Using Matlab's ode45 to solve ODEs

CS 310 — Fall 1999

1 Solving A Simple ODE with ode45

To get started using Matlab's ode45 function, we will solve this differential equation:

$$y' = \sin(y) - \sin(t)$$

with $y(0) = 1$.

You first need to write a Matlab function that will evaluate the right-hand side of this ODE. To do this, create a file called `func.m`, and in it put these lines:

```
function yprime = func(t,y)
yprime = sin(y) - sin(t);
```

The first line means that this file contains a function, and it is called `func`. `func` has two arguments, `t` and `y`. It returns the value `yprime`.

The function `func` calculates the right-hand side of the differential equation we want to solve. The name `func` must be the same as the name of the file that you put the function in, in this case `func.m`.

Create another file called `testode.m` with the following lines in it:

```
clear
[t,y] = ode45('func', [0,10], 1)
plot(t,y)
```

Then, run Matlab, and run the script `testode.m`

This program calls `ode45` to solve the ODE that is given in the function `func`, and then it plots the curve $y(t)$ that is calculated by `ode45`.

In the line `[t,y] = ode45('func', [0,10] ,1)`, the 0 and the 10 are the initial and final values of t. The 1 is the initial value of y. Since our initial condition was $y(0) = 1$, we started y at 1 and t at 0.

Since we didn't put a semi-colon at the end of the line, the values of the vectors `t` and `y` are printed after `ode45` finishes.

2 Solving Coupled First-Order ODEs

Now, let's try solving two ODEs that are coupled:

$$y_1' = \cos(y_2) + \sin(t)$$
$$y_2' = \sin(y_1) - \cos(t)$$

where $y_1(0) = 1$ and $y_2(0) = 2$.

In this problem, we have two functions to worry about since we have two right-hand sides. Instead of writing two functions, we need to write one function that will calculate both right-hand sides.

Create a file `func2.m` with these lines in it:[1]

```
function zprime = func2(t,z)
y1 = z(1);
y2 = z(2);
y1prime = cos(y2) + sin(t);
y2prime = sin(y1) - cos(t);
zprime = [y1prime; y2prime];
```

The input variable `z` is now a vector. (In the previous example, the input variable was called `y` and was just a number.) This vector contains two numbers, the first is y_1 and the second is y_2. The second and third lines of the function extract from `z` each of these variables.

Once we have `y1` and `y2` as separate variables, we can calculate the right-hand sides of the ODEs. These values are put into the variables `y1prime` and `y2prime`.

The last thing the function does is assemble the calculated quantities into the vector `zprime`, which is the derivatives of the variables in `z`.

Create another file called `testode2.m` with the following lines in it:

```
clear
tspan = [0, 10];
z0 = [1, 2];
[t,z] = ode45('func2', tspan, z0)
plot(t,z)
```

Then, run Matlab and run the script `testode2.m`

[1] The following lines could be used instead in `func2.m`

```
function zprime = func2(t,z)
zprime = [cos(z(2)) + sin(t); sin(z(1)) - cos(t)];
```

Do you see why the two files do the same thing?

This will solve the problem from $t = 0$ to $t = 10$, using the initial conditions $y_1(0) = 1$ and $y_2(0) = 2$. This time we created variables for the initial and final times, and the initial conditions, but you could have just typed `[t,z] = ode45('func2',[0,10],[1,2])`

You can see that `z` contains two columns. The first contains the values of y_1, and the second contains the values of y_2. When you say `plot(t,z)`, it will plot both curves. If you want to plot just one of the curves, you could use a statement like `y1 = z(:,1);` or `y2 = z(:,2);`. You would then substitute `y1` or `y2` for `z` in the `plot(t,z)` statement.

3 General ODEs

Not all differential equations are first-order or combinations of several first-order equations. In fact, many of the differential equations encountered in engineering are second order. However, any ODE can always be converted to a system of several first-order ODEs.

To solve a second-order ODE like

$$x'' + x' + x = \sin(t)$$

you would first need to convert the equation to standard form. Once you've done this, you will have the situation described in Section 2: two first-order ODEs that are coupled. You can then use the method described there.

For example, we can convert the above equation to standard form by substituting x_1 for x'. This gives us the two first-order equations:

$$\begin{aligned} x' &= x_1 \\ x_1' &= -x_1 - x + \sin(t) \end{aligned}$$

So, your Matlab function could look like this:

```
function zprime = func2(t,z)
x = z(1);
x1 = z(2);
xprime = x1;
x1prime = -x1 - x + sin(t);
zprime = [xprime; x1prime];
```

When you get the solution to this system, you have values for x and for x_1. (x would be in the first column of `z` and x_1 (which is really x') would be in the second column.)

☞ Make a plot from $t = 0$ to $t = 20$ of the solution to the above differential equation using the initial conditions $x(0) = 5$ and $x'(0) = 6$.

☞ Make a plot of $x(t)$, $x'(t)$ and $x''(t)$ from $t = 0$ to $t = 3$, where $x(t)$ satisfies the ODE

$$x''' + \cos(t)x'' - tx + x = \sin^2(t)$$

and $x(0) = 1$, $x'(0) = x''(0) = 0$.

4 The Chaotic Pendulum

This section has to do with a compound pendulum whose motion is chaotic for most starting points. The pendulum is displayed in Figure 1. It consists of two masses on the ends of a bar that is free to rotate on the end of a pendulum. The length of the pendulum is L_0, and the bar holding the masses has length $L_1 + L_2$. The mass M_1 is at a length L_1 from the point of contact with the pendulum. Similarly, mass M_2 is at a length L_2 from the point of contact with the pendulum. The coordinates that describe the position of the pendulum

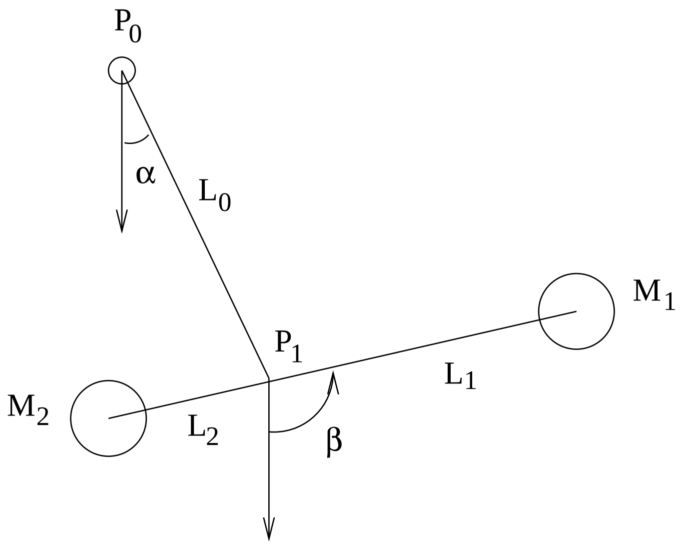

Figure 1: A diagram of the pendulum

system are the two angles α and β. The angle α is measured from the vertical down from the top pivot point P_0 to the bar with length L_0. The angle β is measured from the vertical down from the pivot point P_1 at the end of the bar L_0 to the portion of the bar with length L_1 and holding mass M_1.

The quantities that determine the system are the three lengths and the two masses. In addition, we define the quantities

$$B = \frac{M_1 L_1 - M_2 L_2}{M_1 + M_2}$$
$$\bar{L} = \frac{M_1 L_1^2 + M_2 L_2^2}{L_0(M_1 + M_2)}$$

since they will arise frequently. The magnitude of B is a measure of how balanced the two masses are around the pivot point.

The motion is determined by the masses and lengths, as well as the initial conditions for the motion. The equations are:

$$L_0\ddot{\alpha} + B\cos(\alpha-\beta)\ddot{\beta} = -G\sin\alpha - B\sin(\alpha-\beta)\dot{\beta}^2$$
$$B\cos(\alpha-\beta)\ddot{\alpha} + \bar{L}\ddot{\beta} = -\frac{GB}{L_0}\sin\beta + B\sin(\alpha-\beta)\dot{\alpha}^2$$

We use a dot over a variable to denote the derivative with respect to time. Two dots over a variable means the second derivative of that variable with respect to time. G is the gravitational acceleration. These equations only depend on the relative masses, so we can restrict to the case where $M_1 + M_2 = 1$. To reduce the number of parameters, we'll also assume that $L_0 = 1$, and that $L_1 + L_2 = 1$.

5 Solving this Problem with Matlab

The files that you will need for this section are available from the course Web page in the Matlab folder. The two relevant files are `pendulum.m` and `chaotic.m`. Copy these files into your matlab directory.

To solve this problem, we will use Matlab's `ode45` function, which solves differential equations. To use it, we must first convert the given differential equations to standard form. Since each equation is second order, we need to introduce one additional variable for each equation, which means we will get four equations. Then we need to write a function that will evaluate the right-hand sides of these four equations. This is the function contained in the file `pendulum.m`

It looks like this:

```
function zp = pendulum(t, z)

global B GG Lbar L0

% Convert from standard form to meaningful names.

alpha    = z(1) ;
beta     = z(2) ;
alphadot = z(3) ;
betadot  = z(4) ;

% Compute the right-hand sides of the equations for the dot-dot variables.
```

```
rhs(1) = -B*sin(alpha - beta)*betadot*betadot   -GG*sin(alpha) ;
rhs(2) =  B*sin(alpha - beta)*alphadot*alphadot -GG*sin(beta )*B/L0 ;

% Set up the matrix.

C(1,1) =  L0 ;
C(1,2) =  B * cos( alpha - beta ) ;
C(2,1) =  B * cos( alpha - beta ) ;
C(2,2) =  Lbar ;

% Solve for the first two dot-dot variables.

tmp  =  C \ rhs' ;
alphadotdot = tmp(1) ;
betadotdot  = tmp(2) ;

% Convert back to standard form.
% Note that the variables in zp correspond to the dots of the variables in z.

zp = [ alphadot; betadot; alphadotdot; betadotdot; ] ;
```

This is somewhat more complicated than the examples in Sections 1 and 2, but it really is doing the same thing. The file `pendulum.m` contains quite a few more comments; read them for a more thorough explanation.

The file `chaotic.m` contains commands that will solve the problem using `ode45`, and it will plot the results in various ways. You don't need to worry about understanding how `chaotic.m` works. This is the script you should run within Matlab.

Try different values for the masses, lengths and initial conditions for the angles. The initial conditions for $\alpha, \dot{\alpha}, \beta$ and $\dot{\beta}$ describe the starting position of the pendulum and its starting velocity. The program sets $\dot{\alpha}(0)$ and $\dot{\beta}(0)$ to zero, meaning that the pendulum starts out at rest. You may modify the program to accommodate situations where the pendulum starts with some initial velocity.

For most initial conditions, the motion of the masses is rather chaotic. That is, it is not periodic, and it doesn't have any nice pattern. For some special cases, the solution does have a periodic solution. One such special case is if the masses balance around the pivot point, which happens when $B = 0$, i.e., when $L_1M_1 = L_2M_2$. In this case the angle β remains constant and the pendulum moves as if the total mass $M_1 + M_2$ were located at the pivot point P_1. Try it!