Drawing graphics directly in LaTeX: TikZ

Purpose

These notes explain some basics of how to use TikZ, a LaTeX package which can be used to draw graphics by writing code.

I don't recommend you do this - you are probably better creating graphics in something like Inkscape. But these notes are provided for students who are particularly interested in LaTeX or who would like to see how graphics can be created programmatically.

Starter example

Here is a basic LaTeX document that loads `tikz`, `etoolbox` (which is used for programming) and draws a very simple picture.

```
\documentclass{article}

\usepackage{tikz}
\usepackage{etoolbox}

\begin{document}\pagenumbering{gobble}% get rid of page numbers

% The actual TikZ picture - this is the part to edit
\begin{tikzpicture}
    \draw (0,0) -- (0,1) -- (1,1) -- (1,0) -- cycle;
\end{tikzpicture}

\end{document}
```

All the TikZ code must be within the TikZ environment, which is how LaTeX knows you are drawing with TikZ at this point. This means your code should be between `\begin{tikzpicture}` and `\end{tikzpicture}`.

All TikZ lines must terminate with a semi-colon `;`.

The drawing grows to accommodate the coordinates you give `(x,y)`, with horizontal first coordinate running from negative values to the left to positive right and vertical second coordinate running from negative bottom to positive top.

Drawing simple lines and shapes

As seen in the example above, you draw a line by giving Cartesian coordinates to the command `\draw` in the form `(x,y)` or polar coordinates in the form `(angle:length)`. For example:

```
\begin{tikzpicture}
    \draw (0,0) -- (3,2);
\end{tikzpicture}
```

You can draw a chain of points joined by lines by adding extra coordinates.

```
\begin{tikzpicture}
    \draw (0,0) -- (3,2) -- (-3,2) -- (-2,-2);
\end{tikzpicture}
```

There are various options for changing the style of the lines. Options for thickness are: `ultra thin`, `very thin`, `thin`, `semithick`, `thick`, `very thick` and `ultra thick`. For example:

```
\begin{tikzpicture}
    \draw[blue, dotted, very thick, rotate=45] (0,0) -- (1,0) -- (1,1);
\end{tikzpicture}
```

You can join a set of coordinates back to the starting point (as we did drawing the square above) using `cycle`. See if you can draw a triangle with green, dashed, thick lines.

It isn't always necessary to cycle, however, since some shapes have built-in commands. Here are some examples. Note that if you run all these together they will appear over oneanother - you can avoid this by moving the coordinates.

```
\begin{tikzpicture}
    \draw (-2,-1) rectangle (2,1); % rectangle between (-2,-1) and (2,1)
    \draw (0,0) circle [radius=1.5]; % circle with centre (0,0) and radius 1.5
    \draw (0,0) circle [x radius=2, y radius=1]; % ellipse with major axis 2 a
\end{tikzpicture}
```

You can fill shapes with colour. For example:

```
\begin{tikzpicture}
    \draw [fill=red, ultra thick] (0,0) -- (0,4) -- (1,5) -- (5,5) -- (5,0) --
\end{tikzpicture}
```

You can draw lines with arrows etc. For example:

```
\begin{tikzpicture}
    \draw [->] (0,0) -- (0,2);
    \draw [|->] (1,0) -- (3,2);
    \draw [<->] (3,1) -- (5,1);
\end{tikzpicture}
```

You can also draw in three dimensions.

```
\begin{tikzpicture}
    \draw [->] (0,0,0) -- (5,0,0);
    \draw [->] (0,0,0) -- (0,5,0);
    \draw [->] (0,0,0) -- (0,0,5);
\end{tikzpicture}
```

You can place a label using a `node`. For example:

```
\begin{tikzpicture}
    \node [left] at (0,0) {start};
    \draw [->] (0,0) -- (5,0);
\end{tikzpicture}
```

The label can include LaTeX, including mathematics. For example:

```
\begin{tikzpicture}
    \node [left] at (0,0) {start};
    \draw [->] (0,0) -- (5,0);
    \node [right] at (5,0) {$y=\int e^{ax} \, \mathrm{d}x$};
\end{tikzpicture}
```

There is loads more basic functionality you can discover by searching online.

Plotting functions

PGF has a capacity for plotting functions. For example:

```
\begin{tikzpicture}
    \draw [<->] (-5,0) -- (5,0); % x axis
    \draw [<->] (0,-5) -- (0,5); % y axis
    \draw [domain=-2.2:2.8] plot (\x, {pow(\x,3) - pow(\x,2) - 4*\x + 2});
\end{tikzpicture}
```

Possible functions include: `factorial(\x)`, `sqrt(\x)`, `pow(\x,y)`, `exp(\x)`, `ln(\x)`, `log10(\x)`, `log2(\x)`, `abs(\x)`, `mod(\x,y)`, `round(\x)`, `floor(\x)`, `ceil(\x)`, `sin(\x)`, `cos(\x)`, `tan(\x)`, `min(\x,y)` and `max(\x,y)`. The trig functions, alas, default to degrees; to work in radians you follow the variable name with `r`, as in `tan(\x r)`.

This is possibly not the most efficient way of plotting functions available to you, but it could be useful if you are using LaTeX.

Programming

A for loop can be used to create graphics. For example:

```
\begin{tikzpicture}
    \foreach \i in {0,...,10}{
        \draw(0,\i) -- (10,\i);
        \draw(\i,0) -- (\i,10);
    }
\end{tikzpicture}
```

Hopefully, you can spot the crucial elements of a for loop, as you have experienced it in other languages. A dummy variable (`i`) is defined and given limits (`0,...,10`). Then this is accessed using `\i` as part of the loop. The commands that are executed are contained within `{...}`.

You can change the step length of the dummy variable by indicating this with an extra value in the for loop definition, for example, for even values only:

```
\begin{tikzpicture}
    \foreach \i in {0,2,...,10}{
        \draw(0,\i) -- (10,\i);
        \draw(\i,0) -- (\i,10);
    }
\end{tikzpicture}
```

Or to step more frequently:

```
\begin{tikzpicture}
    \foreach \i in {0,0.5,...,10}{
        \draw(0,\i) -- (10,\i);
        \draw(\i,0) -- (\i,10);
    }
\end{tikzpicture}
```

A loop can be used together with the function plotting option to plot multiple functions on a graph. Here, `[scale=5]` is used to make the plot bigger so we can see the detail.

```
\begin{tikzpicture}[scale=5]
    \draw [<->] (0,0) -- (1,0); % x axis
    \draw [<->] (0,0) -- (0,1); % y axis
    \foreach \i in {1,...,10} {
        \draw [domain=0:1] plot (\x, {pow(\x,\i)});
    }
\end{tikzpicture}
```

You can also arrange if statements using commands from `etoolbox`. For example, below we use `\ifnumcomp`. The format is `\ifnumcomp{a}{relation}{b}{1}{2}` so that if `a` is `relation` to `b` it will do `1` and otherwise it will do `2`. `relation` can be >, <, =.

```
\begin{tikzpicture}
    \foreach \i in {1,...,10} {
        \ifnumcomp{\i}{>}{6}
        {\draw (\i,0) circle [radius=0.4];}
        {\draw (\i-0.4,-0.4) rectangle (\i+0.4,0.4);}
    }
\end{tikzpicture}
```

It is possible, though slightly awkward, to get TikZ to perform mathematical operations. For example, the following code evaluates (on `line *`) the remainder from dividing `\i` by `2`, then uses this (via `\pgfmathresult`) in an `\ifnumcomp` statement to decide whether to draw a circle or a square.

```
\begin{tikzpicture}
    \foreach \i in {1,...,10} {
        \pgfmathparse{int(mod(\i,2))} % line *
        \ifnumcomp{\pgfmathresult}{=}{0}
        {\draw (\i,0) circle [radius=0.4];}
        {\draw (\i-0.4,-0.4) rectangle (\i+0.4,0.4);}
    }
\end{tikzpicture}
```

A more complicated example #1

This loops from 0-59 and draws either a long line with a number or short line. Can you see what it is going to do before you run it? Then run it and see if you are correct.

```
\begin{tikzpicture}
    \draw (0,0) circle [radius=5];
    \draw [gray, fill=gray] (0,0) circle [radius=0.1];
    \foreach \i in {0,...,59} {
        \pgfmathparse{int(mod(\i,5))}
        \ifnumcomp{\pgfmathresult}{=}{0}
        {\draw (\i*6:4.5) -- (\i*6:5);\node at (\i*6:5.5) {\Large\pgfmathparse
        {\draw (\i*6:4.9) -- (\i*6:5);}
    }
\end{tikzpicture}
```

A more complicated example #2

A star graph is a graph with a central node, n radial nodes and n edges connecting the central node to each radial node.

You can draw a graph by naming nodes (contain no text `{}`) and then asking TikZ to draw lines between the nodes. For example:

```
\begin{tikzpicture}
    \node [circle, fill=black] at (1.5,0) (node1) {};
    \node [circle, fill=black] at (4,1) (node2) {};
    \node [circle, fill=black] at (5,2) (node3) {};
    \node [circle, fill=black] at (2,3) (node4) {};
    \draw (node1)--(node2);
    \draw (node2)--(node3);
    \draw (node2)--(node4);
\end{tikzpicture}
```

To draw a star graph for five nodes, I could do this using a for loop.

```
\begin{tikzpicture}[scale=3]
    \node [circle, fill=black] at (0,0) (centre) {};
    \foreach \i in {1,...,5} {
        \node [circle, fill=black] at (\i*360/5:1) (node\i) {};
        \draw (centre)--(node\i);
    }
\end{tikzpicture}
```

Say I want to write a command to draw a star graph for a given number of nodes. In LaTeX, we define a new command `\something` using `\newcommand{\something}[number of parameters]{what it does}`. Then a number of parameters are passed to the new command using `\something{one}{two}{...}` and these are accessed by the code as `#1`, `#2` and so on. The `\newcommand` statement goes before the `\begin{document}`.

```
\newcommand{\stargraph}[1]{\begin{tikzpicture}[scale=3]
    \node[circle,fill=black] at (0,0) (centre) {};
    \foreach \i in {1,...,#1}{
        \node[circle,fill=black] at ({\i*360/#1}:1) (node\i) {};
        \draw (centre)--(node\i);
    }
\end{tikzpicture}}
```

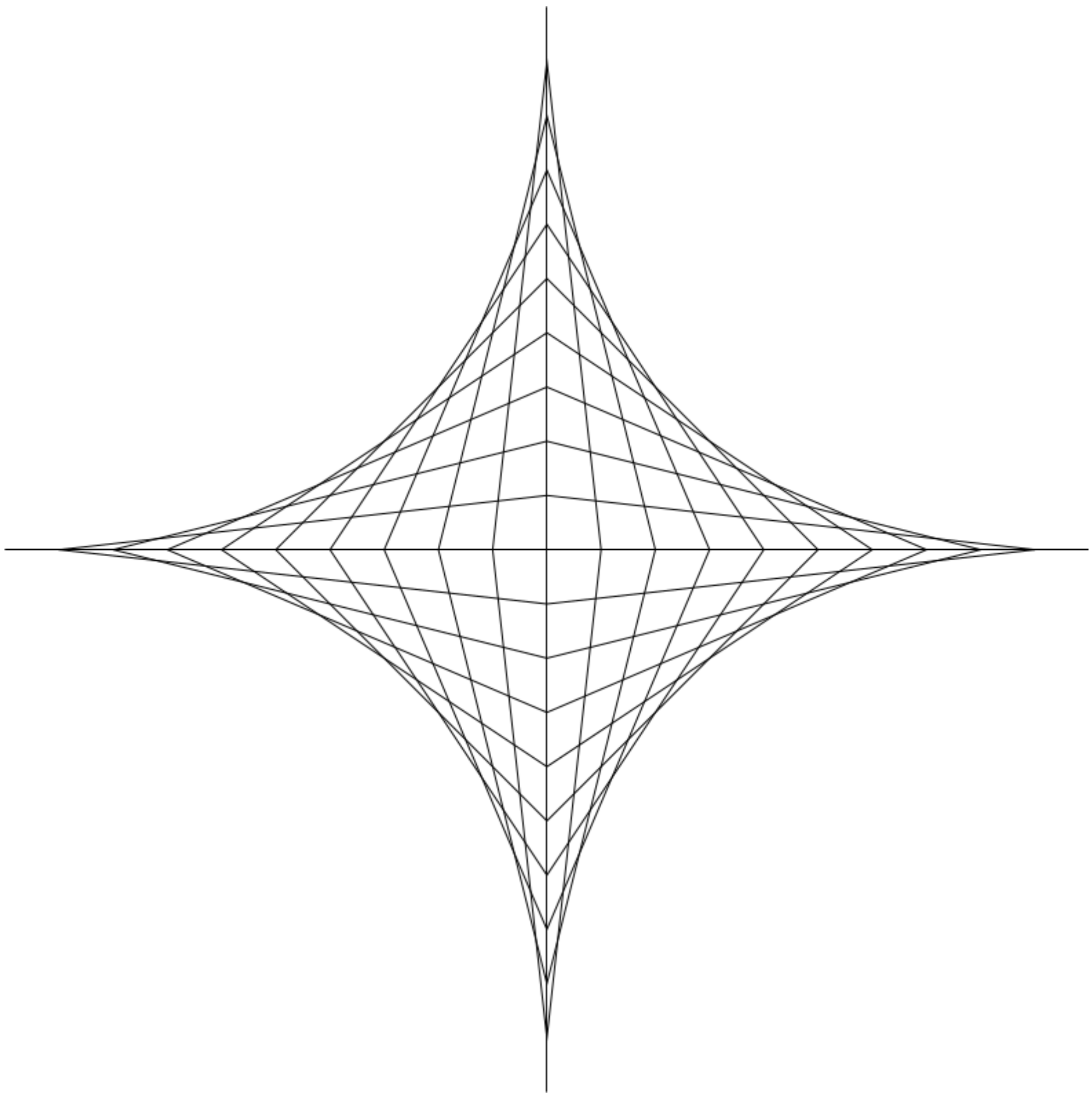
Compare this code to the example for five nodes above. Can you see how the two relate?

Then within the document itself (i.e. after `\begindocument`), you can draw star graphs using your newcommand `\stargraph{n}` where `n` is the number of nodes. Note there is no need to put `\begin{tikzpicture}` because this part is already included in `\stargraph`. Try it out!

```
\stargraph{3}

\stargraph{5}

\stargraph{7}

\stargraph{17}
```

Exercises

1. The following figure is obtained by connecting (0,1) to (10,0) and (0,2) to (9,0) and so on. Try drawing it by hand to get a feel for how it works. Then create it using a TikZ for loop.



1. By adjusting the step length on your code, make a version with more lines for a smoother curve.