Parameters in Differential Equations

Leigh J. Little SUNY Brockport November 2, 2018

Contents

1 Using Parameters with `ode45`

Consider the initial value problem below

$$y'(t) = py, \quad y(a) = Y, \quad t \in [a, b].$$

As we saw before the independent variable in this equation is t and the dependent variable is $y(t)$. However, this equation contains a third variable, p, that is neither the independent or dependent variable. In this example, p is called a parameter. A parameter provides a way to describe an entire family of similar problems. Parameters show up in a number of differential equations.

Suppose we wanted to have MATLAB solve the initial value problem

$$y'(t) = py, \quad y(0) = 100, \quad t \in [0, 5].$$

for several different values of p, say $p = -1, -2, -3$. To run the case when $p = -1$. we could set up the right-hand side function as

```
function yp = odeexample(t,y)
p = -1;
yp = p*y;
```

followed by a driver script that looks like

```
clear
close all

T = [0 5]
Y = 100;
[t,y] = ode45(@odeexample,T,Y);
```

However, to run the other two values of p, we would need to either change the right-hand side function or create a new one that has a different name, but the correct value of p.

This process would work, but it would be prone to a variety of errors; for example, saving the right-hand side function under the wrong name, having two right-hand side functions with the same value of p but different names, *etc.*.

A better way to handle this would be to write a generic right-hand side function that allowed the parameter to be passed as a variable to the function. Fortunately MATLAB allows us to do this with a slight change in how we write and call the function.

Enter the following into a file called `odeexample.m` and save it

```
function yp = odeexample(t,y,p)
yp = p*y;
```

Then enter the following into some driver script

```
clear
close all

T = [0 5]
Y = 100;
p = -1;
[t,y] = ode45(@(t,y)odeexample(t,y,p),T,Y);
```

Notice that the calling sequence to `ode45` has changed somewhat. In addition to the name of the right-hand side function, we also have to state the entire function prototype as

```
@(t,y)odeexample(t,y,a)
```

This is necessary because the inputs to `odeexample` are no longer just the independent and dependent variables `t` and `y`. We also have the parameter `p` as an input to the right-hand side function. This necessitates the list of input arguments after the function name. However, we also need to explicitly tell MATLAB which of these inputs are the independent and dependent variables for the differential equation itself. This is accomplished by the `@(t,y)` in front of the right-hand side function name.

This mechanism can be extended to allow for any number of parameters to be used in solving a differential equation using `ode45`. For example, if we had the initial value problem.

$$y' = \sin(py) + \cos(qt) + rty, \quad y(0) = 100, \quad t \in [0, 10]$$

and wanted to solve this for $p = 2, q = 3, r = -1$. we could do the following

```
function yp = example2(t,y,p,q,r)
yp = sin(p*y) + cos(q*t) + r*t.*y;
```

with the driver script

```
clear
close all

T = [0 10]
Y = 100;
p = 2;
q = 3;
r = -1;
[t,y] = ode45(@(t,y)example2(t,y,p,q,r),T,Y);
```

2 Cell Arrays

We have made extensive use of array variables in MATLAB. We have worked mainly with vectors, but working with matrices is not much different. For example, if we wanted to define the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix},$$

we have see there are many ways to create this matrix; for example

```
>>  A = [1 2 3; 4 5 6; 7 8 9];
```

What has always been true is that the element that lives at a particular location in a vector or matrx has been a scalar. For example, $A_{2,1} = 4, A_{3,2} = 8$ and so on.

In cases where large amounts of data need to be efficiently managed, it would be useful if the entity stored in a particular element of a vector or a matrix could be something other than a scalar. Fortunately MATLAB has a special type of array called a *cell array* that permits us to do just that.

Enter the following commands (note the use of curly braces instead of parentheses).

```
>> B{1,1} = 1;
>> B{1,2} = ones(10,1);
>> B{2,1} = 'abcdefg';
>> B{2,2} = [1 2 3; 4 5 6; 7 8 9];
>> B
B =
  2x2 cell array
    {[      1]}    {10x1 double}
    {'abcdefg'}    { 3x3 double}
```

In this example, the array B is a 2×2 cell array. It has two rows and two columns like the matrices we have been working with, but now the entities that live in a particular location are not limited to simple scalars. For example, a scalar lives in $B\{1,1\}$ but a 3×3 matrix lives at $B\{2,2\}$.

```
>> B{1,1}
ans =
     1
>> B{2,2}
ans =
     1     2     3
     4     5     6
     7     8     9
```

In our case, the rational for needing the cell array structure arises from the need to solve a parametrized initial value problem for several values of the parameter. For example, go back to the problem we had from before. Solve

$$y'(t) = py, \quad y(0) = 100, \quad t \in [0, 5].$$

for $p = -1, -2, -3$. Each value of p will generate an array of `t` coordinates and array of `y` coordinates. It would make data management easier if we could store all of the `t` and `y` coordinates column by column matrices called `TC` and `YC`. In this case the driver might look like the following:

```
clear
close all

T = [0 5];
Y = 100;
P = [-1 -2 -3];
for i = 1:length(P)
     [TC(:,i),YC(:,i)] = ode45(@(t,y)odeexample(t,y,P(i)),T,Y);
end
```

The driver loops over the values in the P array. Each time a new set of solution coordinates are obtained, these are stored in the `TC` and `YC` arrays.

Unfortunately, there is a subtle problem with this approach. Consider the simple example below

```
>> t1 = [1 2 3]';
>> t2 = [1 2]';
>> TC = [t1 t2]
Error using horzcat
Dimensions of arrays being concatenated are not consistent.
```

The attempt to place the column vectors `t1` and `t2` in the TC matrix fails because these vectors don't have the same length.

The `ode45` solver automatically determines how many steps to take from $t = a$ to $t = b$ in order to obtain approximate solutions that meet its convergence tolerances. What this means is that the lengths of the `t` and `y` coordinates may be different for different values of p. This means that the statement

```
[TC(:,i),YC(:,i)] = ode45(@(t,y)odeexample(t,y,P(i)),T,Y);
```

will fail eventually because the new column being added at loop index `i` will not have the same length as the previous columns.

The answer to this dilemma is cell arrays. Because each element of a cell array can be anything, we can store the `t` and `y` coordinates in a large cell array. The fact that one set of coordinates might have a different length than another set of coordinates is no longer a problem. The new driver would look like the following:

```
clear
close all

T = [0 5];
Y = 100;
P = [-1 -2 -3];
for i = 1:length(P)
     [TC{i},YC{i}] = ode45(@(t,y)odeexample(t,y,P(i)),T,Y);
end
```

Some notes on the script above:

- Note the use of curly braces {,} instead of parentheses when assigning values to `TC` and `YC`.
- Note that `TC` and `YC` only require a single subscript (*i.e.,* the are cell array vectors). Each element `TC{i}` of `TC` contains the entire vector of `t` coordinates corresponding to `P(i)`.