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

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

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, 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.

Recall that Euler's method to approximate a solution to the initial value problem

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

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.

```
function yp = yprime(t,y)
%% define f to be the right hand side of the ode
yp = f(t,y) %% f(t,y) is the right hand side of the ode
end
```

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

```
%% This script will use Euler's method to compute the solution of a
   differential equation y' = f(t,y) with initial conditions y(t0) = y0.
    The approximate solution is a vector "y" with values corresponding
   to a vector of times "t"

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
```

We can modify the above code outline to perform Euler's method. Let's test our code by solving the initial value problem

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

To do this pick $t \in [0, 2]$ and $h = 0.001$. Compare the approximate solution to the true 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.

4. Homework

Consider the initial value problem

$$y'(t) = \cos(t + y), \quad y(0) = 1. \tag{6}$$

Modify your Euler's method code to solve it with timesteps $h = 0.5, 0.1, 0.05$, and 0.01. Plot all these solutions **on a single plot**. Label your plot, and your axes, and provide a legend indicating which curve corresponds to which timestep.