Drexel University

Office of the Dean of the College of Engineering

ENGR 232 – Dynamic Engineering Systems

Week 2 – Pre Lab

Numerical Methods

For first order system models, if the equation is separable we can find a solution through separation of variables, providing that the integrals in the separated form exist. For linear first order system models, we also have the added benefit of being able to solve using the integrating factor method.

Not all equations will have a solution, and at times, the integrals required to solve systems may be difficult to compute. Furthermore, as you would have seen in lecture this week, it is not always easy (or possible) to express the solution in the form: $y=f(t)$ (implicit solution).

Example:

$$\frac{dy}{dt} = \frac{t^2+1}{y^2-1}$$

This has a solution via separation of variables:

$$y^3 - 3y - (t^3 + 3t + 3c) = 0$$

This is not the most straightforward equation to plot, since you need to solve the equation for $y(t)$. In these situations, a numerical solution will suffice to analyze the system behavior.

In lecture last week we would have studied the method using direction fields and Euler's method. This week we will study more precise numerical methods –the Runge-Kutta 4th order method using MATLAB for the implementation.

We will see an implementation of these methods using the following example:

$$\frac{dy}{dt} = 4y - y^2, \quad y(0) = 1$$

a. Euler's Method – Lecture from Wednesday

Euler's method as we derived in the class notes from last Wednesday requires use to write a function that has to be used. Start by defining a function in MATLAB as follows:

```
function [dydt] = diff_example(t,y)
dydt = 4*y -y^2;
```

Note: The function has to be written with inputs (t,y) in that order. This is needed for ode45 so we will adopt this convention for defining all first order functions.

As shown in the class notes, Euler's method implementation solves the differential equation at a series of time points.

```
%% ENGR 232 Week 1 Euler Example
%% Initialize Variables
dt = 0.1;
yI = 1;
tI = 0;
tEnd = 5;
%% Define time points and solution vector
tSpan = tI:dt:tEnd;
y = zeros(size(tSpan));
%% Initialize the solution at the initial conditions
y(1) = yI;
%% Implement Euler's method
for i=2:length(tSpan)
yprime = diff_example(tSpan(i-1),y(i-1));
y(i) = y(i-1) + dt*yprime;
end
%% Plot Solutions
figure(1)
plot(tSpan,y)
grid on
xlabel('Time')
ylabel('y(t)')
```

We have to know the start time, the end time for the computation, the initial value of y and the step size. Smaller step size as we saw in lecture gives a more accurate solution, though it takes longer to compute the solutions.

b. MATLAB's ode45

The Runge-Kutta method is more complicated and requires many more steps in the algorithm. While this is not impossible to code, MATLAB has a function that makes its implementation very easy.

Step 1. Define the function for the differential equation as before. Be aware that the function inputs have to be listed in the order (t,y). Variable names don't matter in fact, and it should be (independent_variable, dependent_variable)

Step 2. Define the parameters to compute the solution:
Initial condition, start time, end time

Step 3. Solve the differential equation using MATLAB's built in solver:

```
[t_out, y_out] = ode45(@diff_func,[tStart tEnd], yI)
```

Go to slides from this point on.