Using Matlab to integrate Ordinary Differential Equations (ODEs)

Erica McEvoy*
(Dated: June 17, 2009)

1. INTRODUCTION

Ordinary differential equations tend to arise whenever you need to model changing quantities that depend on the amount of other quantities around it. For example, in chemistry, the time rate of change of concentration ($\frac{d}{dt}$) of a chemical solution often depends on the concetrations of other chemicals that surround it. In biology, differential equations are often used in population dynamics, to model the evolution and/or extinction of a particular species (like people, animals, bacteria, or even viruses like HIV) (eg., Volterra Equations). In finance, the stock market is often modeled via sets of coupled differential equations (e.g., Black-Scholes equation). In physics, dfq's are everywhere – we've seen them in Cosmology (e.g., Friedmann's Equations, non-linear structure growth and perturbation theory), Classical Dynamics (e.g., the orbits of planets, stars, and galaxies as specialized N-body problems, hydrodynamics), and Radiative Transfer. Most differential equations are too complicated to write down a solution by hand (an "analytical solution"), so one has to revert to numerics to find any kind of solution at all!

Here, I focus on Ordinary Differential Equations (where the functions we solve for depend on just one variable, like time), and how a few simple commands in Matlab allows us to get numerical solutions to even the most complicated looking equations.

2. 1ST ORDER DFQS

A 1^{st} order differential equation is an equation where the highest derivative of your function is one. A simple example would be (for some function $y(t)$)

$$\frac{dy}{dt} = -5y \tag{1}$$

From calculus, we all know that the solution to this equation is $y(t) = Ce^{-5t}$, where C is some arbitrary constant. If we specified an initial condition (say, $y(0) = 1.43$), then our analytical solution would be $y(t) = 1.43e^{-5t}$.

In Matlab, we can use numerical integration techniques to solve differential equations like this one. For the example above, you would make two .m files (one will be a function file, and the other will be a script that calls the function file). Use your favorite text editor to create and save a file called 'ilovecats.m' and copy the following into it:

```
function dy = ilovecats(t,y)
dy = zeros(1,1);
dy = -5 * y;
```

Now create another file called 'happyscript.m' and cut and paste the following into it:

```
[t,y] = ode45('ilovecats',[0 10], 1.43);

plot(t,y,'-');
xlabel('time');
ylabel('y(t)');
title('This plot dedicated to kitties everywhere');
```

*Electronic address: emcevoy@math.arizona.edu

Here in happyscript.m, we are using the built-in Matlab function called ode45. There are a few different built-in ode integrators that Matlab has (all of which you can find on the mathworks.com website, or simply typing in 'help ode45' at your prompt), but as a general rule of thumb, ode45 is the best function to apply as a first try for most problems. (For those who like detail, ode45 is an explicit (4,5) Runge-Kutta integrating technique). When using ode45, the first argument 'ilovecats' is the name of the function you defined in the file ilovecats.m (the name of the function must be delimited by a single quote!). The 2nd argument [010] specifies the time interval of integration (i.e., $t = 0$ until $t = 10$). The 3rd argument, 1.43, is the initial condition (i.e., $y(0) = 1.43$). $[t, y]$ is the set of data returned by ode45 – t is a vector that contains the time datapoints, and y is a vector that contains the y datapoints.

At your Matlab prompt, type happyscript. You should get the same image as Figure 1. That's the numerical solution! If you want to see the actual numbers that Matlab used to plot this picture, just type t (to see the time values it integrated at) or y (to see the y values at the corresponding t values).

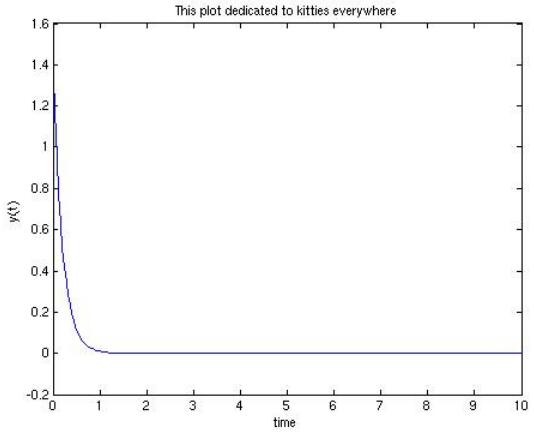

FIG. 1: This plot proves that kitties everywhere are just adorable

Now, if you're not convinced that this is the correct solution, you can compare your numerical solution to the exact one ($y(t) = 1.43e^{-5t}$) by looking at the error between both functions. Just append the following at the bottom of your happyscript.m file:

```
realsolution = 1.43 *exp(-5* t);
error = abs(y - realsolution);

figure;

subplot(221)
plot(t,y,'-');
xlabel('time');
ylabel('y(t) computed numerically');
title('Numerical Solution');
subplot(222)
plot(t,realsolution,'-');
xlabel('time');
ylabel('y(t) computed analytically');
title('Analytical Solution');
subplot(223)
plot(t,error,'-');
xlabel('time');
ylabel('Error');
title('Relative error between numerical and analytical solutions');
subplot(224)
plot(0,0,'.');
xlabel('time')
```

```
ylabel('Kitties!!!!');
title('Kitties are SO CUTE!');
```

To plot more than one figure, use the subplot command. For example, if you want three columns and two rows of plots on your figure, and you currently want to work in the first of those regions, type subplot(321) in your script. When you finish giving matlab commands for the first plot, then type the next command to work in the next subsection of the figure: subplot(322). If you append the above to your happyscript.m file, then you should get Figure 2. (If instead, you'd rather generate 4 separate plots, then replace each occurance of the subplot command with the command figure).

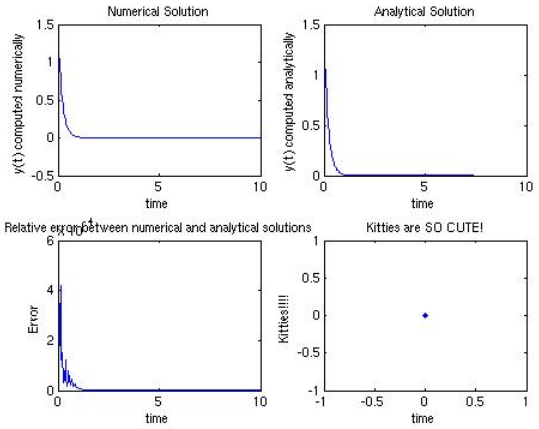

FIG. 2: The correlation between cats and cuteness gets tighter with time – cats get more adorable as they age!

3. 2ND ORDER DFQS

A 2nd order differential equation is one where the highest derivative term is of order 2. A classic example is the equation of motion for a single undamped pendulum (where ω^2 is $\frac{g}{lm}$, for those who derive this from Newtons Laws):

$$\frac{d^2\theta}{dt^2} = -\omega^2 \sin(\theta) \tag{2}$$

To integrate this in Matlab, we have to re-write this single equation into a set of 2 first order differential equations. (The reason behind this is because all Runge-Kutta solvers, including ode45, are built to only integrate over equations of the type $\frac{dy}{dt} = f(t, y)$). We can easily do this by hand, by setting:

$$\frac{dy_1}{dt} = y_2 \tag{3}$$

$$\frac{dy_2}{dt} = -\omega^2 \sin(y_1) \tag{4}$$

where $y_1(t)$ represents $\theta(t)$, and $y_2(t)$ represents $\frac{d\theta}{dt}$. Integrating these equations in Matlab is very similar to how we did it for the 1st order example in the previous section. Only now, call your function file 'pendulumcats.m' and copy the following:

```
function dy = pendulumcats(t,y)
dy = zeros(2,1);
omega = 1;
dy(1) = y(2);
dy(2) = -omega*omega*sin(y(1));
```

and call your script something else, like pendulumscript.m, and copy the following:

```
[t,y] = ode45('pendulumcats',[0 25], [1.0 1.0]);

plot(t,y(:,1),'-');
xlabel('time');
ylabel('y_{1}(t)');
title('\theta (t)');

figure;
plot(t,y(:,2),'-');
xlabel('time');
ylabel('y_{2}(t)');
title('d \theta / dt (t)');

figure;
plot(y(:,1),y(:,2),'-');
xlabel('\theta (t)');
ylabel('d \theta / dt (t)');
title('Phase Plane Portrait for undamped pendulum');
```

The change in the function file, pendulumcats.m, is the initialization part in line two – dy = zeros(2,1); This is because we now have two equations we are integrating over ($y_1(t)$ and $y_2(t)$), so Matlab will store their data points into a matrix with two columns. If you just type the letter y at your Matlab prompt, you will get 2 columns of data that display. The first column is the set of y(1) (or, $y_1(t)$), whose datapoints you can alone access by typing y(:,1) at your prompt. The second column of y are the datapoints for $y_2(t)$, which you can access by themselves by typing y(:,2) at your prompt.

The change in the script file comes when you use the ode45 command – now we are integrating from $t = 0$ until $t = 25$ seconds, and we specified that the initial conditions are $y_1(t = 0) = 1.0$ and $y_2(t = 0) = 1.0$ (via the third argument as [1.0 1.0]). Since we now have 2 sets of equations to integrate over, the initial conditions input must be a 2x1 vector (to match the size of the outgoing vector, y). (E.g., if $y_1(t = 0) = 33$ and $y_2(t = 0) = 42$, then [1.0 1.0] would be replaced by [33 42]). (In the 1st example at the beginning, we only had one initial condition, which was sufficient to represent as the scalar 1.43).

For those who are interested in phase plane portraits, you just replace the time variable, t, in your plot command, with the datapoints for the other function, so that you are effectively making a plot of position vs. velocity of the pendulum. Running the above commands should give you Figure 3 (sans subplots).

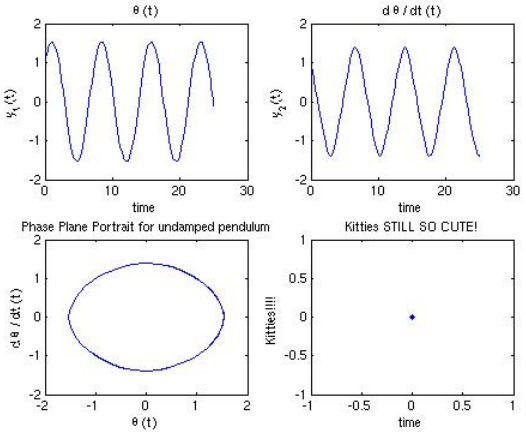

FIG. 3: Kitties will one day rule the world!

4. THE LORENZ EQUATIONS AND CHAOS

Back in the day, scientists didn't know as much, and thought they could accurately predict the weather once computers became more powerful. This is because weather people used many sets of differential equations to model the weather, and it took a long time to integrate those equations (keep in mind that awesome things like Matlab weren't around in the 50s and 60s – people still used slide rulers and tables to calculate many things, and the computers that were available back in the day had very little computing power, so integrating some ODEs, like those in the pendulum example, would take a crazy long time for the computer to chug through!).

Edward Lorenz was a mathematician and weather forcaster for the US Army Air Corps, and later an MIT professor. For many years, he was interested in solving a simple set of 3 coupled differential equations just because he wanted to "find out what the weather would be like during the next week." These equations are called the Lorenz Equations, and were derived from simplified equations of convection rolls rising in the atmosphere. They are pretty simple and can be expressed as:

$$\frac{dx}{dt} = -Px + Py \tag{5}$$

$$\frac{dy}{dt} = rx - y - xz \tag{6}$$

$$\frac{dz}{dt} = xy - bz \tag{7}$$

where P, r, and b are all constants (P represents the Prandtl number, and r is the ratio of Rayleigh number to the critical Rayleigh number), and x, y and z are all functions of time. (You can read more about what these equations represent in Lorenz's classic paper, $Deterministicnonperiodicflow.J.Atmos.Sci.20:130-141$). We can use Matlab to look at trajectories (i.e., plots of $x(t)$ vs. time, $y(t)$ vs. time, and $z(t)$ vs. time) or phase plane portraits (i.e., $x(t)$ vs. $y(t)$, $x(t)$ vs. $z(t)$, and/or $y(t)$ vs. $z(t)$) for this system. The function file (lorenz.m) should look like:

```
function dy = lorenz(t,y)
dy = zeros(3,1);
P = 10;
r = 28;
b = 8/3;
dy(1) = P*(y(2) - y(1));
dy(2) = -y(1)*y(3) + r*y(1) - y(2);
dy(3) = y(1)*y(2) - b*y(3);
```

(where we've chosen $P = 10$, $r = 28$, and $b = 8/3$ for our constants). And your script file should look like:

```
[t,y] = ode45('lorenz',[0 250], [1.0 1.0 1.0]);

subplot(221)
plot(y(:,1),y(:,2),'-');
xlabel('x(t)');
ylabel('y(t)');
title('Phase Plane Portrait for Lorenz attractor -- y(t) vs. x(t)');

subplot(222)
plot(y(:,1),y(:,3),'-');
xlabel('x(t)');
ylabel('z(t)');
title('Phase Plane Portrait for Lorenz attractor -- z(t) vs. x(t)');

subplot(223)
plot(y(:,2),y(:,3),'-');
```

```
xlabel('y(t)');
ylabel('z(t)');
title('Phase Plane Portrait for Lorenz attractor -- z(t) vs. y(t)');

subplot(224)
plot(0,0,'.');
xlabel('Edward Lorenz')
ylabel('Kitties');
title('Kitties vs. Lorenz');
```

Running the above script may take a little while to run (since there are now 3 equations, and we are integrating from $t = 0$ until $t = 250$), and should give you the same as Figure 3.

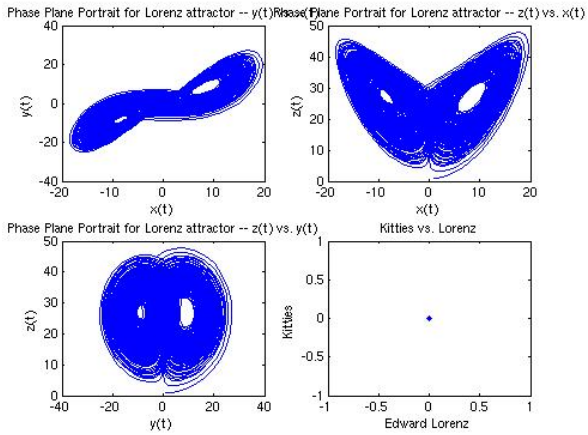

FIG. 4: Edward Lorenz vs. Kitties – who will win?

To make a 3D plot that you can play with (just use your mouse to click on the image, and you can rotate it in whichever direction you'd like), add the following to the bottom of your script:

```
plot3(y(:,1),y(:,2),y(:,3),'-')
xlabel('x(t)');
ylabel('y(t)');
zlabel('z(t)');
title('3D phase portrait of Lorenz Attractor');
```

These plots are pretty famous, and that crazy butterfly looking picture is known as a "strange attractor", or the "Lorenz attractor." Strange attractors appear in phase spaces of chaotic dynamical systems. Edward Lorenz is the first person to report such bizarre findings, and as such, he is often considered the father (or founder) of Chaos Theory. The "butterfly effect" was coined and described after studying the numerical solutions of these very equations. The idea is that chaotic systems have a sensitive dependence on initial conditions – if you were to play around with the initial conditions for $x(t)$, $y(t)$ and $z(t)$ in these equations and plot phase space portraits for each solution set, you'd find that even the tiniest changes in initial conditions can lead to a crazy huge difference in position in phase space at some later time (which is not what you'd expect if the equations were considered "deterministic" – you'd expect that equations that were almost identical to give you almost identical trajectories and phase space portraits at any time!) Because of this, chaotic systems like the weather are difficult to predict past a certain time range since you can't measure the starting atmospheric conditions completely accurately.

As far as Astrophysics and Astronomy is concerned, it's been shown (I think by MIT Prof. Jack Wisdom) that the orbits of the planets in our solar system are actually not deterministic, but chaotic, over a very long time period. As such, we won't be able to predict where any of the planets will be in our solar system, a long time from now. As for other things, it's probably important to look for signatures of chaos whenever you have any set of non-linear coupled differential equations that you are integrating over for a long period of time (e.g., N-body simulations?)

5. THE FRIEDMANN EQUATIONS

We can use ode45 to find solutions of the scale factor, $a(t)$, to the Friedmann equations:

$$(\frac{\dot{a}}{a})^2 = \frac{8\pi G}{3}\tilde{\rho} - \frac{k}{a^2} \tag{8}$$

$$\frac{\ddot{a}}{a} = \frac{-4\pi G}{3}(\tilde{\rho} + 3\tilde{P}) \tag{9}$$

6. THE 2-BODY PROBLEM

Consider 2 point masses, with masses m_1 and m_2, described by the two-dimensional coordinate positions $(x_1(t), y_1(t))$ and $(x_2(t), y_1(t))$. The gravitational force between both of these masses is given by $\frac{Gm_1m_2}{r_{12}^2}$, where $r_{12} = (x_1 - x_2)^2 + (y_1 - y_2)^2$. If we apply Newton's 2nd Law for both masses in each coordinate direction (x and y), we get a set of 4 coupled differential equations:

$$m_1\frac{dx_1^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}(x_2 - x_1) \tag{10}$$

$$m_1\frac{dy_1^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}(y_2 - y_1) \tag{11}$$

$$m_2\frac{dx_2^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}(x_1 - x_2) \tag{12}$$

$$m_2\frac{dy_2^2}{dt^2} = \frac{Gm_1m_2}{r_{12}^3}(y_1 - y_2) \tag{13}$$

To solve these equations in Matlab, we'll need to rewrite each of the 4 equations into two first order differential equations (ala the undamped pendulum example), thereby giving us a system of 8 differential equations to solve:

$$\frac{dx_1}{dt} = u_1 \tag{14}$$

$$\frac{du_1}{dt} = \frac{Gm_2}{r_{12}^3}(x_2 - x_1) \tag{15}$$

$$\frac{dy_1}{dt} = v_1 \tag{16}$$

$$\frac{dv_1}{dt} = \frac{Gm_2}{r_{12}^3}(y_2 - y_1) \tag{17}$$

$$\frac{dx_2}{dt} = u_2 \tag{18}$$

$$\frac{du_2}{dt} = \frac{Gm_1}{r_{12}^3}(x_1 - x_2) \tag{19}$$

$$\frac{dy_2}{dt} = v_2 \tag{20}$$

$$\frac{dv_2}{dt} = \frac{Gm_1}{r_{12}^3}(y_1 - y_2) \tag{21}$$

We can solve this system of now 8 coupled first order differential equations in the same way we did for all of the previous examples. Your function file (twobody.m) should look something like:

```
function dz = twobody(t,z)
dz = zeros(8,1);

G = 2;
m1 = 2;
m2 = 2;

dz(1) = z(2);
dz(2) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(5) - z(1));
dz(3) = z(4);
dz(4) = ((G*m2)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(7) - z(3));
dz(5) = z(6);
dz(6) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(1) - z(5));
dz(7) = z(8);
dz(8) = ((G*m1)/(((z(1) - z(5)).^2 + (z(3) - z(7)).^2).^(3/2)))*(z(3) - z(7));
```

where we define $z(1)$, $z(2)$, ... through $z(8)$ to represent the functions $x_1(t)$, $u_1(t)$, $y_1(t)$, $v_1(t)$, $x_2(t)$, $u_2(t)$, $y_2(t)$, and $v_2(t)$, respectively. (So that $r_{12}^2 = (z(1) - z(5))^2 + (z(3) - z(7))^2$).

Your script file should look something like:

```
[t,z] = ode45('twobody',[0 25], [-1 0 0 -1 1 0 0 1]);
[t,z] = ode45('twobody',[0 25], [-1 0 0 -1 1 0 0 1]);


plot(z(:,1),z(:,3),'-');
xlabel('x_{1}(t)');
ylabel('y_{1}(t)');
title('Particle 1 orbit in xy space -- first 25 seconds');

figure;
plot(z(:,5),z(:,7),'-');
xlabel('x_{2}(t)');
ylabel('y_{2}(t)');
title('Particle 2 orbit in xy space -- first 25 seconds');
```

for an integration time from $t = 0$ until $t = 25$, and some set of initial conditions. As a note of caution, it is very easy to make a mistake while typing up these equations into your function file, so be very careful when you do!

You can make all kinds of fun plots and movies with data like these! For example, you can make a movie out of a group of plots, where each plot shows you the location of one of the particles at a particular time; do this for multiple time steps, string them together, and you've got yourself a homemade movie.

You create movies in Matlab as well, but that's going to be a topic for another day.

Acknowledgments

Many of these examples come from a few courses I took at MIT – namely, 18.353 and 8.301. Thanks to the website www.math.tamu.edu/ mpilant/math614/twobody.pdf for details on setting up the two-body Kepler problem. Thanks to my kitties for staying up with me and keeping me company by being adorable while writing this. Also, thanks also to MIT's Junior Lab (8.13/8.14) for providing the LaTeXtemplate for this paper.