APPM 2460
Numerically Solving ODEs

1 Introduction

Today we'll review numerically solving ordinary differential equations. This is one of the key uses of Matlab, so it's helpful to learn it as early on as possible.

What does it mean to "numerically" solve an ODE? Suppose we have the initial value problem

$$y'(t) = y(t)^2 \cos\big(t + y(t)\big), \quad y(0) = 1. \tag{1}$$

Take a moment to try solving this. Separation of variables won't work, since the $y(t)$ is *inside* the cosine function. Integrating factor won't work either. Equation (1) is in fact nonlinear, and quite difficult to solve. None of the methods we know work, and in some cases, there are in fact no methods that allow us to write down a solution.

So what are we to do? It turns out that even if none of our usual methods for solving the equation work, we can use computers to find an **approximate** solution.

> Numerically solving an ODE gives us an **approximate** solution.
> It is most helpful when we cannot find an **exact** solution.

You will also see the phrase "analytical solution" used in place of "exact solution." **When you solve an ODE using a pen and paper (e.g. by separation of variables, integrating factor, variation of parameters, or any other method) you are finding the analytical solution.**

There are myriad methods that can be used to numerically (read: approximately) solve an ODE. Matlab has many of these methods built in, and the development and analysis of such methods is an active area of research here in the applied math department. We will only discuss the most basic method, known as Euler's method. However, we'll also learn to deploy more advanced methods using Matlab's built-in functionality in future assignments.

2 Euler's Method

Today we will build our own functions so we can solve difficult problems like (1). Typically when we build approximate solvers for ODEs, we should test our code on a problem that we can actually solve–we will implement this process today.

Consider the (general) first order initial value problem

$$y'(t) = f(t, y), \quad y(0) = y0, \tag{2}$$

Recall that Euler's method to approximate solutions to such problems is

$$y_{n+1} = y_n + hf(t_n, y_n), \quad y_0 = y0. \tag{3}$$

We've had some practice doing this on our 2360 homework this semester, and I'm sure that you've found it to be quite tedious to go through this calculation. That's why we want to use Matlab to do all the tedious calculations for us.

3 Euler's Method in Matlab

If we want to implement Euler's method we first rewrite the differential equation as

$$y'(t) = f(t, y). \tag{4}$$

Let's write some *pseudocode* to help us understand how to implement Euler's method. First we write a **function** to define the right hand side of the ode. For differential equations such as (4), an anonymous function will work well:

```
y_prime = @(t,y) function_definition;    % y_prime = f(t,y)
```

Now we can write a **script** that can actually preforms Euler's method.

```
h = ;             % specify the step size in Euler's method
t = t0:h:t_end;   % specify the time interval
y(1) = y0;        % set initial condition

for n = 1:(length(t)-1)
    y_{n+1}  = y_{n} + h*yprime(t_n,y_n)
end
```

Note: Parts of the above pseudocode is not consistent with Matlab syntax. Subscripts can be interpreted as the index within a vector in order to translate mathematical expression to Matlab code.

Note: The script computes a *vector* of time values `t` then (via the Euler solve in the loop) it computes a *vector* of approximate solutions values `y`. We can visualize the approximate solution by plotting `y` against `t`.

4 Example

Now let's implement the above code to solve a specific problem with Euler's method.

Consider the initial value problem

$$y' = yt, \quad y(0) = 1. \tag{5}$$

- In a script, define the function $f(t, y)$ (let's use an anonymous function for now):

```
f = @(t,y) y*t;     % The RHS of the DE
```

- To implement the method, we need a step size h and an interval of t values where we will solve the problem. Pick $\boldsymbol{t \in [0, 4]}$ and $\boldsymbol{h = 0.001}$. As in the pseudocode, define the step size and then create the vector of times `t` (let's also initialize the y-vector):

```
h = 0.001;   % Step size
t = 0:h:4;   % Vector of t values

y(1) = 1;    % The initial value of the y-vector.
             % Notice that this notation means "the first
             % position in the y-vector has the value 1"
```

- Now we will create the for-loop to implement the Euler solve:

```
for n = 1:(length(t)-1)
    y(n+1) = y(n) + h*f(t(n),y(n));   % Euler's method.
                                      % Careful of indices!
end
```

- Save and run the script to make sure that there are no errors. Then, to visualize the numerical solution, plot `y` against `t`.
- Now find the analytic solution $y_{\text{true}}(t)$ using the method of your choice.
- Compare the numerical solution to the analytical solution by plotting both on the same axes. The true solution and the numerical solution should be almost indistinguishable.

Now we can modify our code to approximate the solution of a problem that we cannot solve by hand.

APPM 2460 Homework 5

Submit a published pdf of your script solving the following problem to Canvas by Monday, February 18 at 11:59 p.m. See the 2460 webpage for formatting guidelines.

Consider the initial value problem

$$y' = y^2 \cos(t + y), \quad y(0) = 1.$$

(a) Modify your Euler's method code to solve this IVP with time-steps $h_1 = 0.5$, $h_2 = 0.1$, $h_3 = 0.05$, and $h_4 = 0.01$ over the interval $t = [0, 23]$.

(b) Plot *all* these solutions **on a single plot**. Be sure to label the axes, title the plot, and provide a legend distinguishing the solution curves by step-size.