LECTURE 9: DIFFERENTIAL EQUATIONS, VECTORS & LISTS

3.1 Solving ordinary differential equations using `dsolve`

The "`dsolve`" command can be used to find the closed form solutions of many types of differential equations. For example, to solve the homogeneous differential equation

$$\frac{d^2y}{dx^2} - y = 0$$

we use the `symfun` class as follows:

```
>> syms y(x)
>> ode = diff(y,x,2) - y == 0
>> dsolve(ode)
```

Notice the way by which the symbolic toolbox gives the constants of integration. The important thing here is that `y(x)` was created as a `symfun` and that the `diff(y,x)` was used to create the ODE (and not `diff(y(x),x)`); this is an example of the toolbox differentiating arbitrary functions, which was discussed previously in Section 2.3.1.

Inhomogeneous equations are no different; to solve

$$\frac{dy}{dx} + \frac{y}{x} = x^2$$

type

```
>> clear
>> syms y(x)
>> ode = diff(y,x) + y/x == x^2
>> dsolve(ode)
```

Sometimes MATLAB does not automatically simplify the solution as much as it could—the solution of

$$\frac{d^3x}{dt^3} - 3\frac{d^2x}{dt^2} + 3\frac{dx}{dt} - x = 16e^{3t}$$

is found by typing

```
>> syms x(t)
>> ode = diff(x,t,3) - 3*diff(x,t,2) + 3*diff(x,t) - x == 16*exp(3*t)
>> soln = dsolve(ode)
>> simplify(soln)
```

To insert initial and/or boundary conditions in MATLAB, proceed as in the following examples. The solution of the boundary value problem

$$y'' + 5y' + 6y = 0, \quad y(0) = 0, \quad y(1) = 3$$

is found as follows:

```
>> syms y(x)
>> DE = diff(y,x,2) + 5*diff(y,x) + 6*y == 0
>> soln = dsolve(DE, y(0)==0, y(1)==3)
```

We could then plot the solution:

```
>> ezplot(soln(x), [0 1])
>> ylim([0 6])
>> grid on
>> ylabel('y')
```

Now consider the initial value problem

$$y'' = y, \quad y(0) = 1, \quad y'(0) = 0,$$

which could be solved as follows

```
>> syms y(t)
>> DE = diff(y,t,2) == y;
>> yp = diff(y,t);
>> Y = dsolve(DE, y(0)==1, yp(0)==0)
```

And again a plot:

```
>> Yp = diff(Y, t);
>> clf
>> ezplot(Y, [0 2])
>> hold on
>> h = ezplot(Yp, [0 2])
>> set(h, 'color', 'red', 'linestyle', '-.')
>> grid on
>> % note how we enter y'(x) here:
>> legend('y(x)', 'y''(x)')
```

Exercise 3.1 Use MATLAB to solve the following differential equations:

$$\text{(i)} \quad \frac{d^2y}{dx^2} + 3\frac{dy}{dx} + 4y = \sin x, \qquad \text{(ii)} \quad \frac{dy}{dx} = 3xy, \; y(0) = 1.$$

In (ii), plot the solution.

Exercise 3.2 Solve the differential equation $y'' + y = 0$, subject to the initial conditions $y(0) = a,\ y'(0) = b$. Assign the result to a variable and then evaluate the solution at $x = \pi$. □

3.2 Vectors and lists in Matlab

MATLAB was originally designed to make it easy to manipulate matrices and vectors. A row vector takes the form `[expr1, expr2, expr3,..., exprn]`, in other words it is a sequence of expressions enclosed in square brackets. We will often choose to think of this as a *list*. In a list, order and repetition are both important.

An example of entering a list is:

```
>> L = [1, 2, 3, 4]
```

In fact the commas are optional:

```
>> L = [10 12 11 10 9]
```

Specific elements can be extracted. Here we pick out the third element:

```
>> L(3)
```

In a similar fashion, you can replace entries of a list:

```
>> L(3) = 42
>> L
```

There is a useful shortcut in list construction of the form "`start:step:end`", for example:

```
>> L1 = 1:10
>> L2 = 3:2:13
>> L3 = 17:-1:10
>> L4 = 1.0:0.2:1.6
```

Lists can be concatenated

```
>> clear
>> L1 = [1 2 3];
>> L2 = [4 5 6 7];
>> L3 = [L1 L2]
```

Note this makes a longer list rather than making a "list of lists" (the latter can be done in MATLAB using cell arrays: see Appendix B.3). The number of elements in a list is its length:

```
>> n = length(L3)
```

You can make an empty list of length zero:

```
>> L = []
>> n = length(L)
```

3.2.1 Symbolic lists

We can also make lists/vectors of symbolic objects and manipulate their elements

```
>> syms x
>> L = [sin(x) cos(x) tan(x)]
>> L(1)
>> diff(L(1), x)
>> % in fact we can differentiate all elements at once:
>> derivs = diff(L, x)
```

Every element of a list must be the same type. If you mix `sym` and `double` in a list construction, everything will be cast to `sym`:

```
>> clear
>> x = sym(10);  y = 11;  z = 12;
>> L = [x y z];
>> a = L(1);  b = L(2);  c = L(3);
>> whos
```

Notice that `L` is a `1x3 sym` and the `a`, `b`, and `c` are all of class `sym`.

Exercise 3.3 Design an experiment to check what happens if you replace one element of an existing `sym` list with a `double`. On the other hand, what happens if you replace one element of an existing `double` list with a `sym`? □

3.2.2 Indexing in lists

Ranges of elements can be extracted using the colon notation:

```
>> L = 2:0.1:4
>> L(1:3)
>> L(2:2:12)
```

There is also a keyword called "`end`"

```
>> L(1:end-4)
>> L(end-3:end)
```

You can use one list as indices to access another list:

```
>> I = [1 5 3 2]
>> L(I)
>> % or directly
>> L([1 5 3 2])
```

This leads to a common design pattern in MATLAB:

```
>> I = find(L > 3.05)
>> L(I)
>> I = find( (L >= 3.1) & (L < 3.8) )
>> L(I)
```

Here `I` is a list of indices satisfying the condition. You can combine the "list accessing a list" pattern with the "`end`" keyword. For example, the following will "rotate" a list:

```
>> rotLeft = L([2:end-1  1])
>> rotRight = L([end  2:end-1])
```

(This is a bit of a trick—so called "syntactic sugar"—e.g., you cannot store `I = [2:end-1 1]`.)

3.2.3 Vector operations

To do component-wise operators on a vector, we need special "dot" operators. To raise each element to a power, use:

```
>> L = [1 2 4 8];
>> L3 = L .^ 3
```

Similarly, to multiple or divide the components of one vector by the components of a another vector (component-wise multiplication/division), do:

```
>> M = L .* L3
>> Z = (L.^4) ./ L - L3
```

The component-wise multiplication symbol is typically pronounced "dot star". The usual multiplication symbol * (without the dot) means matrix multiplication in this context (see Section 3.5).

Vectors and matrices are so fundamental in MATLAB that we return to them in Sections 3.5 and 3.6 but first we look at some applications of lists, namely for set theory and performing sums.

3.3 Sets

MATLAB doesn't have particular support for mathematical sets. (MuPAD does using its worksheet mode, see Appendix B.4). However, using lists, we can implement some basic ideas of set theory.

The set $\{1,2,3\}$ could be presented by

```
>> S = [1 2 3]
```

Similarly the set of three polynomial expressions could be

```
>> syms x
>> S = [x^2 3*x 2]
```

Order should not be important and repetitions should be ignored in sets:

```
>> S = [1 3 4 1 2 1]  % not really a set
>> S = unique(S)      % much better
```

Given two sets A and B, MATLAB can perform the usual set operations of union $A \cup B$, intersection $A \cap B$ and difference $A \setminus B$:

```
>> A = [1 2 3 4]
>> B = [2 4 6 8]
>> union(A, B)
>> intersect(A, B)
>> setdiff(A, B)
```

Exercise 3.4 Use MATLAB to verify that if $X = \{1,2,3,5\}$, $Y = \{2,3,4\}$ and $Z = \{1,3\}$ then

$$X - (Y \cup Z) = (X - Y) \cap (X - Z) \quad \text{and} \quad X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z).$$

(Of course this does not constitute a proof of these general results!) □