Some Useful MATLAB Functions for AE252 (Part 2)

February 7, 2007

So far we have used MATLAB to plot a function of one variable and to plot a parametric function. Now we would like to use MATLAB to integrate equations of motion numerically.

If you need more help. A useful introductory reference is *Getting Started with MATLAB 7: A Quick Introduction for Scientists and Engineers* (Rudra Pratap, Oxford University Press, 2006). See the website `http://www.mathworks.com/support/books/book11696.html`.

A note about numerical integration. Let $x(t)$ be a function of time. Say we already know the value of $x(t_0) = x_0$ and $\dot{x}(t_0) = v_0$. Then from basic calculus we can approximate the value of $x(t_0 + \Delta t)$ for any small increment of time Δt by

$$\begin{aligned} x(t_0 + \Delta t) &\approx x(t_0) + \dot{x}(t_0)\Delta t \\ &= x_0 + v_0 \Delta t \end{aligned}$$

If we repeat this process over and over, we can construct an approximation to the curve $x(t)$. This is the basic idea of numerical integration. For more information, see Section 14.6 in your text (Bedford/Fowler, p148-152). In other courses you will learn a lot more about good ways of doing numerical integration. Luckily, there are many tools other people have created to do the integration for us. All we have to do is know how to use them.

Integrating equations of motion. Let's say you derived the equation of motion for a simple pendulum:

$$\ddot{\theta} = -\frac{g}{l} \sin\theta.$$

This expression is difficult to integrate analytically. (In fact, it can be done in terms of elliptic functions, but we don't know anything about those yet.) So instead, we will use MATLAB to integrate it numerically. Unfortunately, the equation of motion is a second-order ordinary differential equation (because it has a second-derivative, $\ddot{\theta}$. For MATLAB to understand this equation, we have to transform it into two first-order ordinary differential equations. This is called putting the equation of motion in *state variable form*.

We have one variable, θ. So we define states $y_1 = \theta$ and $y_2 = \dot{\theta}$. Then we can rewrite our equation of motion as follows:

$$\begin{aligned} \frac{dy_1}{dt} &= y_2 \\ \frac{dy_2}{dt} &= -\frac{g}{l} \sin y_1 \end{aligned}$$

Now we are ready to integrate. It is easiest to do this by creating a program, or an *M-file*, for MATLAB to execute. Open an editor and save a new file called "pendulum.m" which contains the following text:

```
function pendulum
disp('Hello!');
```

Now, if you were to type "pendulum" at the MATLAB prompt, you would see the message "Hello!" as follows:

```
>> pendulum
Hello!
```

The first thing we will do is add a subroutine to "pendulum" that describes our equations of motion. We add several lines at the bottom of the file:

```
function dy=f(t,y)
g=9.8;
l=1;
y1=y(1,1);
y2=y(2,1);
dy(1,1)=y2;
dy(2,1)=-(g/l)*sin(y1);
```

The first line gives our subroutine a name, `f`, and says that this subroutine will output the derivatives `dy` given the time `t` and the states `y`. The second two lines define the constants in the equations of motion (in this case, we've assumed $l = 1$). The third two lines extract the state variables from the vector `y`. This is not necessary, but makes things easier to write. The final two lines describe our equations of motion—they compute dy_1/dt and dy_2/dt (encoded as `dy(1,1)` and `dy(2,1)`) given y_1 and y_2 (encoded as `y1` and `y2`).

Next we will define the initial and final times over which we want to integrate. Say we start at $t = 0$ and end at $t = 1$. Then we return to the beginning of `pendulum` and add two lines:

```
t0=0;
t1=1;
```

Next we need to define the initial conditions in our problem. Let's assume $\theta = 0$ and $\dot{\theta} = 1$ at $t = 0$. In other words, let's assume we start with $y_1 = 0$ and $y_2 = 1$. Then we add the following line:

```
y0=[0 1];
```

Remember that the brackets tell MATLAB that we are defining a vector. The first element in this vector corresponds to y_1, and the second to y_2. Finally we are ready to integrate by adding the line

```
[t,y]=ode45(@f,[t0 t1],y0);
```

This tells MATLAB to use the integration method `ode45` (a Runge-Kutta method), to integrate equations of motion that are defined by your subroutine `f`, to integrate from the initial time `t0` to the final time `t1`, and to use the initial conditions defined by the vector `y0`.

The results are returned in a vector of time values `t` (just like you used to generate with the command `linspace`) and a matrix of states `y` at each time. To be able to see the results, we recover θ and $\dot{\theta}$ from `y` as follows:

```
theta=y(:,1);
thetadot=y(:,2);
```

This tells MATLAB that everything in the first column of `y` (corresponding to `y1`) should be assigned to the variable `theta`, and that everything in the second column (corresponding to `y2`) should be assigned to the variable `thetadot`. Now, to plot θ as a function of time, we simply write

```
figure(1);
clf;
plot(t,theta);
```

We might also want to plot the trajectory in Cartesian coordinates. For the simple pendulum, we have the following (this is *NOT* the same for all problems!!!):

$$x = l \sin\theta$$
$$y = -l \cos\theta$$

So in MATLAB we write:

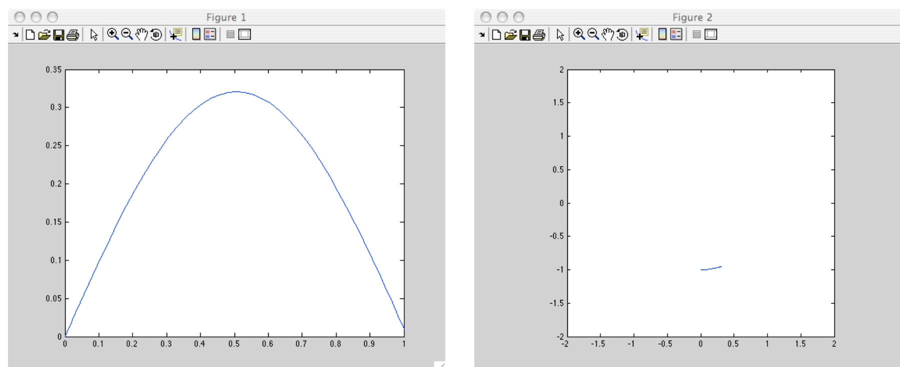
```
l=1;
x = l*sin(theta);
y = -l*cos(theta);
figure(2);
clf;
plot(x,y);
axis equal;
axis([-2 2 -2 2]);
```

Remember that it is a good idea to use the command `axis equal` when plotting a trajectory, so that both axes have the same scale. We also set the axis to a box around the origin of height and width 2, so that things look better.

To actually see the result of all this, we return to the command line and enter:

```
>> pendulum
```

The result is as follows:



If you want to see the entire thing, download `pendulum.m` from the course website.

Remember, if you ever have any questions about the syntax of a particular function, type "help [function-name]" like this:

```
>> help cos
 COS    Cosine of argument in radians.
    COS(X) is the cosine of the elements of X.

    See also acos, cosd.

    Overloaded functions or methods (ones with the same name in other directories)
       help distributed/cos.m
       help sym/cos.m

    Reference page in Help browser
       doc cos
```

Have fun!