Nur Adila Faruk Senan
Department of Mechanical Engineering
University of California at Berkeley

A brief introduction to using ode45 in MATLAB

MATLAB's standard solver for ordinary differential equations (ODEs) is the function ode45. This function implements a Runge-Kutta method with a variable time step for efficient computation. ode45 is designed to handle the following general problem:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{1}$$

where t is the independent variable, $\mathbf{x}$ is a vector of dependent variables to be found and $\mathbf{f}(t, \mathbf{x})$ is a function of t and $\mathbf{x}$. The mathematical problem is specified when the vector of functions on the right-hand side of Eq. (1), $\mathbf{f}(t, \mathbf{x})$, is set and the initial conditions, $\mathbf{x} = \mathbf{x}_0$ at time t_0 are given.

In ME175, the solution is often not complete once you have solved the problem and obtained the ode's governing the systems motion. It is often beneficial to produce a visual representation of what exactly the trajectories of a particle represented by a highly complicated looking ordinary differential equation looks like and the following is a brief explanation of how to accomplish this.

Step 1: Reduce the given ode to a series of first order equations

This is the very first step you should do, ideally on a separate piece of paper. For example, if the given ode is,

$$m\ddot{y} + \dot{y}e^y - y^2 = 5, \qquad y_0 = 3, \dot{y}_0 = -1, \tag{2}$$

then the vector $\mathbf{x}$ has two components: y and $\dot{y}$, or,

$$\begin{aligned} x(1) &= y \\ x(2) &= \dot{y}, \end{aligned} \tag{3}$$

and

$$\begin{aligned} \frac{d}{dt}x(1) &= x(2) \\ \frac{d}{dt}x(2) &= \frac{1}{m}\left(5 - x(2)e^{x(1)} + (x(1))^2\right), \end{aligned} \tag{4}$$

where we have made use of the representations for $y, \dot{y}$ and $\ddot{y}$ given in Eqn (3) to write (4).

What if we have more than one ode? For example, suppose that in addition to Eq. (2) above, we also have a second equation,

$$\frac{d^3z}{dt^3} + \frac{d^2z}{dt^2} - \sin(z) = t, \qquad z_0 = 0, \dot{z}_0 = 0, \ddot{z}_0 = 1. \tag{5}$$

Then, we can easily solve for both y and z simultaneously by making $\mathbf{x}$ a larger vector:

$$\begin{aligned} x(3) &= z \\ x(4) &= \frac{dz}{dt} \\ x(5) &= \frac{d^2z}{dt^2} \end{aligned} \tag{6}$$

Thus,

$$\mathbf{x} = [y, \dot{y}, z, \dot{z}, \ddot{z}]\,, \tag{7}$$

and

$$\mathbf{x}\,|_{t=0} = \left[y\,|_{t=0}\,, \frac{dy}{dt}\,|_{t=0}\,, z\,|_{t=0}\,, \frac{dz}{dt}\,|_{t=0}\,, \frac{d^2z}{dt^2}\,|_{t=0} \right]. \tag{8}$$

There is no preference for placing the y-related variables as the first two variables instead of the z-related variables. In fact, any arbitrary order is perfectly valid, such as,

$$\mathbf{x} = [y, z, \dot{y}, \dot{z}, \ddot{z}] \quad \text{and} \quad \mathbf{x} = [z, \dot{z}, \ddot{z}, y, \dot{y}]\,. \tag{9}$$

What is important however is that you remain consistent throughout your computations in terms of how you write out your system of first order ode's.[1] Thus, if you use the ordering given in Eq. (7), then the system of first order ode's are comprised of,

$$\begin{aligned}
\frac{d}{dt}x(1) &= x(2) \\
\frac{d}{dt}x(2) &= \frac{1}{m}\left(5 - x(2)e^{x(1)} + (x(1))^2\right), \\
\frac{d}{dt}x(3) &= x(4) \\
\frac{d}{dt}x(4) &= x(5) \\
\frac{d}{dt}x(5) &= t - x(5) + \sin{(x(3))},
\end{aligned} \tag{10}$$

while employing the representation $\mathbf{x} = [y, z, \dot{y}, \dot{z}, \ddot{z}]$ results in,

$$\begin{aligned}
\frac{d}{dt}x(1) &= x(3) \\
\frac{d}{dt}x(2) &= x(4) \\
\frac{d}{dt}x(3) &= \frac{1}{m}\left(5 - x(2)e^{x(2)} + (x(2))^2\right), \\
\frac{d}{dt}x(4) &= x(5) \\
\frac{d}{dt}x(5) &= t - x(5) + \sin{(x(2))}.
\end{aligned} \tag{11}$$

Basically, you could have an arbitrary number of higher order ode's. What is important is that you reduce them to multiple first order ode's and keep track of what order you've assigned the different derivatives in the final $\mathbf{x}$ vector that will be solved.

Step 2: Coding it in

Now that you have everything in first order form, you will need the following commands in your main code:

```
[t,x] = ode45(@fname, tspan, xinit, options)
```

- `fname` is the name of the function Mfile used to evaluate the right-hand-side function in Eq. (1). This is the function where we will input the system of first order ode's to be integrated (such as in Eqs. (10) and (11)). I will explain this in a little more detail later on.

[1]Of course, there might be some subtleties with regards to how `ode45` numerically integrates the given equation and in some cases, it might make sense to solve some equations before others but for the simple problems we will be dealing with, this is a non-issue.

- `tspan` is the vector defining the beginning and end limits of integration, as well as how large we want our time steps to be. I.e. if we are integrating from $t = 0$ to $t = 10$, and want to take 100 time steps, then `tspan=` [0:0.1:10] or `tspan=linspace(0,10,100)`).

- `xinit` is the vector of initial conditions. Make sure that the order corresponds to the ordering used to write y, z and their derivatives in terms of `x`. Also note that if `x` consists of 5 variables, then we need an input of 5 initial conditions (see Eqn. (8)).

- `options` is something that is very well explained in the help session of MATLAB. For most purposes, using the default value is sufficient.

- `t` is the value of the independent variable at which the solution array `x` is calculated. This vector is not necessarily equal to `tspan` above because ode45 does some amount of playing about with step sizes to maximize both accuracy and efficiency (taking smaller steps where the problem changes rapidly and larger steps where it's relatively constant). The length of `t` however is the same as that of `tspan`

- `x` is what this is all about.[2] `x` is an array (or matrix) with size `length(t)` by `length(xinit)`. Each column of `x` is a different dependent variable. For example, if we have `x` $= [y, \dot{y}, z, \dot{z}, \ddot{z}]$, and assume (for simplicity) that we require only the values at $t = 0, 1, 2, 3, ..., 10$ (i.e. we evaluate the function at 11 different times) then

$$\mathtt{x} = \begin{bmatrix} y\,|_{t=0} & \dot{y}\,|_{t=0} & z\,|_{t=0} & \dot{z}\,|_{t=0} & \ddot{z}\,|_{t=0} \\ y\,|_{t=1} & \dot{y}\,|_{t=1} & z\,|_{t=1} & \dot{z}\,|_{t=1} & \ddot{z}\,|_{t=1} \\ & & \cdots & & \\ & & \cdots & & \\ & & \cdots & & \\ y\,|_{t=10} & \dot{y}\,|_{t=10} & z\,|_{t=10} & \dot{z}\,|_{t=10} & \ddot{z}\,|_{t=10} \end{bmatrix}. \tag{12}$$

Thus, `x`(1,4) gives us the value of $\dot{z}$ at $t = 0$, `x`(7,4) gives us the value of $\dot{z}$ at $t = 6$, and `x`(11,4) gives us the value of $\dot{z}$ at $t = 10$ since $\dot{z}$ is the fourth variable in `x`. The same holds for all the other variables. In short,

- `x`(:,k) gives us the column vector of the k'th variable: $k = 1$ would correspond to y, $k = 2$ to $\dot{y}$, etc.

- `x`(j,:) gives us the values of all the computed variables at a single instant of time corresponding to index j.

[2]Before the invention of the Hokey Pokey dance, ancient children attending ancient campfire events would sit around ancient campfires singing:
You put your left foot in
You put your left foot out
You put your left foot in
And you shake it all about
You use the ode45 MATLAB *function*
to get your homeworks done on time
x is what it's all about!
Unfortunately, the lack of computers (and MATLAB) soon made this version obsolete.

$\Rightarrow$ **x(j,k) $\equiv$ x(time, variable of interest).**

The line following `[t,x] = ode45(@fname, tspan, xinit, options)` should be a redefinition of your variables. Thus, if you used x $= [y, \dot{y}, z, \dot{z}, \ddot{z}]$, then you would write:

```
[t,x] = ode45(fname, tspan, xinit, options)
y=x(:,1);
ydot=x(:,2);
z=x(:,3);
zdot=x(:,4);
zdotdot=x(:,5);
```

Of course, you could just as well not bother to define y, ydot, etc. and instead deal directly with the indicial form of x (e.g. using x(:,1) whenever we mean y, etc) but defining the variables clearly does make things a little easier to debug at 3 in the morning the day the homework is due.

Below this, you usually plot what exactly it is you're interested in showing (or have been asked to show): The trajectory of a particle as a function of time? The relationship between r and θ for a planar pendulum? etc. The plot and subplot commands in MATLAB are lucidly explained in the MATLAB help and I won't go into detail about them here. Bear in mind that if you plan to hand in 20 plots, you will do the grader (and mother nature) a favor by using the subplot function to fit multiple plots into one page. Don't go overboard with this however - 20 plots on a single page isn't a good idea either. Additionally, don't forget to label your graphs adequately: Each plot should have a title, labels for the x-axis, y-axis and a legend if there are multiple curves on the same plot. *A plot without attached lables is just a bunch of lines!*

Finally, note that everything in `[t,x] = ode45(@fname, tspan, xinit, options)` are just variables. You can use whatever terms you wish - T instead of t, x0 instead of xinit, or OUT instead of x to cite a few examples. Just remember that if you have defined a variable, then you have to always refer to it by the same name.
`[TIME,OUT] = ode45(@...)` followed by y = x(:,1); will give you an error since x is undefined. (The correct equation should be y= OUT(:,1);)

Another common mistake is to redefine the same variable. For example, doing something like x = x(:,5);. If you're lucky, you will get an error message. If you're not however (which usually happens if the error is not as glaring), then you could end up spending hours trying to figure out why your solution looks a little funny.

Aside: What is 'fname?

Recall that we still haven't told MATLAB what exactly the equations of motion are that need to be integrated. This is where fname comes in. fname is the name of the function containing all the first order ode's we wrote right at the beginning. We **can name this function anything we like** so long as the name you give it is the same as what you use when calling it in `[t,x] = ode45(@fname, tspan, xinit, options)`. Thus, if you call your function superman, then
`[t,x] = ode45(@superman, tspan, xinit, options)` is right but
`[t,x] = ode45(@batman, tspan, xinit, options)` is wrong (if you have a function

named batman in your library, then you'll end up getting the solution to that problem instead!).

Furthermore,, the function *doesn't have to be in the same mfile* as your original code - some people prefer to write it as a sub-function right at the end of the program, especially if the code isn't too large or complicated. For example, say your function is called ME175example. Then, your mfile would look like this:

```
function dxdt = ME175example(t,x)
%Here, t,x and dxdt are again, just variables. You can call them whatever you want,
%so long as t is the independent variable, x is the vector of dependent variables and
%dxdt is the vector of first order derivatives (with respect to the independent variable)


% Define the constants:
m = 1;


% Define the variables to make things little more legible
% Recall that x = [y, ydot, z, zdot, zdotdot]
y=x(1);
ydot=x(2);
z=x(3);
zdot=x(4);
zdotdot=x(5);
%Notice that here, x is just a 1 by 5 row vector and not a large array like in
%[t,x] = ode45(@fname, tspan, xinit, options)


% the array dxdt is the same length as x
dxdt = zeros(size(x));
dxdt(1) = ydot;
dxdt(2) = 1/m(5 − x(2)exp(y) + y^2);    %This is ydotdot
dxdt(3) = zdot;
dxdt(4) = zdotdot;
dxdt(5) = t-zdotdot+sin(z);    %This is zdotdotdot


%Note that the input arguments must be t and x (in that order) even in the case where t is not
explicitly used in the function.
```

A basic template

The following is a basic template you can just cut and paste into MATLAB every time you would like to integrate a higher order ODE:

```
function callthiswhateveryouwant
%define tstart, tend and the number of time steps you want to take, n:
tstart = ?;
tend = ?;
n = ?;
tspan = linspace(tstart,tend,n);


%define the initial conditions making sure to use the right ordering
xinit = [...;...;...; ...];
```

%Get x. I have left options as the default value so it's not included in the function call below
[t,x] = ode45(@integratingfunction, tspan, xinit)
%define the output variables:
y = x(:,1);
ydot = x(:,2);
etc...

%Include whatever plotting functions you need to include here. You can plot anything you want: %y(t), $\dot{y}$(t), $\dot{y}$(y), $\dot{y}$($\ddot{y}$), etc.
subplot(?,?,?)
plot(whatever you are plotting)
%plot3(...) % use this is you are plotting a 3D plot title(' ')
xlabel(' ')
ylabel(' ')
%zlabel(' ') % uncomment the last line if you are doing a 3D plot
%——————————————-
%THE INTEGRATION FUNCTION
% This can either be in a separate mfile or right at the bottom of the same mfile.
%Either way, the syntax is as follows:

function dxdt = integratingfunction(t,x)

%1. Define your constants - mass, gravity, etc.

%2. Define new variables: y=x(1), ydot=x(2), etc
%You don't have to do this, but is makes it a lot easier for others to decipher your code

%3. Write out your first order system of ODE's:
dxdt = zeros(size(x));
dxdt(1) = ?;
dxdt(2) = ?;
etc.

That's it!

Assuming you did everything correctly and your equations are correct, you should be able to get a glimpse of the amazing beauty behind the equations you've slogged so hard to derive. Nice job.

Finally, note that there are many different ways of coding the same thing. Play about with MATLAB and you will probably discover multiple ways of doing the same thing, some more efficient or easier than others. I have been using MATLAB for three years and I'm still discovering new and more effective ways of doing things. Above all, have fun. Despite what you think, MATLAB is not out to get you (it has it's hands full giving me a hard time).