Math 347H Use MATLAB to Solve ODEs

February, 2020

Zhengfang Zhou

This note is to show some examples how MATLAB can be used to solve ordinary differential equations.

Part 1. Construct simple vectors and matrices

The simplest way to create a vector or a matrix in MATLAB is to use the matrix constructor operator, [ ]. Create a row vector (That is, an $1 \times n$ matrix) by entering elements within the brackets. Separate each element with a comma or space. For example if you want to have a row with elements $1, 2, 3, 4, 5$, you type

$$>> row = [1, 2, 3, 4, 5], \quad \text{or} \quad >> row = [1\ 2\ 3\ 4\ 5].$$

To create a matrix with multiple rows, you enter the each row as above, and separate rows with a semicolon.

For example, to constructs a 3 row, 5 column (or 3-by-5) matrix of numbers (make sure that all rows must have the same number of elements), you enter

$$>> A = [\ 12\ 62\ 93\ -8\ 22;\ 16\ 2\ 87\ 43\ 91; -4\ 17\ -72\ 95\ 6\ ],$$

which will generate the following output

$$A = \begin{pmatrix} 12 & 62 & 93 & -8 & 22 \\ 16 & 2 & 87 & 43 & 91 \\ -4 & 17 & -72 & 95 & 6 \end{pmatrix}$$

When entering signed numbers into a matrix, make sure that the sign immediately precedes the numeric value.

Some special matrices:

$>> ones(m,\ n)$ will create a $m \times n$ matrix of all ones.

$>> zeros(m,\ n)$ will create a $m \times n$ matrix of all zeros.

$>> eye(n)$ creates the identity matrix of $n \times n$.

diag (vector) creates a diagonal matrix from the vector (row or column).

magic(n) will create a square $n \times n$ matrix with rows, columns, and diagonals that add up to the same number.

Examples:

$>> eye(3)$ will generate $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

$>>$ diag([1, 2, 3]) will generate $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$.

$>> A = magic(5)$ will generate $A = \begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix}$

Part 2. Plot Graphs, Direction Fields

1. Basic plots

The matlab command plot plots points in a two-dimensional figure and connects them with straight line segments. If y is a vector with n entries, then the command plot(y) plots the points $(1, y(1)), (2, y(2)), (3, y(3)), ..., (n, y(n))$. If x is another vector with the same number of entries as y, then the command plot(x,y) plots the points $(x(1), y(1)), (x(2), y(2)), (x(3), y(3)), ..., (x(n), y(n))$. The plot is displayed in a separate window. For example, the commands

$>> y = [1\ 0\ -1\ 3\ 4]$

$>> plot(y)$ yield the plot in Figure 1 below.

The commands

$>> x = [1, 3, 4, 7, 8];\ \ y = [1, 0, -1, 3, 4];$

$>> plot(x, y)$ will generate Figure 2.

2. Plot graphs of functions

Let $y = f(x)$ is a function, $x \in [a, b]$. To display the graph of a function on this interval $[a, b]$, we plot a sufficiently dense, but finite, set of points on the graph and connect them with (very short) line segments. You need to first define a vector that gives the set of x -coordinates of the points in the graph. In most cases this set has the form

$$\{x = a + ks \mid k = 0, 1, ..., (b - a)/s\} = \{a, a + s, a + 2s, ..., b - s, b\},$$

where s is called the step size between consecutive points in the set. Smaller step sizes produce a 'prettier' pictures, but require more computation. To produce the vector of x values in MATLAB, we use the colon operator. The command $x = a : s : b$ creates a vector x with approximately $(b - a)/s$ entries going from a to b with steps of size s. Next, the command $y = f(x)$ produces the vector of corresponding y -values, and the command $plot(x, y)$ produces the desired plot. For example, the commands

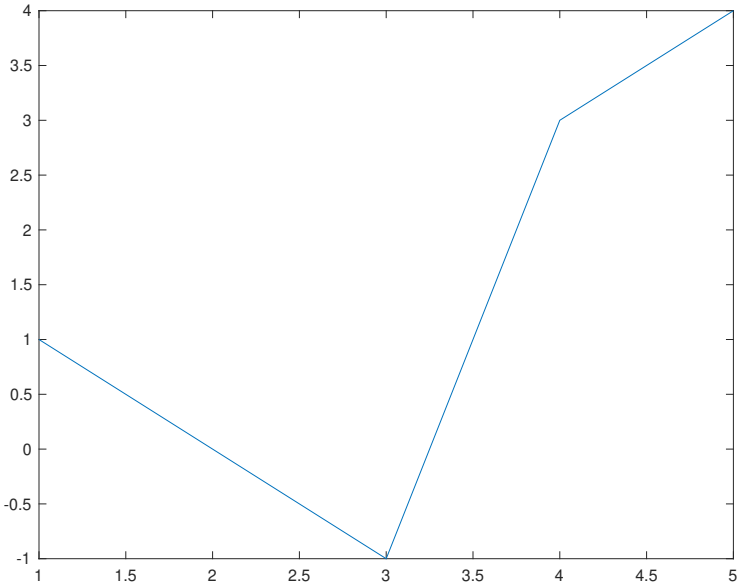

Figure 1: A Basic Plot

$>> x = -3 : 0.01 : 3;\ y = x.^3 + \sin(x) - 5;\ plot(x, y)$ generate Fiqure 3.

There are a variety of optional arguments for the plot command in MATLAB that allow users to change colors, change the symbols MATLAB uses as markers for the points being plotted, etc., as well as additional commands to add title to figures, add grids to the plot, and so on. For details, please use MATLAB help.

MATLAB allows users to plot more than one graph in the same figure. If you type $plot(x, y, ...)$, then change something and type $plot(x, y, ...)$ again, the first plot is simply replaced by the second one. If you want to plot two graphs in the same figure window, you can use the command hold on which freezes the current graph in the figure window, so that the next plot command adds a graph to the current graph, rather than replacing it. For example, the commands

$>> x = -3 : 0.01 : 3;\ y = x.^3 + \sin(x) - 5;$

$>> plot(x, y)$

>> hold on

$>> z = [-2, -1, 0, 1, 2];\ u = z.^3 + \sin(z) - 5;$

$>> plot(z, u, 'ro')$ generate Fiqure 4.

3. Plot Direction Fields

To sketch a direction field for a first order differential equation with MATLAB, the equation must be in normal form $y' = f(x, y)$, and we use the MATLAB functions meshgrid to creates a grid of points in the (x, y)-plane, and quiver to plot little vectors at each point, whose directions

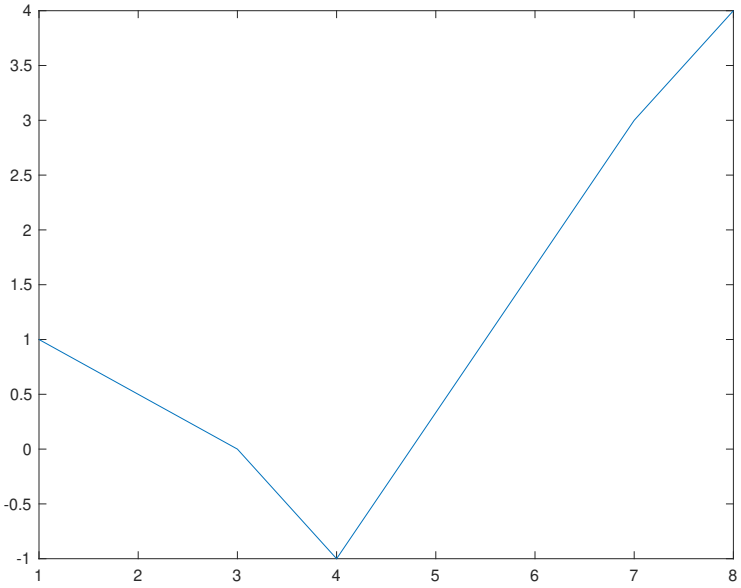

Figure 2: A Basic Plot

are determined by the righthand side of the differential equation $f(x, y)$. Suppose that you want to draw the direction field in a rectangle $[MinX, MaxX] \times [MinY, MaxY]$ with step sizes h, we should use the following commands:

$[X, Y] = meshgrid(MinX : h : MaxX, MinY : h : MaxY])$ to generate a uniform grid;

$dY = f(X, Y)$ to compute the values of $f(x, y)$ = slopes of the solution curve at grid points determined by $[X, Y]$;

$dX = ones(size(dY))$ to create a matrix with every element 1 and the same size as the matrix dY;

$quiver(X, Y, dX, dY)$ to plot a vector $(1, dY)$ at each mesh point.

Usually the picture is not very good since vectors in dY may have vastly different lengths. To fix this problem, you can normalize the vector by inserting the command

$L = sqrt(1 + dY.^2)$ before the *quiver* command, and modify the *quiver* command to

$quiver(X, Y, dX./L, dY./L)$.

If the normalized vectors $dX/L, dY/L$ is too long, you can scale them, also axis tight command may eliminate the white space around the vector fields.

Example: Draw the director filed of $y' = \cos(2x) - xy^2$ on $[0, 4] \times [-4, 4]$ with step size $h = 0.2$

$>> [X, Y] = meshgrid(0 : 0.2 : 4, -4 : 0.2 : 4);$

$>> dY = cos(2 * X) - X. * Y.^2;$

$>> dX = ones(size(dY));$

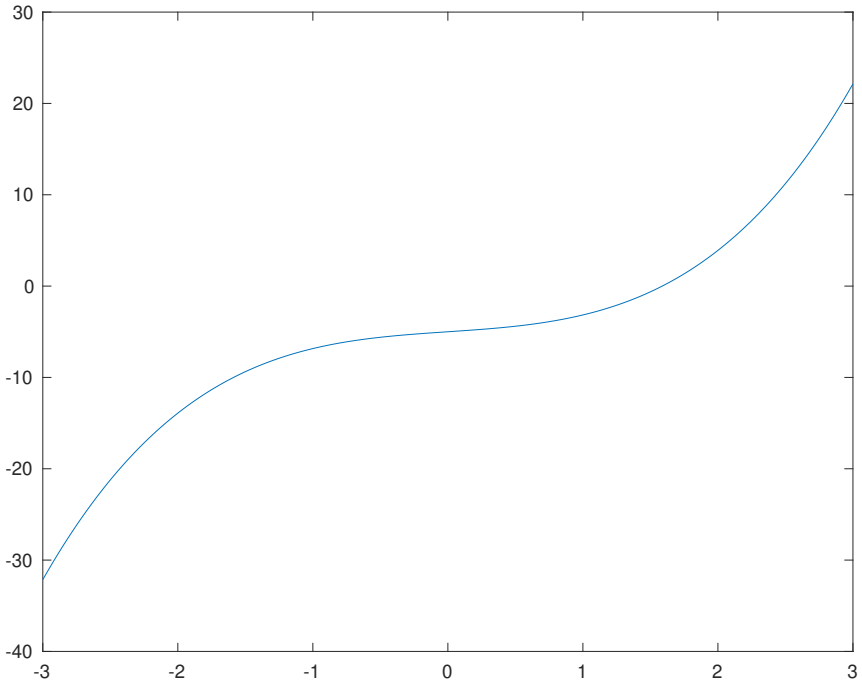

Figure 3: The graph of $y = x^3 + \sin x - 5$

$>> L = sqrt(1 + dY.^2);$
$>> quiver(X, Y, dX./L, dY./L, 0.8, 'r')$
>> axis tight generate Figure 5, which is scaled to 0.8 with red arrows.

4. Solve first order ODEs $y' = f(x, y)$.

Even though MATLAB is primarily a numerics package, it also can solve straightforward differential equations symbolically by using built-in command dsolve. WARNING: NOT EVERY ODE HAS an explicit solution!

Example. Find the solution of $y'(x) = xy$.

$>> eqn1 = 'Dy = x * y';$
$>> y = dsolve(eqn1, 'x')$ generate $y = C1 * exp(1/2 * x^2)$.
To solve the initial. value problem $y'(x) = xy,\ y(1) = 1$, use
$>> y = dsolve(eqn1, 'y(1) = 1', 'x')$ generate $y = exp(-1/2) * exp(1/2 * x^2)$.
If you want to plot the solution to get a rough idea of its behavior, you run immediately into two minor difficulties: (1) the expression for $y(x)$ isn't suited for array operations (.*, ./, .^), and (2) y, as MATLAB returns it, is actually a symbol (a symbolic object). The first of these obstacles

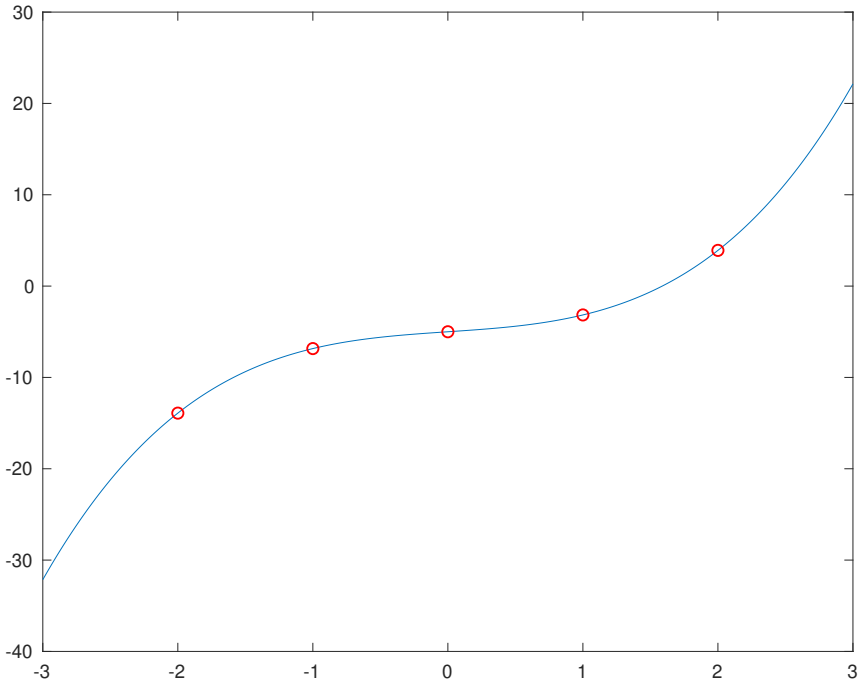

Figure 4: The graph of $y = x^3 + \sin x - 5$ with red dots

is straightforward to fix, using. vectorize() . For the second, we employ the useful command eval(), which evaluates or executes text strings that constitute valid MATLAB commands.

$x = [0 : 0.05 : 1]$; $z = eval(vectorize(y))$; $plot(x, z)$ generates Figure 6.

dsovle can also easily solve the second order or higher otder equations, and linear ODE systems. The commands are similar. Check it out yourself.

5. Numerical Methods to solve first order ODE initial value problems

If an explicit solution is impossible, or not easy, to find, you can always use numerical methods to approximate solutions. Matlab has several different functions for numerical solutions of ODEs. Most popular one to use is *ode45* based on Runge-Kutta method (4th and 5th order expansion). For other methods and their strength and weakness, Numerical Analysis will study them carefully.

Example. Find the solution of $y'(t) = y(t) + t\, y^2(t)$, $y(0) = 1$ for $t \in [0, 4]$.

Step 1. Create and save the file dequation.m for the function $y + t\, y^2$:

function $dydt = dequation(t, y)$ % You can use any name for dequation

$dydt = y + t * y^2$

Step 2. At Basic Command window:

>> $[t, y] = ode45('dequation', [0, 4], 1)$ % The solution is saved in two vectors t and y.

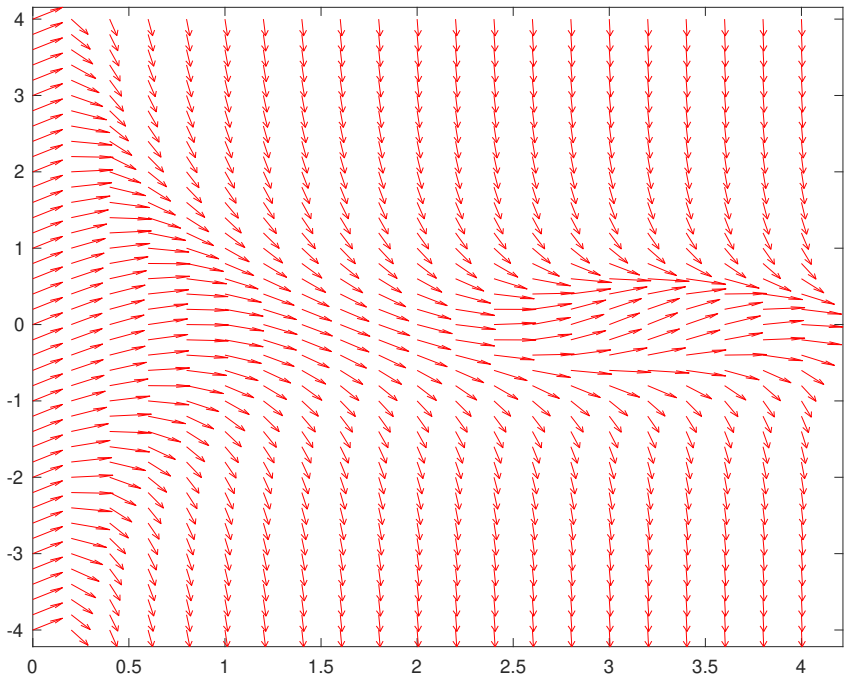

Figure 5: Direction Field of $y' = \cos 2x - xy^2$

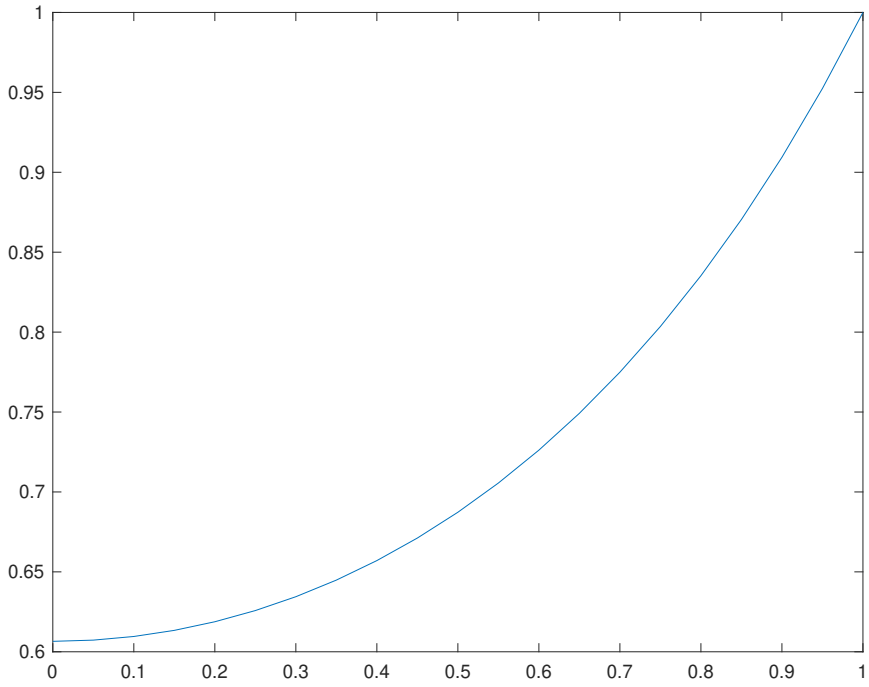

Figure 6: Solution of $y = xy,\ y(1) = 1$

>> $plot(t, y)$ % To plot the graph of the numerical solution of $y(t)$.

Similarly, you can solve second order equation $y'' + p(t)y' + q(t)y = g(t)$ with initial data $y(t_0) = y_0,\ y'(t_0) = y_1$.

Step 1. Convert the equation into an equivalent system. Let $x_1 = y,\ x_2 = y'(t)$, then

$$\begin{cases} x_1' &= x_2; \\ x_2' &= -q(t)x_1 - p(t)x_2 + g(t); \\ x_1(t_0) &= y_0; \\ x_2(t_0) &= y_1. \end{cases}$$

Step 2. Create and save a .m file to define the vector valued function corresponding to the right hand side.

Step 3. Use ode45 command to find the numerical solution on a domain $[t_0, t_1]$ with specified initial data. You can use the usual *plot* command to visualize the solution.

Example. Find the solution $y'' + ty' + e^t y = \sin t$ with $y(0) = 2,\ y'(0) = 4$.

It is equivalent to

$$\begin{cases} x_1' &= x_2; \\ x_2' &= -e^t x_1 - t x_2 + sint; \\ x_1(0) &= 2; \\ x_2(0) &= 4. \end{cases}$$

Then create the following file and save it as $F.m$:

function $xp = F(t, x)$

$xp = \text{zeros}(2, 1)$; % Since out put must be a column vector $\begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix}$.

$xp(1) = x(2)$;

$xp(2) = -exp(t) * x(1) - t * x(2) + sin(t)$;

Finally, at MATLAB prompt, type

>> $[t, x] = ode45('F', [0, 3], [2, 4])$; % $[2, 4]$ is the initial data, $[0, 3]$ is the solution interval with initial data specified at 0.

since $x_1(t) = y(t)$, we can plot the solution by using the following command:

>> $plot(t, x(:, 1))$, which produces Figure 7.

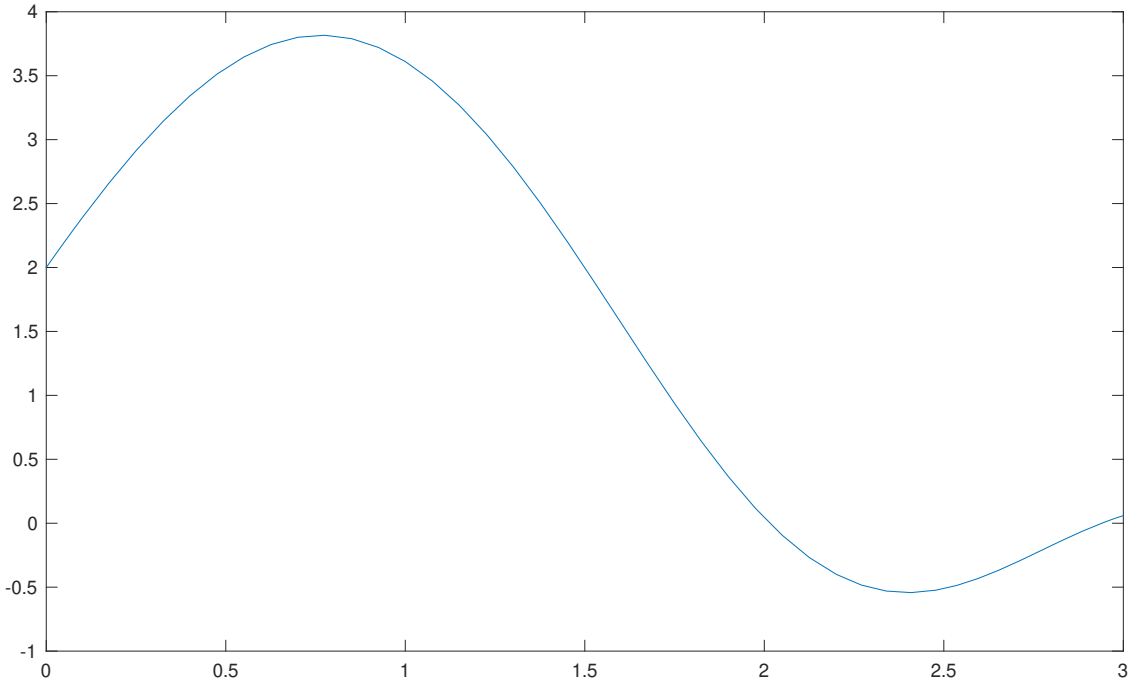

Figure 7: Solution of $y'' + ty' + e^t y = \sin t$ with $y(0) = 2,\ y'(0) = 4$