AMS 27L **LAB #8** **Winter 2009**

Solving ODE's in Matlab

Objectives:

1. To use Matlab's ODE Solvers
2. To practice using functions and in-line functions

1 Matlab's ODE Suite

Matlab offers a suite of ODE solvers including: `ode23, ode45, ode15s, ode23s, ode23t, ode23tb, ode113` and `ode15i`. The types of differential equation solvers include nonstiff and stiff problems, and fully implicit ODEs. The AMS upper division course, **AMS 147: Computational Methods and Applications**, teaches numerical methods for solving ODE's such as the Runge-Kutta, trapezoidal rule, iterative methods and more, for solving problems in sciences and engineering. In this course, we will simply learn how to use the Matlab ODE solvers, rather than learning the various methodological logistics. In this lab, we will focus on the general purpose solver, `ode45` uses an explicit Runge-Kutta method to solve ODEs, and is usually the best function to use as a "first try" for most problems.

2 Single First Order Differential Equations

We are looking at an initial value problem of the form $x' = f(t, x)$, with $x(t_0) = x_0$.

The calling syntax for using `ode45` to find an approximate solution is:

`ode45(odefcn, tspan, x0)`, where
`odefcn` calls for a functional evaluation of $f(t, x)$,
`tspan=[t0,tfinal]` is a vector containing the initial and final times, and
`x0` is the x-value of the initial condition.

Example 1. Use `ode45` to plot the solution of the initial value problem

$$x' = \frac{cos\ t}{2x - 2}, \quad x(0) = 3,$$

on the interval $[0,\ 2\pi]$.

Note that the equation is in normal form $x' = f(t, x)$, where $f(t, x) = cos\, t/(2x - 2)$. The problem indicates that `tspan` = `[0,2*pi]` and $x0 = 3$. We need to encode the `odefcn`.

Open your editor and create an ODE function M-file with

```
function xprime = ex1(t,x)
xprime = cos(t)/(2*x - 2);
end
```

and save the file as `ex1.m`.

Problem #1. What is the value of the function, $f(t, x)$, when $t = 0$ and $x = 2$?

Plotting ODE Solutions. A quick way to plot the solution is to enter

```
>>  ode45(@ex1, [0,2*pi],3)
```

An alternative way to code the function is to create an inline function, such as

```
>>  f = inline('cos(t)/(2*x - 2)','t','x')
```

Problem #2. Use an inline function to plot the solution using `ode45`.

In the previous lab, we learned how to write function and subfunction M-files. Solving and plotting solutions to ODEs by writing a subfunction is really handy because it is easy to reproduce the result of a subfunction, once it's written.

---

3 Systems of First Order Equations

Systems are no harder to handle using `ode45` than are single equations. We simply write the system in vector form. For example, let's use `ode45` to solve the initial value problem

$$\begin{aligned} x_1' &= x_2 - x_1^2 \\ x_2' &= -x_1 - 2x_1x_2 \end{aligned}$$

on the interval $[0, 10]$ with initial conditions $x_1(0) = 0$ and $x_2(0) = 1$. We can write the system as the vector equation

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}' = \begin{bmatrix} x_2 - x_1^2 \\ -x_1 - 2x_1x_2 \end{bmatrix}$$

Open your editor and create the following ODE file

```
function xprime = F(t,x)
   xprime = zeros(2,1);     % The output must be a column vector
   xprime(1) = x(2) - x(1)^2;
   xprime(2) = -x(1) - 2*x(1)*x(2);
```

and save the file as `F.m`. The line `xprime = zeros(2,1)` initializes `xprime`, creating a **column** vector with two rows and 1 column.

Problem #3. What is the value of $F(t, [x_1, x_2])$ when $[x_1, x_2] = [3, 4]$? Does the value of $F(t, [x_1, x_2])$ change for different values of t?

Before we call `ode45` we need to enter the initial conditions for x_1 and x_2, which was $x_1(0) = 0$ and $x_2(0) = 1$. We do this by entering `x0 = [0;1]` so that x0 is a column vector.

Problem #4. Enter the ODE system by typing `[t,x] = ode45(@F,[0,10],[0;1]);` at the Matlab prompt. Then, type `whos` to see the list of variables. What are the sizes of the variables t and x? Why are they so large? Use the documentation for `ode45` to investigate the outputs.

Problem #5. I've written a function that includes the following lines of code in **plotF.m**. Download the M-file from our course webpage at `http://www.soe.ucsc.edu/classes/ams027l/Winter09/` and give it a go by typing `plotF` at the Matlab prompt. The function **plotF.m** should produce an image similar to Figure 1.

(a) *Fill in the blank:* The phase plane plot indicates that the solutions, x_1 and x_2, are $\underline{\text{(linear and cyclic)}, \ \text{(non}-\text{linear and cyclic)}}$.

You can take a look at the values of t and x by typing: `[t,x]` at the Matlab prompt. These are the Matlab commands that plot the solution to the ODE in different forms.

```
>> plot(t,x)
>> title('x_1'' = x_2 - x_1^2 and x_2'' = -x_1 - 2x_1x_2')
>> xlabel('t'), ylabel('x_1 and x_2')
>> legend('x_1','x_2'), grid
```

It is also possible to plot the components of the solution against each other with the commands

```
>> plot(x(:,1),x(:,2))
>> title('x_1'' = x_2 - x_1^2 and x_2'' = -x_1 - 2x_1x_2')
>> xlabel('x_1'), ylabel('x_2'), grid
```

The result is called a *phase plane* plot. Another way to present the solution to the system graphically is in a three dimensional plot, where both components of the solution are plotted as separate variables against the independent variable t by typing

```
>> plot3(t,x(:,1),x(:,2))    To see the solution from one perspective
>> plot3(x(:,1),x(:,2),t)       ...or yet another perspective.
```

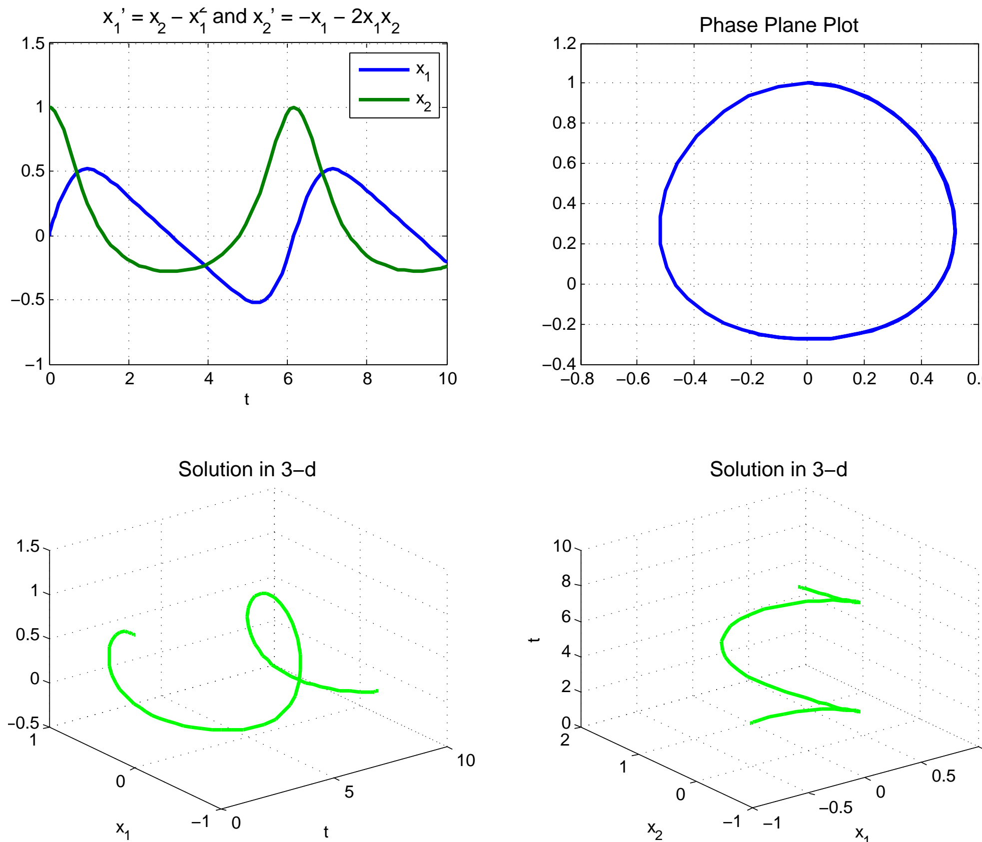

Figure 1: Graphic Techniques of the solution to the first order ODE System.

4 Parameter Passing and Global Variables

Variables defined in Matlab functions are only "known" to the function itself (*i.e.* not outside the function). In other words, the variable **scope** is **local**. We can pass parameters between functions, so that they can access the variable value. We've already had some practice with writing functions with more than one parameter. In the previous lab, we programmed a function and subfunction, that modeled a simple RC-circuit. The main function passed several variables to the

subfunction to calculate the voltage response. We'll write another set of functions that will pass parameters between the function and subfunctions. Another way to make variables accessible to subfunctions is to declare them as **global**. That will make the scope of the variables **globally accessible**. We won't discuss this method today.

The solvers in MATLAB can solve first order systems containing as many equations as you'd like. Let's take a look at a system of three equations that represents a simplified model for atmospheric turbulence beneath a thunderhead, called the Lorenz system of equations.

$$\begin{aligned} x' &= -ax + ay, \\ y' &= rx - y - xz, \\ z' &= -bz + xy \end{aligned}$$

where a, b and r are positive constants.

To make things more simple, let's set $u_a = x,\ u_2 = y$, and $u_3 = z$. With these substitutions, system of equations becomes:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}' = \begin{bmatrix} -au_1 + au_2 \\ ru_1 - u_2 - u_1u_3 \\ -bu_3 + u_1u_2 \end{bmatrix}$$

Problem #6. I've written a function M-file to find the solution of the Lorenz system of equations with the constants $a = 10$, $b = 8/3$ and $r = 28$ and initial conditions

$$u(0) = \begin{bmatrix} u_1(0) \\ u_2(0) \\ u_3(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Download the M-file **plotLorenz.m** from our class website. The function **plotLorenz.m** uses the ode45 Matlab solver to solve the system of equations. The first two parameters for ode45 are the time span and initial conditions. Open the function file in the Matlab editor. Notice the use of [ ] as a parameter in the `ode45` command. This is just a place holder in the command call for the use of **options**. We'll talk about the options available to the `ode45` solver later. The **plotLorenz.m** function will plot the solution and ought to produce an image similar to Figure 2.

---

5 Eliminating Transient Behavior

In many mechanical and electrical systems we experience *transient behavior*. This involves behavior that is present when the system starts, but dies out quickly in time, leaving only a fairly regular *steady-state* behavior. In Figure 2 there appears to be transient behavior until about $t = 1$. After $t = 1$ the solution settles down.

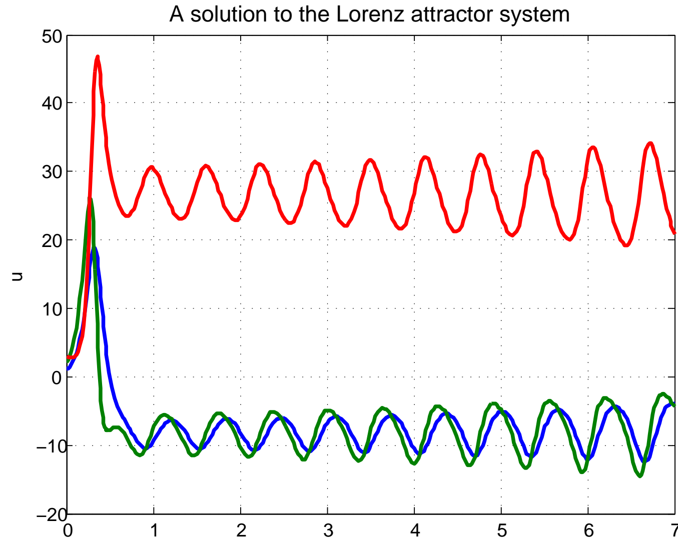

Figure 2: Solution traces for X,Y and Z over the time span from 0 to 7.

Problem #7. To examine the situation more closely, let's compute the solution over a longer period, say $0 \leq t \leq 100$, and then plot the part corresponding to $t > 10$ in three dimensions. Revise the function **plotLorenz.m** to incorporate these changes. The Matlab function `find` returns the indices that satisfy a logic conditional. For example:

`N = find(t>10);`	*Produces a list of the indices of those elements of t for which t¿10.*
`V = U(N,:);`	*Produces a matrix containing the rows of U with indices in N.*

Problem #8. We also would like to allow for randomly chosen initial values in order to see if the steady-state behavior is somehow independent of the initial conditions. We can use the Matlab command `rand` for this. The random number generator, `rand` produces random numbers in the range $[0, 1]$. However, we want each component of our initial conditions to be randomly chosen from the interval $[-50, 50]$. Easy! We can scale the output from `rand` using the command `u0 = 100*(rand(3,1) - 0.5)`. The M-file **randLorenz.m** from our class website does just this for us. Download the function M-file **randLorenz.m** and give it a go.

The figure produced by **randLorenz.m** is a butterfly-like approximation of the *Lorenz attractor*, as seen in Figure 3. With this particular choice of parameters, any solution, no matter what the choice of initial conditions, is attracted to this rather complicated, butterfly-like, set.

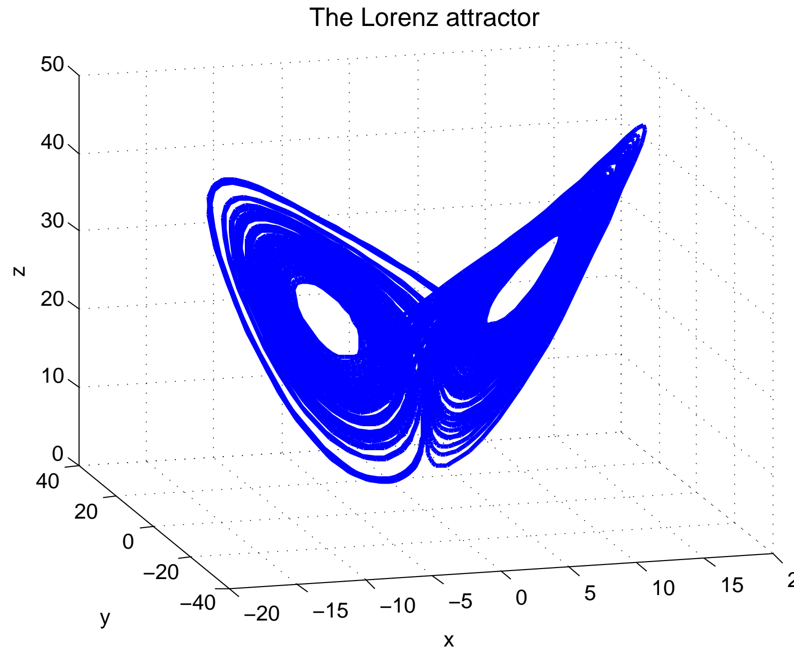

Figure 3: 3-d plot of the solution to the Lorenz attractor.

Problem #9. Click on the rotate icon in the toolbar, and click and drag the mouse in the axes in the butterfly figure to rotate the axes to a different view of the Lorenz attractor.

6 Stiff Equations

Solutions to differential equations often have components which are varying at different rates. If these rates differ by a couple of orders of magnitude, the equations are called stiff. For example, the equation $x' = e^t \, cos\, x$ is a **stiff equation** because the factor e^t gets very large. To see what this does to `ode45`,

Problem #10. Execute the following lines and notice the speed of the plot

```
>> f = inline('exp(t)*cos(x)','t','x');
>> ode45(f,[0,50],0)
```

When you are tired of waiting for the solution to end, click on the **Stop** button. This very slow computation is the typical response of `ode45` to stiff systems of equations. The fast rate of change in the function, f, requires `ode45` to take extremely small steps, and therefore a takes a long time

to complete the solution.

The Matlab suite of ODE solvers includes routines designed to solve stiff equations, including `ode15s, ode23s, ode23t` and `ode23tb`. `ode15s` is the first of these to try.

Problem #11. Execute the Matlab function `ode15s(f,[0,50],0)` and compare the speed of the solution to the previous problem.

So how do we tell when a system is stiff?? It's often obvious from the physical situation being modeled that there are components of the solution which vary at rates that are significantly different, and therefore the system is stiff. However, there is no general rule that allows us to recognize stiff systems.

Rule of Thumb: Try to solve using `ode45`. If that fails, or is way slow, try `ode15s`.

Quit MATLAB by clicking on the `File` menu in the upper left corner and choosing `Exit`. Please remember to **Log Off** (from the "Start" menu in the lower left of the screen).