Lab 13:
Ordinary Differential Equations

13.1 Introduction

This lab is aimed at introducing techniques for solving initial-value problems involving ordinary differential equations using MATLAB. Specifically, the lab will focus on solving equations that can be written as:

$$\frac{dy}{dt} = f(t, y, k)$$

where y represents an *array* of dependent variables, t represents the independent variable, and k represents whatever array of constants might be needed for the system. Note that although the equation above is a first-order differential equation, higher-order equations can be re-written to satisfy the form above. The information you will need to complete this lab is mainly written on the Pundit page `MATLAB:Ordinary Differential Equations` so you should be sure to read through that page and the examples. By the end of lab, you should be able to write functions to model and solve for multiple-dimension linear and nonlinear ordinary differential equations.

13.2 Resources

The additional resources required for this assignment include:

- Books: Chapra
- Pratt Pundit Pages: `MATLAB:Ordinary Differential Equations` and the two subpages linked therein
  - `MATLAB:Ordinary Differential Equations/Templates`
  - `MATLAB:Ordinary Differential Equations/Examples`

13.3 Getting Started

1. Log into one of the PCs in the lab using your NET ID. Be sure it is set to log on to acpub.
2. Start X-Win 32 on the PC. Roll the mouse over the X to make sure it is set to Display 0. If not, quit all instances of X-Win 32 and re-open X-Win 32.
3. Start PuTTY on the PC. Load a session, make sure X11 Tunneling is set in the SSH category and connect.
4. Once connected to a machine, switch into your `EGR53` directory and create a `lab13` directory inside it:

   ```
   cd EGR53
   mkdir lab13
   ```

5. Switch to your `~/EGR53/lab13` directory:

   ```
   cd lab13
   ```

6. Copy all relevant files from Dr. G's public `lab13` directory:

   ```
   cp -i ~mrg/public/EGR53/lab13/* .
   ```

   Do not forget the "." at the end.
7. Open MATLAB by typing `matlab &` at the prompt that appears in your terminal window. It will take MATLAB a few seconds to start up.

13.4 Problems

13.4.1 Chapra Chapter 20, Problem 4

This problem investigates population growth using the population of the world. You will complete the differential function and the execution script to run it in order to simulate this system and determine a good value for the growth rate. You will use the data contained in the table presented with the problem. One of the files you copied is the `WorldPop.mat` file, which has vectors called `t` and `p` for the time in years and the population in millions, respectively.

You will use a guess-and-check method for finding the appropriate value of k_g. To help with this, the execution script has a plot showing both the original data (`p` vs. `t`) and your calculated data `yout` vs. `tout`). In addition to completing the required file name and system initialization tasks in the first ten lines of the program, you will add a statistical analysis to find a coefficient of determination between the actual population data (`p`) and the data you obtained using `ode45` (`yout`).

Keep changing the value of the constant and re-running the script until you get an r^2 of at least 0.995. You should be able to figure out whether to make the value of the constant larger or smaller based on the graph. You will need to show the TA your differential function and your execution script as well as your final values for the constant k_g and the coefficient of determination r^2.

Note: this problem could be better solved using `fminsearch`, leaving as free parameters the initial population and the growth rate. The "tricky part" is that MATLAB's ODE solvers send out the interesting information - the `yout` values - as the *second* matrix, making it difficult/impossible to use a single anonymous function to construct the required `fSSR`.

13.4.2 Chapra Chapter 20, Problem 14

A more complicated model of population dynamics involves what is known as the *Lotka-Volterra equations*, also known as a predator-prey model. The model takes into account the fact that the predators and prey interact in ways that change the growth rate of each other. The equations can be written as follows (from Chapra, p. 504 with a slight change to make them state equations by writing the "prey" as y_1 and the "predators" as y_2):

$$\begin{aligned}\frac{dy_1}{dt} &= ay_1 - by_1y_2\\ \frac{dy_2}{dt} &= -cy_2 + dy_1y_2\end{aligned}$$

"where ... a = the prey growth rate, c = the predator death rate, and b and d = the rates characterizing the effect of the predator-prey interactions on the prey death and the predator growth, respectively."

Since there are two linked differential equations, the ODE solver needs to keep track of two values - the number of prey and the number of predators. The function with the differential equation must return the values of the derivatives of each population in a column vector.

You will use the data contained in the table presented with the problem - one of the files you copied is the `IsleRoyale.mat` file, which has vectors called `Year`, `Moose`, and `Wolves` for the time in years and the populations of the animals, respectively. Most of the execution script is written - you will need to add in the initial state values and the constants. You will have to write almost all the code for the differential equation file, however.

For this particular problem, a graph called a *phase plot* may be more useful than simply plotting the states themselves as functions of time. A phase plot consists of plotting one of the states of the equation versus the other - in this case, the Wolves vs. the Moose (literally). The code to accomplish this, `PredPreyPlot.m`, is listed at the end of the lab handout.

Note: This is another case where `fminsearch` may have come in handy. For one thing, the statistics should show that the model fairly accurately predicts the moose count but is woefully inaccurate when it comes to wolves. The `fminsearch` command could be used with *six* free parameters for this initial value problem - the four Lotka-Volterra parameters and the two initial animal populations - to minimize some statistical measure. In this case, some two-dimensional version of the coefficient of determination would be needed since there are two variables being modeled.

13.4.3 Cannon Ranging

Another common example using differential equations involves projectile motion. If a cannonball is launched into the air from some initial location $(r_x(0),\ r_y(0))$ at some initial velocity $(v_x(0),\ v_y(0))$ on Earth, then its motion can be modeled by the equations:

$$\frac{d^2r_x(t)}{dt^2} = 0 \qquad\qquad \frac{d^2r_y(t)}{dt^2} = -g$$

Since these are second order differential equations, they must be broken down into two first-order differential equations each. For the x-direction variables, the position r_x can be stored in the first state (y_1) and the velocity v_x can be stored in the second (y_2). The differential equation for the first state is simple - the derivative of the x position is the x velocity, and so the derivative of the first state is the second state:

$$\frac{dy_1}{dt} = y_2$$

The derivative of the x velocity is the x acceleration, so the function for the derivative of the second state comes from the left-hand second-order differential equation above; that is:

$$\frac{d^2r_x(t)}{dt^2} = \frac{dy_2}{dt^2} = 0$$

For the y-direction variables the position r_y can be stored in the third state (y_3) and the velocity v_y can be stored in the fourth (y_4). As above for the derivative of the x position, the derivative of the y position is the y velocity so the derivative of the third state is the fourth state:

$$\frac{dy_3}{dt} = y_4$$

The derivative of the y velocity is the y acceleration which comes from the right-hand second-order differential equation above:

$$\frac{d^2r_y(t)}{dt^2} = \frac{dy_4}{dt^2} = -g$$

Given all this, you will complete the function `CannonDiff` to return the four first derivatives for the four states - gravity should be passed in using the `const` variable. Then finish the script `RunCannonDiff.m` that will use `CannonDiff` to solve the differential equation for a cannonball with an initial position of (0 m, 10 m) and an initial velocity of 20 m/s at some given angle. Your job is to use the code to figure out the two unique angles, in degrees, required to get the cannonball to hit a target at (20 m, 0 m) and then determine the times at which the cannonball hits for each. Your ranging must be such that the distance from the point (20 m, 0 m) to the nearest calculated position of the cannonball must be less than 0.1. This will require guess-and-check with the angle as well as using more than 100 points in the time span. You may also want to change your ending time if the cannonball either isn't landing or is going thousands of meters under ground.

There are some plot commands already in the code to help with this task. The top left plot in the first figure shows the y values as a function of time, the bottom right plot shows the x value as a function of time *except* for this plot the independent and dependent axes are swapped. The top right plot shows y as a function of x. In each case, a blue line represents the desired x value and a red line represents the desired y value. In the zoomed plot and the top right plot of the first figure, the target is marked with a purple star. You will need to show the TA your initial angles and the times it takes for the cannonball to hit the target.

Note: (you knew it was coming) - This is yet *another* case where `fminsearch` would have been able to help solve the problem. In fact, `fminsearch` would have been able to find not only the angle but also the exact time at which the cannonball would hit the target by allowing the maximum time to also be a free parameter.

13.5 Templates

13.5.1 Differential Equation Template

```
function dydt = GeneralDiff(t, y, k)
% Template for calculating first derivatives of state variables
% t is time
% y is the state vector
% k contains any required constants
% dydt must be a column vector
dydt =
```

13.5.2 Script Template

```
% Template for using an ODE solver in MATLAB
% tout and yout will be the time and state variables

% Be sure to change name of file containing derivatives,
% time span, initial values, and any constants, as well
% as setting the flag for whether to make state plots

% Set name of file containing derivatives
DiffFileName = '';

% Set up time span, initial value(s), and constant(s)
% Note: Variables should be in columns
tspan = ;
yinit = ;
k     = ;

% Determine if states should be plotted
PlotStates = 1;

%% Under the hood
% Use ODE function of choice to get output times and states
DE = eval(sprintf('@(t, y, k) %s(t,y,k)', DiffFileName))
[tout, yout] = ode45(@(t,y) DE(t,y,k), tspan, yinit);

% Plot results
if PlotStates
    StatePlotter(tout, yout)
end
```

13.5.3 Predator-Prey Data Plotter

```
% PredPreyPlot.m
% Michael R. Gustafson II - 2007

% Requires Time, Prey, Pred, PreyHat, and PredHat
% If no PredHat or PreyHat, they are set to 0

% Axis limits
TAXIS = [min(Time) max(Time)];
TAXIS = TAXIS + 0.1 * [-1 1] * diff(TAXIS);
XAXIS = [min([Prey; PreyHat]) max([Prey; PreyHat])];
XAXIS = XAXIS + 0.1 * [-1 1] * diff(XAXIS);
YAXIS = [min([Pred; PredHat]) max([Pred; PredHat])];
YAXIS = YAXIS + 0.1 * [-1 1] * diff(YAXIS);

% Plot values
subplot(2,2,1)
plot(Time, Pred, 'ks:', tout, PredHat, 'r--o');
xlabel('Time'); ylabel('Pred'); legend('Data', 'Estimates', 0)
title('Predator Population vs. Time');
axis([TAXIS YAXIS])

subplot(2,2,3)
plot(Time, Prey, 'ks:', ...
     tout, PreyHat, 'r--o');
xlabel('Time'); ylabel('Prey'); legend('Data', 'Estimates', 0)
title('Prey Population vs. Time');
axis([TAXIS XAXIS])

subplot(2,2,4)
plot(Prey, Time, 'ks:', ...
      PreyHat, tout,'r--o');
xlabel('Prey'); ylabel('Time'); legend('Data', 'Estimates', 0)
title('Time vs. Prey Population');
axis([XAXIS TAXIS])

subplot(2,2,2)
plot(Prey, Pred, 'ks:', ...
     PreyHat, PredHat, 'r--o');
xlabel('Prey'); ylabel('Pred'); legend('Data', 'Estimates', 0)
title('Predator Population vs. Prey Population');
axis([XAXIS YAXIS])
```