Plotting graphs, direction fields and contours with MatLab

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

$$(1, y(1)),\ (2, y(2)),\ (3, y(3)),\ \ldots,\ (n, y(n)).$$

If x is another vector with the *same number of entries* as y, then the command plot(x,y) plots the points

$$(x(1), y(1)),\ (x(2), y(2)),\ (x(3), y(3)),\ \ldots,\ (x(n), y(n)).$$

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,

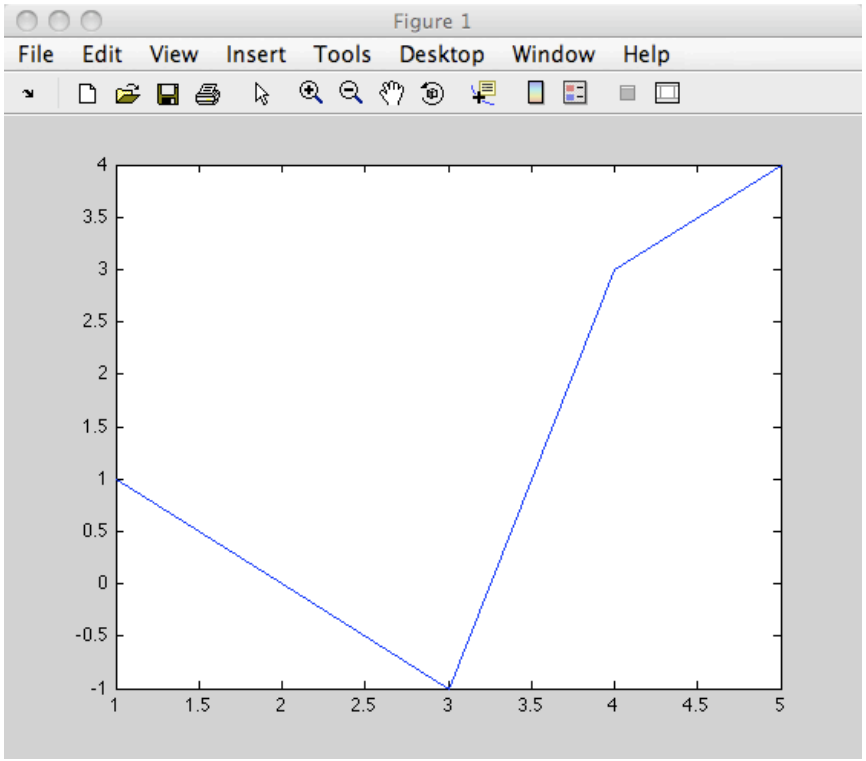

Figure 1: A basic plot.

and the commands

```
>> y = [1 0 -1 3 4]; x=[1 3 5 7 10];
>> plot(x,y)
```

produce the plot in Figure 2 below.

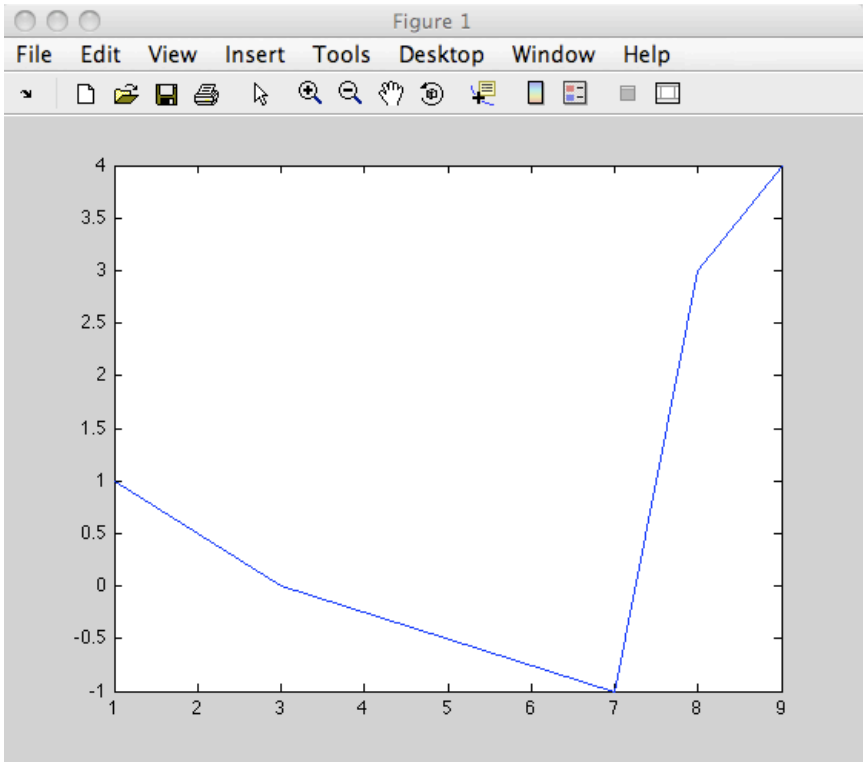

Figure 2: Another basic plot.

2. Plotting graphs of functions.

If $y = f(x)$ is a function, then the *graph* of this function is the collection of points $\{(x, f(x)) : x \in D\}$ in the plane, where D is the *domain* of the function, i.e., the set of points x where the function is defined. To display the graph of a function we typically choose some interval of values for x, and plot the points of the graph whose x-coordinates lie in the given interval.

In practice, we can't plot all of the points of a graph, even if we limit x to a bounded interval, because there are typically infinitely many points to plot. What we do (and by we, I mean graphing calculators and other software that draws pictures for us), is plot a sufficiently dense, but **finite**, set of points of the graph and connect them with (very short) line segments.[†]

To plot the graph of a function in MATLAB , 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

$$\{x = a + ks : 0 \le k \le (b-a)\cdot s\} = \{a, a+s, a+2s, \ldots, b-s, b\},$$

where a is the smallest x-value we use, b is the largest x-value and s is the *step*

[†]More sophisticated programs connect the dots with 'curved' segments.

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

```
>> x=-3:0.01:3;
>> y=-x.^3+2*x.^2+4*x-8;
>> plot(x,y)
```

produce the graph of the function $y = -x^3+2x^2+4x-8$, for x between -3 and 3 in Figure 3 below.

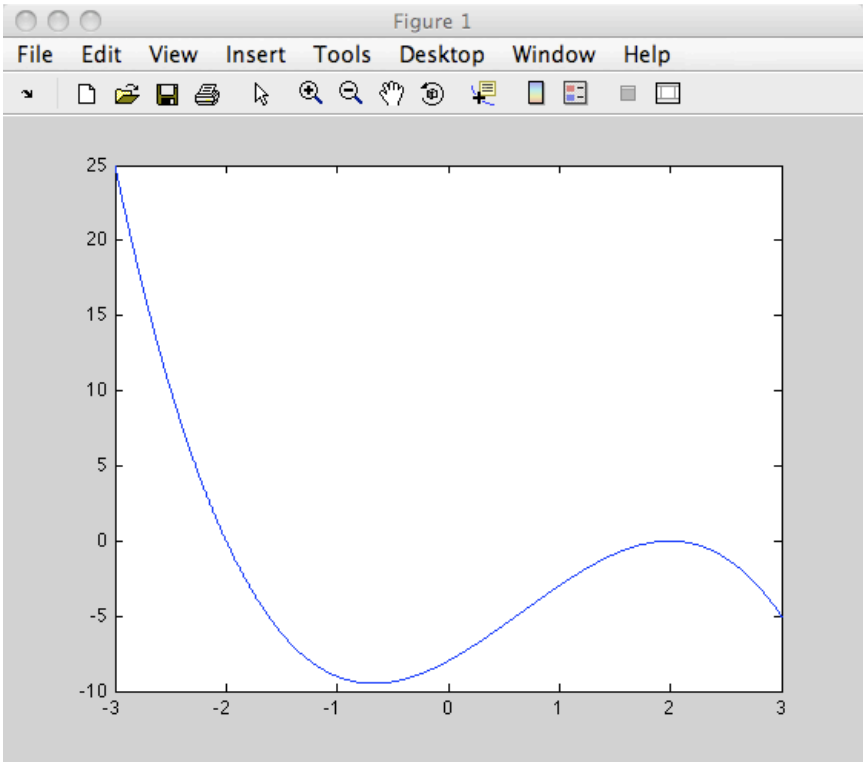

Figure 3: The graph of $y = -x^3 + 2x^2 + 4x - 8$

3. Optional arguments and plotting commands.

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.

[‡]You need to enter an explicit function at this point, like y= sin(x) or y=x.^2-3*x+1, not the generic expression y=f(x).

The default style of a plot in MATLAB consists of single dots for the markers of the points being plotted, a solid line for the segments between the points and the color blue. To change these parameters, we add a *string* of letters and symbols following the vector(s) to be plotted in the plot(...) command. For example, the string 'go' specifies green circles as the markers of the points (with no connecting line segments) and the string ':r+' specifies red plus signs as the markers and (red) dotted lines for the connecting segments. Strings must appear between single quotes ' (on the key with the double quotes, next to the return key). The commands

```
>> x=-3:0.1:3;
>> y=-x.^3+2*x.^2+4*x-8;
>> plot(x,y,'g+')
```

plots the designated points of the graph of the function $y = -x^3 + 2x^2 + 4x - 8$, for x between -3 and 3 with green plus signs, shown in Figure 4.

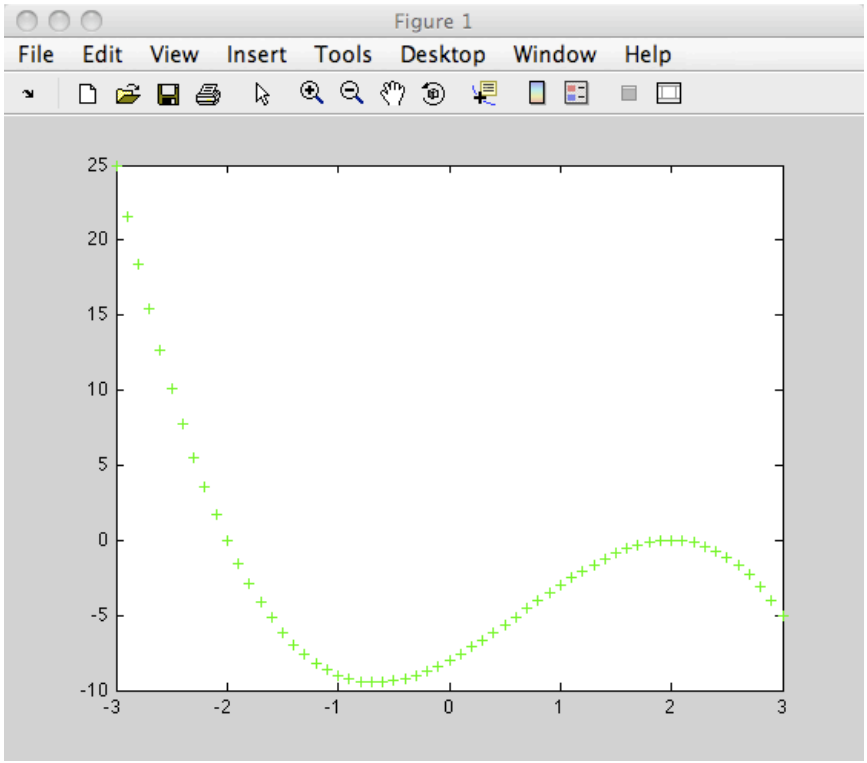

Figure 4: Another graph of $y = -x^3 + 2x^2 + 4x - 8$

Since no line style was specified, MATLAB assumes that no line segments should be used. With x and y as before, the command

```
>> plot(x,y,':g+')
```

connects the points of the previous plot with dotted line segments, as depicted in Figure 5. Note that when the markers are very close together, MATLAB doesn't put line segments between them.

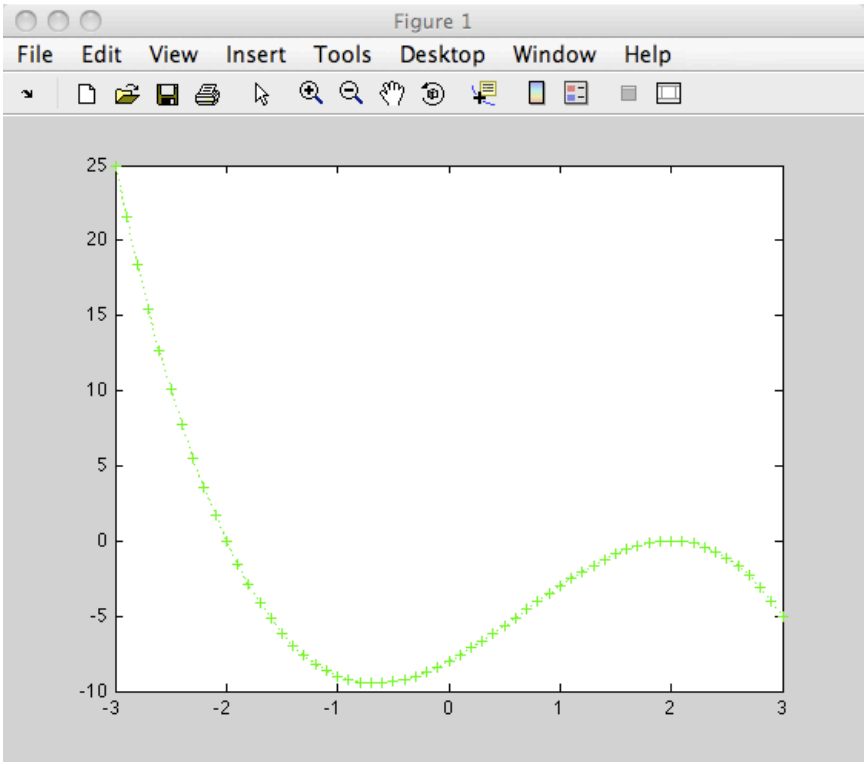

Figure 5: Yet another graph of $y = -x^3 + 2x^2 + 4x - 8$

The color choices, line styles and markers that MATLAB offers are listed in the table below, together with the symbols used to invoke them.

Marker	Symbol	Color	Symbol	Line style	Symbol
dot (default)	.	blue (default)	b	solid (default)	-
diamond	d	cyan	c	dotted	:
circle	o	green	g	dashed	--
hexagram	h	yellow	y	dash-dot	-.
pentagram	p	red	r		
plus sign	+	magenta	m		
square	s	black	k		
star	*				
triangle down	v				
triangle up	^				
triangle right	>				
triangle left	<				
x-mark	x				

4. Additional plotting commands.

There are additional basic plotting commands in MATLAB that allow users to plot more than one graph in the same figure, insert a basic grid in the figure, name a figure, etc.

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+2*x.^2+4*x-8;
>> plot(x,y)
>> hold on
>> z = [-2 -1 0 1 2];
>> u=-z.^3+2*z.^2+4*z-8;
>> plot(z,u,'ro')
```

draws the graph of the function $y = -x^3 + 2x^2 + 4x - 8$ and puts red circles at the points $(-2, 0)$, $(-1, -9)$, $(0, -8)$, $(1, -3)$ and $(2, 0)$ as shown in Figure 6.

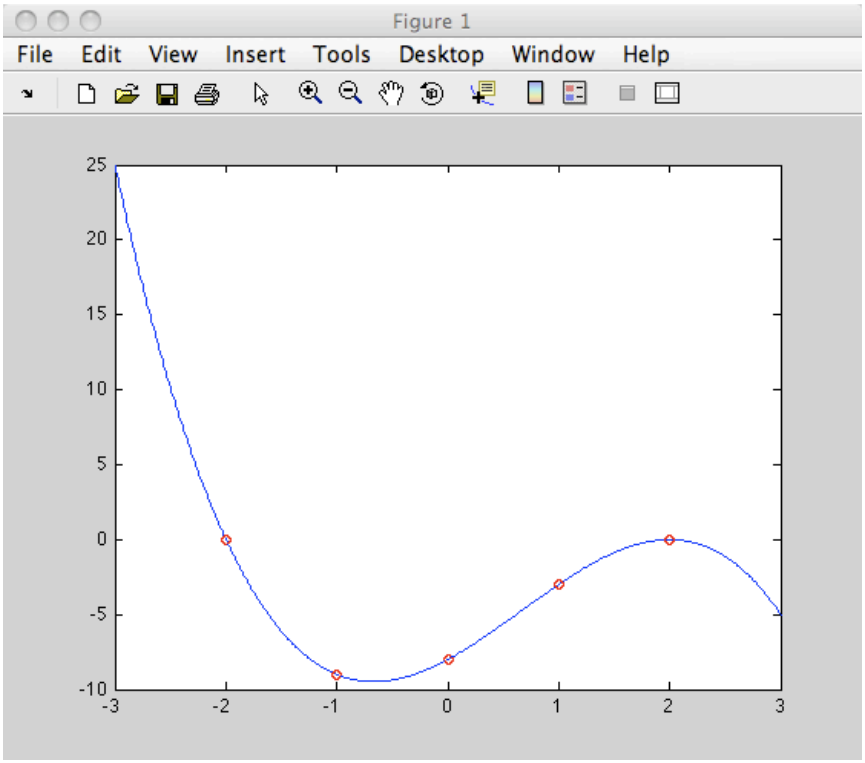

Figure 6: The graph of $y = -x^3 + 2x^2 + 4x - 8$ again (with red dots).

Following the previous commands with

```
>> grid on
>> title('graph of y=-x^3+2x^2+4x-8')
```

adds a grid to the figure, as well as the chosen title, as shown in Figure 7. Notice how MATLAB interprets the ^ symbol in the displayed title. Needless to say, you can invoke grid on without title('...') or vice versa.

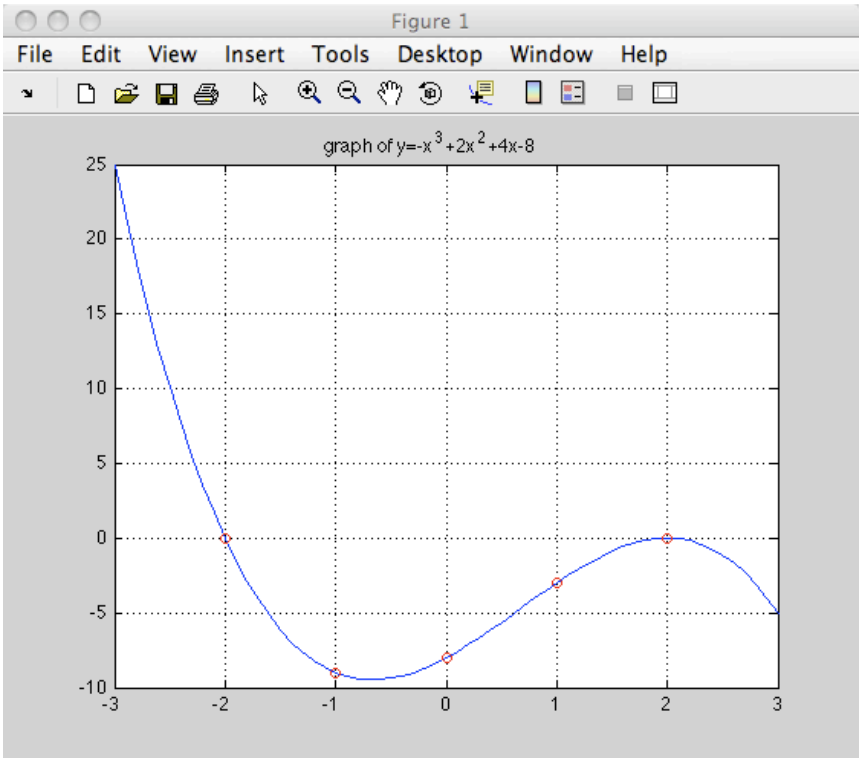

Figure 7: Grids and titles and cubics, oh my.

5. Direction fields.

To sketch a direction field for a first order differential equation with MATLAB, the equation must be in **normal form**. A first order differential equation is in normal form if it looks like this

$$y' = f(t, y).$$

To sketch a direction field, we use the MATLAB functions **meshgrid** and **quiver**. Briefly, **meshgrid** creates a grid of points in the (t, y)-plane, and **quiver** plots little vectors at each point, whose directions are determined by the righthand side of the differential equation.

The **basic** syntax is something like this:

```
>> [T Y] = meshgrid(minT:step:maxT, minY:step:maxY);
>> dY = f(T,Y);
>> dT = ones(size(dY));
>> quiver(T,Y, dT, dY);
```

Explanation:

a. **meshgrid** creates a uniformly spaced grid of points in the rectangle

$$\{(T, Y) : minT \leq T \leq maxT, \quad minY \leq Y \leq maxY\}$$

and assigns the horizontal coordinates of the points to T and the vertical coordinates to Y. The spacing is determined by the parameter 'step'.

b. The command 'dY = f(T,Y)' computes the matrix of slopes of the vectors attached to each point in the grid. Note that you need to type in an actual function of T and Y here (not just write 'f(T,Y)').

c. This creates a matrix of 1s of the same dimension as dY.

d. The vector that the 'quiver' command plots at the point (T,Y) in the grid will be parallel to the vector (dT,dY) = (1,dY), giving it the correct slope. Quiver automatically scales the vectors so that they do not overlap.

I'll use the syntax above to sketch a direction field for the differential equation $y' = \cos 2t - y/t$ in the rectangle $\{(t, y) : 0 \leq t \leq 8,\ -4 \leq y <\leq 4\}$:

```
>> [T Y]=meshgrid(0:0.2:4,-4:0.2:4);
>> dY=cos(2*T)-Y./T;
>> dT=ones(size(dY));
>> quiver(T, Y, dT, dY);
```

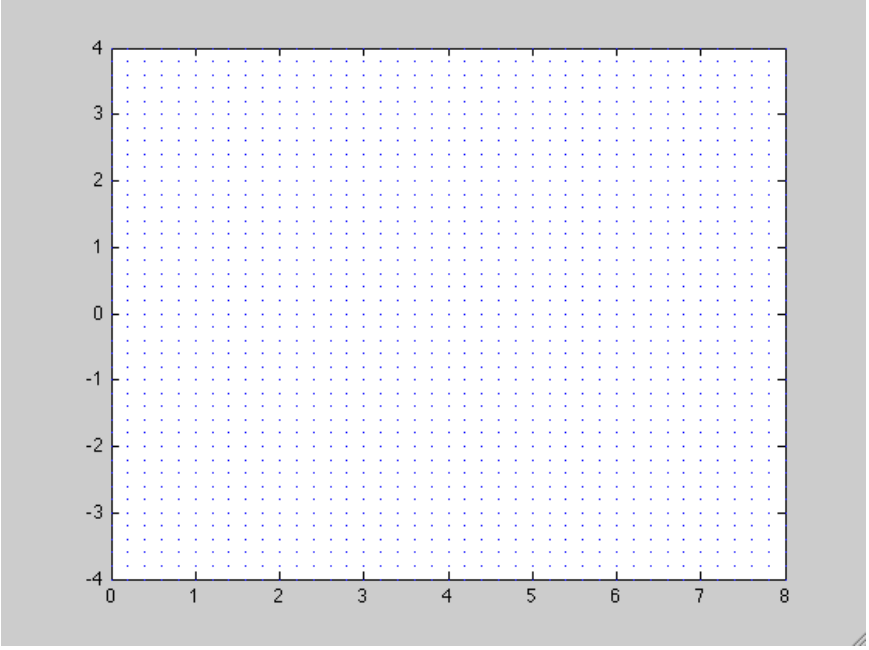

Figure 8: First try at direction field for $y' = \cos 2t - t/y$.

The (unsatisfying) result is in Figure 8, above. The problem lies with the automatic scaling feature of quiver, and the fact that the vectors in the direction field above have vastly different lengths. To fix this problem, we can scale all the vectors to have unit length, by dividing each one by its length. Inserting the command

```
>> L=sqrt(1+dY.^2);
```

before the quiver command in the sequence above, and then changing the quiver command to

```
>> quiver(T, Y, dT./L, dY./L)
```

produces the output in Figure 9.

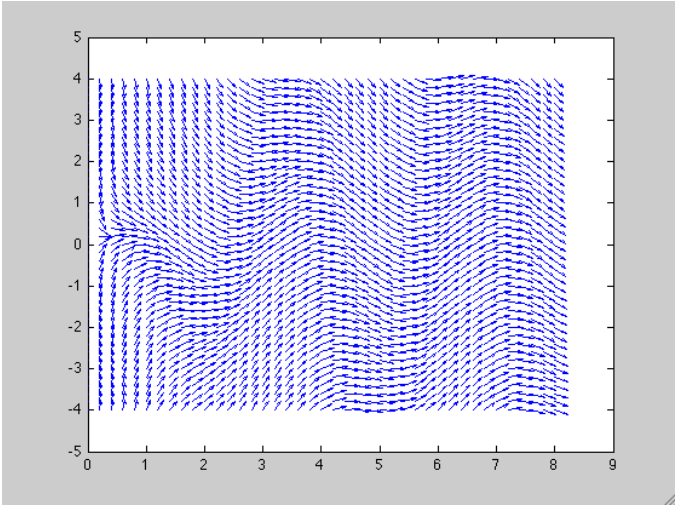

Figure 9: Second try at direction field for $y' = \cos 2t - t/y$.

This one looks like a direction field, but there are still things to fix, e.g. the white space around the direction field and the fact that the vectors are now overlapping in places. The pair of commands

```
>> quiver(T, Y, dT./L, dY./L, 0.5)
>> axis tight
```

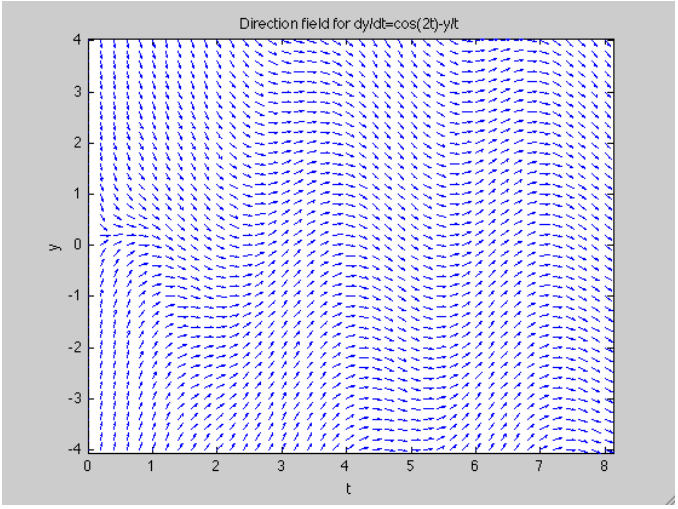

Figure 10: Rescaled direction field without annoying white space

produces the more pleasing direction field in Figure 10, where the vectors have been scaled to half their former length with the optional argument '0.5' in the quiver command and the white space is removed with the 'axis tight' command.

Since quiver is a plotting command many (if not all) of the optional plotting arguments and commands can be used in conjunction with quiver. You can add labels to the axes, titles to the figures, change colors, etc. For example, the commands

```
>> xlabel 't', ylabel 'y';
>> title 'Direction field for dy/dt=cos(2t)-y/t';
>> quiver(T, Y, dT./L, dY./L, 0.5,'r'), axis tight
```

produce the red direction field, complete with axis labels and title in Figure 11.

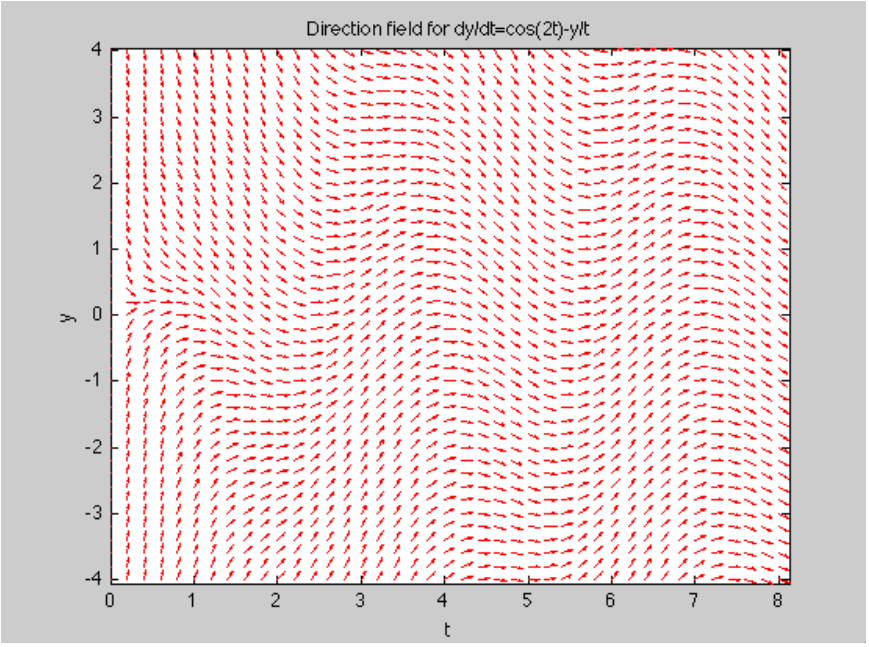

Figure 11: Red direction field for $y' = \cos 2t - t/y$.

6. Contour plots.

A *contour plot* is essentially the graph of an equation of the form $F(x, y) = C$. To graph such equations using MATLAB , we use the command **contour** together with **meshgrid**. The basic syntax is

```
>> [X Y] = meshgrid(minX:step:maxX, minY:step:maxY);
>> contour(X, Y, F(X,Y));
>> %% Remember: you have to type an actual function for F(X,Y).
```

This pair of commands will plot *several contours*, i.e., the result will be the graphs of several equations of the form F(X,Y)=C, for different values of C, which MATLAB chooses automatically, based on the grid and other considerations. E.g., the pair of commands

```
>> [X Y] = meshgrid(-5:0.2:5,-2:0.2:2);
>> contour(X,Y,Y.^4+X.^2);
```

produces the plots in Figure 12. If you want to choose the particular values of C that MATLAB uses in the contour plot, you can do so with the optional argument $[C_1, C_2, \ldots, C_k]$ in the **contour** command. This tells MATLAB to plot contours for the k equations $F(X,Y) = C_j$, for $1 \le j \le k$. E.g., replacing the **contour** command above with

```
>> contour(X,Y,Y.^4+X.^2, [1,4,16]);
```

produces the contours in Figure 13.

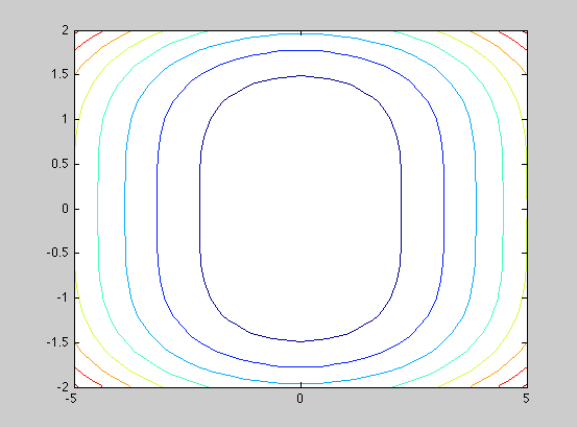

Figure 12: Contour plots for $y^4+x^2 = C$.

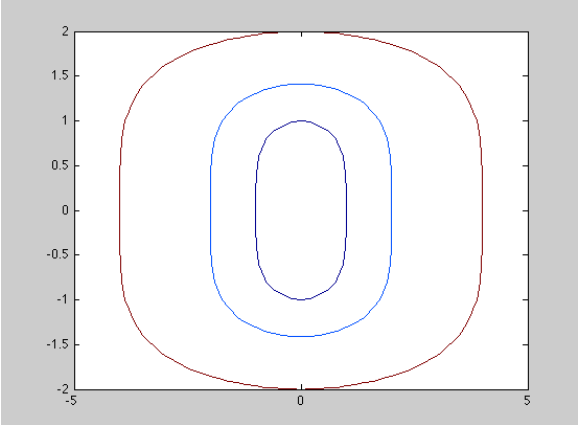

Figure 13: Contours for $C = 1$, 4 and 16.

7. Combining direction fields and contour plots.

Contour plots may be combined with direction fields to give a more complete picture of the solutions of differential equations. The contour plot is particularly useful in the case of separable differential equations, where implicit solutions are (relatively) easy to find, but explicit solutions are not.

Consider, for example the separable differential equation

$$\frac{dy}{dx} = \frac{4-x}{y^3+2}.$$

A direction field for this differential equation is given in Figure 14.

While the direction field by itself does reveal certain patterns, it may not be completely clear what solutions look like. You should verify that the general solution of the differential equation above is given by $y^4+2x^2-16x+8y = C$. Adding a contour plot of this equation to the direction field above, produces a much clearer picture of the nature of the solutions, as shown in Figure 15. The graph of such an equation is called an *integral curve* of the differential equation.

Each curve corresponds to two solutions of the differential equation. The 'top' of the curve corresponds to an initial y-value that is greater than $-\sqrt[3]{2}$, and the 'bottom' of the curve corresponds to an initial y-value that is less than $-\sqrt[3]{2}$. Can you explain where the number $-\sqrt[3]{2}$ came from? Can you explain how this value is used to determine the *interval of definition* of a particular solution?

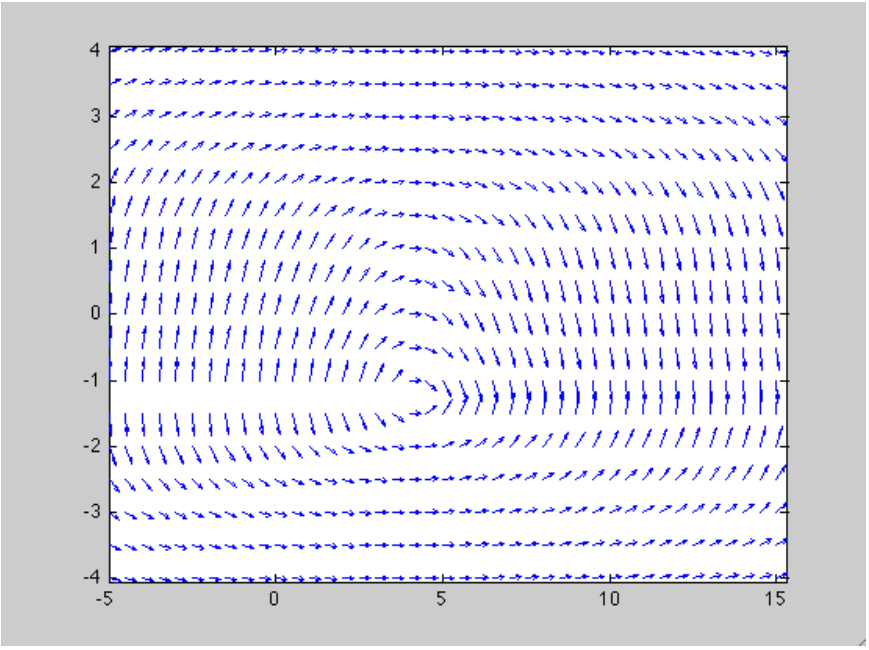

Figure 14: A direction field for $y' = (4 - x)/(y^3 + 2)$.

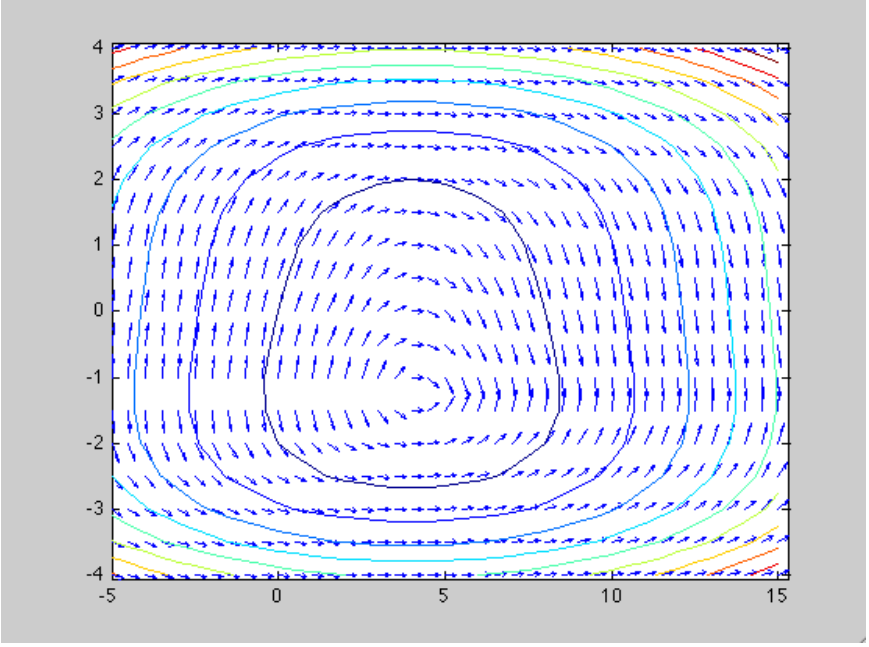

Figure 15: The direction field together with several *integral curves*.