Command to plot a vector field:

Here is an example of the syntax which produces the vector field associated to a first-order ODE (type this all in one cell):

```
g[x_,y_] : = 3y;
VectorPlot[{1,g[x,y]},{x,-3,-3},{y,-3,3}, VectorPoints -> 20,
Axes -> True, VectorScale -> {Automatic, Automatic, None},
VectorStyle -> Orange]
```

What this command does: First, the function $g(x, y)$ is defined in the first line. If you wanted to sketch a vector field for a different equation, you can change the `3y` here to whatever formula is given by $g(x, y)$. Second, the `VectorPlot` command tells *Mathematica* to sketch the vector field. The relevant ingredients here are as follows:

- The `{x, -3, 3}` and `{y, -3, 3}` tell *Mathematica* to produce a picture that runs from $x = -3$ to $x = 3$ and $y = -3$ to $y = 3$, i.e. it specifies the range of the picture.
- The `VectorPoints -> 20` tells *Mathematica* how many vectors to draw. If you change 20 to a larger number, you get more densely drawn arrows. This produces a more accurate picture of the vector field, but the program will take longer to run. On the other hand, if you change 20 to a smaller (positive) number, less arrows will be drawn, resulting in a less dense picture.
- The `Axes -> True` tells *Mathematica* to draw the x- and y- axes on the picture.
- The `VectorScale -> {Automatic, Automatic, None}` tells *Mathematica* to make sure all the vectors are scaled so that you can see them (without this command many of the vectors drawn would be too small to see).
- The `VectorStyle -> Orange` tells *Mathematica* to make the vectors orange. An interesting thing to do is to replace this command with the following phrase: `VectorColorFunction -> Hue` to get a 'tie-dye' looking picture.

Command to produce a steam plot:

Here is a command which plots both the vector field and some solution curves to the differential equation $y' = g(x, y)$. It comes from taking the command above and adding some additional programming inside the `VectorPlot` command:

```
g[x_,y_] : = 3y;
VectorPlot[{1,g[x,y]},{x,-3,-3},{y,-3,3}, VectorPoints -> 20,
Axes -> True, VectorScale -> {Automatic, Automatic, None},
VectorStyle -> Orange, StreamPoints -> 35, StreamScale -> Full,
StreamStyle -> {Blue, Thick}]
```

As before, the first line defines $g(x, y)$; the `{x, -3, 3}` and `{y, -3, 3}` determine the range of the picture; the `VectorPoints -> 20` tell *Mathematica* how many arrows to draw; `Axes -> True` tells *Mathematica* to draw the x- and y- axes; and the `VectorScale` command ensures the vectors are big enough to see.

What' s new here are the commands regarding "Streams". The `StreamPoints -> 35` asks *Mathematica* to draw 35 stream lines (a stream line is a graph that ' follows' the vector field as described in lecture). If you lower the number 35, less stream lines are drawn and if you increase the number, more stream lines are drawn.

The `StreamScale -> Full` command ensures that the stream lines are connected when they should be (so that you get a much better picture). The last bit tells *Mathematica* what color to make the stream lines and to make them thick.

Command to produce a steam plot through a particular point:

Suppose you wanted to draw one stream line that went through one specified point. In this case, you modify the above command by changing the `StreamPoints` directive from `35` to `{{a,b}}` where you want to draw the stream line through the point (a, b). For example, in the above picture, if you wanted the stream line passing through the point $(1, 2)$, your command would be :

```
g[x_,y_] : = 3y;
VectorPlot[{1,g[x,y]},{x,-3,-3},{y,-3,3}, VectorPoints -> 20,
Axes -> True, VectorScale -> {Automatic, Automatic, None},
VectorStyle -> Orange, StreamPoints -> {{1,2}}, StreamScale -> Full,
StreamStyle -> {Blue, Thick}]
```