document, something like this:

```
I     N   RMS

1    10  0.0265782
2    20  ?
3    40  ?
4    80  ?
5   160  ?
6   320  ?
7   640  ?
8  1280  ?
```

Send a copy of your table to `trenchea@pitt.edu`