

# NUMERICAL SOLUTIONS TO ODEs USING MATLAB

Suppose you have a system, linear or nonlinear, you need to find out its behavior, and either you don't know how to solve it analytically, or you don't want to bother. If you can write the equations in the form  $\dot{\bar{x}} = \bar{f}(\bar{x}, t)$ , then you can get MATLAB to solve them for you, using the MATLAB functions `ode45()` or `nareul()`.

## **ode45 vs. nareul**

`ode45()` and `nareul()` are two MATLAB functions that do basically the same thing. `ode45()` is a sophisticated built-in MATLAB function that gives very accurate solutions. `nareul()` is a function written for this class. It does not give as accurate solutions, but it has two advantages:

1. It does the solution step-by-step, rather than automatically, showing you exactly what is going on in the solution process.
2. It does a bunch of automatic error checking, so it is likely (but not guaranteed) to give you a helpful error message if you use it wrong.

I recommend starting with `nareul()` to get the advantage of its explanations and error checking, and then moving to `ode45()` when you get bored with the step-by-step operation of `nareul()`. However, any time you run into a problem using `ode45()`, you can switch back to `nareul()` and use it to help you find your bug. Then, once everything is running, just substitute `ode45()` for `nareul()` to get a faster more accurate solution.

`nareul()` can be downloaded from the course web site ([www.dartmouth.edu/~sullivan/engs22.html](http://www.dartmouth.edu/~sullivan/engs22.html)), or, on the thayer AFS system, it can be found in the `engs022` public directory `\afs\thayer\home\course\engs022\public`

`nareul()` is supposed to give a helpful error message for anything you might do wrong with it. If you find a way to misuse it with which you do not get a helpful error message, please let me know.

## **Writing a system as $\dot{\bar{x}} = \bar{f}(\bar{x}, t)$**

Actually, if you've got the equation in any sort of state-space form, you've already done this. For example, suppose you have  $\dot{\bar{x}} = \mathbf{A}\bar{x} + \mathbf{B}\cos(\omega t)$ . Well, all that stuff on the right side depends only on  $\bar{x}$  and  $t$  (once you've fixed the constants in  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\omega$ ). So it's a function of  $\bar{x}$  and  $t$ .

## **Telling MATLAB about your function**

MATLAB (or more specifically, `ode45` or `nareul()`) is going to need to calculate this function,  $\bar{f}(\bar{x}, t)$ , in order to calculate the derivative and figure out what happens in the system. The way you instruct it to do this is to write a function, as an m-file, and tell `ode45` or `nareul` the name of that function. It can then call your function whenever it needs it. The function might be very simple, but you've still got to make it into an official function in an m-file. For example, if we had a first order linear system with no input,  $\dot{x} = -\frac{1}{4}x$ , the function could be simply:

```
-----in the file derivem-----
function xdot = deriv(t,x)
    xdot = -0.25*x;
-----
```

For the system  $\dot{\bar{x}} = \mathbf{A}\bar{x} + \mathbf{B}\cos(\omega t)$ , it could be:

```
-----in the file ABsys.m-----
function xd = ABsys(t,x)
    A = [0 1 ; -5 -2];
    B = [1 ; 0]; w = 4;
    xd = A*x + B*cos(w*t);
-----
```

If your system is more than first order, the state,  $x$ , will be a vector. The function must be set up to work with it as a *column* vector. What if the way you have written the equations does work with the state as a vector at all? Just separate out the components at the start of the function, and pack them back in at the end, as shown below.

```
-----in the file another.m-----
function xd = another(t,x)
y = x(1);
w = x(2);
wdot = -10 * y^5;
ydot = 6 * w - 0.6*y;
xd = [ydot ; wdot];
-----
```

Now that you've got your function defined, and stored in a file, `ABsys.m`, or `deriv.m`, you need to tell `ode45` or `nareul` the name of the function, so it can call it. You do this by passing the name of your function to `ode45` or `nareul` as a string, which is one of the arguments of the function, as explained in the next section.

### Getting a Solution

You can now get your solution (to, for example, `ABsys`), by calling `nareul` with the following command line:

```
nareul('ABsys',[t0,tfinal],x0)
```

(make sure you have downloaded or copied `nareul` first!) The three arguments are a string naming your function, a vector giving the time span, and the vector giving the initial state at the starting time. The time span vector is simply two elements—the initial and final times. You can set up variables such as `t0`, `tfinal`, and `x0` before you call the function, or type numbers directly in the command line.

You can then follow the instructions on the screen. One tricky thing to be aware of is that use must click in the command window to type something, or at the graphics window to click on the graph, respectively. The command above will just plot the results as you go. If you also want the results in an array, so you can plot them differently, or look at the numerical values, or whatever, you can use

```
[t,x] = nareul('ABsys',[t0,tfinal],x0);
```

You will get back a vector `t` and a matrix `x`, which gives the values of the state at each time in the vector `t`.

### ode45 differences

You call `ode45` exactly the same way, and it generates a solution, just more quietly and accurately. If you call it with

```
[t,x] = ode45('ABsys',[t0,tfinal],x0);
```

it won't even plot anything, but will just stick the results in the vector `t` and a matrix `x`, which gives the values of the state at each time in the vector `t`. You can now, for example plot the response by typing

```
plot(t,x)
```

or

```
plot(t,x(:,n))
```

to plot the  $n$ th state. Or you can analyze the data, for example with `max(max(x))` to find the largest value of any state at any time. You can also get it to plot the results as it goes, which is particularly nice if you are doing something that takes a while to solve—with a plot proceeding you know that it is working and can see how far it has gotten. If you call it without output arguments:

```
ode45('ABsys',[t0,tfinal],x0);
```

it figures you must want a plot since you aren't saving the data, and it plots as it goes. For another way to turn on plotting, use the optional options argument as described in the next section.

## ode45 options

There are a multitude of options available for `ode45`, none of which work with `nareul`. You set options with the function `odeset`, as follows:

```
options = odeset('name1',value1,'name2',value2,...)
[t,x] = ode45('ABsys',[t0,tfinal],x0,options)
```

This would set option 'name1' to value1, 'name2' to value2, etc. The variable `options` is a "structure"—an object that contains all the information about the options you have set. Useful options include:

- 'OutputFcn' If you set this to 'odeplot', it will automatically plot as it goes along, whether or not you have output arguments.
- 'Refine' Set this to an integer to increase the time resolution of the solution. This defaults to 4 in `ode45`, so to increase resolution, you'd need a higher value (8, 12, etc.).
- 'Maxstep' This is another way to increase the time resolution of the solution.
- 'RelTol' The accuracy of the solution. This defaults to 0.1% ( $1e-3$ ). Good enough for most work, but sometimes you'll want it tighter.
- 'AbsTol' This defaults to  $10^{-6}$ . When the solution is that small, MATLAB assumes you don't really need 0.1% accuracy anymore. (Otherwise, when the solution went through zero, it would need to compute it with infinite precision!) But be careful about this one—if you were solving for the behavior of a micromachine, and you did everything in meters, MATLAB would assume you didn't care about the  $\mu\text{m}$  scale motions of your machine at all! (To solve the problem, you could set this to  $10^{-12}$ , or do the problem in units of  $\mu\text{m}$ .)

## Hazards

Although it's better than the "solvers" built into a lot of other software, sometimes `ode45` can give you misleading results, and sometimes it can fail completely.

**Failures**— If it fails completely, or seems to be having a lot of trouble with your problem, first go back to using `nareul` to see what's going on. There is also another solver built into MATLAB, `ode15s`, that you can try, using exactly the same syntax. Actually, there are a whole slew of them, but with `ode45` and `ode15s` you should be able to solve just about anything. For this class you are unlikely to need anything but `ode45`.

**Big Steps**— The problem that can occasionally give misleading results is that the solver is trying to take as big steps as it can, to get done sooner. Even if it can accurately step that far, the result may not look like a smooth curve when it's plotted. With the default setting of the "refine" option (four), you rarely run into this, but it's good to be aware of the possibility, and to look at your plot critically. Does it look like a series of line segments rather than a curve? Could there be important peaks in between the points? (The automatic plot function `odeplot` plots the points as circles so you can judge this.) If it looks like there may be trouble, you can increase the value of the `Refine` option. (You can also decrease the value of `MaxStep`, or specify the actual times where you want the solution calculated by providing a full vector of times at which you want solutions in place just the initial and final times, e.g.,

```
[t,x] = ode45('ABsys',linspace(t0,tfinal,300),x0).);
```

## For more information

For full details on the options above and more, read the help files on `ode45`, `odeset` and `odefile`.