Introduction to using the LaTeX TikZ package

Contents

1 Introduction

This is a brief introduction to using the LaTeX TikZ package to draw figures. The basic commands are described first, and the use of the OU styles described in Section 4.

More comprehensive descriptions include:

- Jacques Cremer's "A very minimal introduction to TikZ", (but not that minimal !)
  http://cremeronline.com/LaTeX/minimaltikz.pdf
- The full manual, available at, for example,

http://mirror.ox.ac.uk/sites/ctan.org/graphics/pgf/base/doc/generic/pgf/pgfmanual.pdf

2 Getting started

2.1 A minimal file

A very simple tikz file is as follows.

```
\documentclass[11pt]{article}
\usepackage{tikz}

\begin{document}

\begin{tikzpicture}
\draw (0,0) -- (1,1);
\end{tikzpicture}

\end{document}
```

Notes

- `\usepackage{tikz}` includes the standard tikz package. Replace this with `\usepackage{outikz}` to use the OU styles, as described in Section 4.
- Tikz drawing commands are contained within `\begin{tikzpicture}` and `\end{tikzpicture}`

  Figures can be scaled using optional arguments to `\begin{tikzpicture}`, for example:

  - `\begin{tikzpicture}[scale=2]` to make the figure twice the normal size
  - `\begin{tikzpicture}[xscale=2,yscale=3]` to scale differently in the horizontal (x) and vertical (y) directions,

  Line widths and text sizes are not affected by such scaling.

Important All tikz drawing commands end with a semi-colon (`;`). If this is forgotten, processing the file will hang, for ever unless interrupted!

2.2 Processing the file

TikZ code can be processed by both `pdflatex` (to produce a pdf file) and `latex` to produce a dvi file.

When developing figures, it is often conveient to use `pdflatex` and view the output in a pdf viewer that does not "lock" the file (which would then prevent `pdflatex` from updating the pdf when it is next run). Ghostview (`gsview`) allows the pdf to be updated, and automatcially refreshes to show the updated file.

To view a tikz dvi file using `yap` (which comes as part of MiKTeX), ensure that from the `View menu`, in the `Render method` submenu, `Dvips` is selected.

To produce a final eps of a figure for inclusion in a module unit, run `latex` to produce a `.dvi` file, then process this to an `eps` file. This `eps` file then needs to be further processed to remove any fonts embedded within the file (which may be replaced with different characters if the diagram is interactively edited by LTS) with drawings of the characters.

On a command line, the file `file.tex` would be processed with

```
latex file
dvips -E file.dvi -o tmp.eps
gswin32 -dNOPAUSE -dNOCACHE -dBATCH -sDEVICE=epswrite -sOutputFile=(file).eps tmp.eps
```

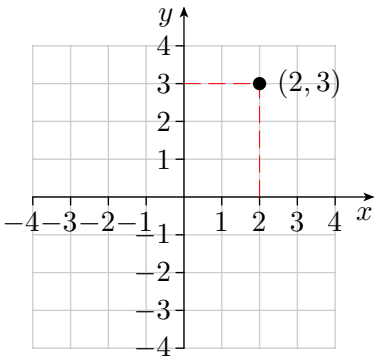
(If the option `epswrite` is not recognised, use `eps2write` instead.)

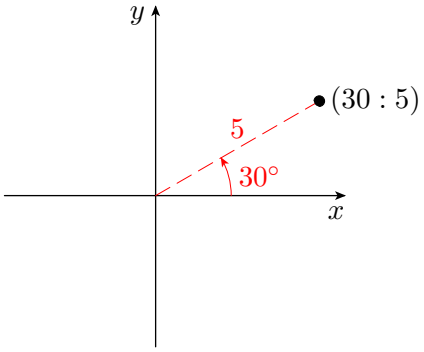
This final processing of figures to eps files could be done on a whole batch of files using a script.

3 Basic drawing

3.1 Coordinates

Points can be specfied using either Cartesian coordinates, eg `(2,3)`, or polar coordinates, eg `(30: 5)`.





To give an angle in radians, follow it by "`r`", for example `0.25*pi r`.

3.2 Drawing lines

Draw a line between two coordinates using: `\draw (1,2) -- (3,4);`

More than two points can be included, and a set of straight-line segments is drawn between them: `\draw (1,2) -- (3,4) -- (5,6);`

Ending a line with `-- cycle` joins the last point back to the first.

Incremental coordinates can be used. For example `\draw (1,2) -- ++(3,4);` starts at $(1, 2)$, moves 3 units in the x-direction and 4 units in the y-direction and joins the two points.

Line properties are set with optional arguments, for example

`\draw[red] (1,2) -- (3,4);` draws a red line.

Multiple options can be given, separated by commas.

3.3 Naming points

Points can be given a name and then used later. For example,

```
\coordinate (A) at (1,2);
```

defines `A` to be the point `(1,2)`. This can then be used to draw: `\draw (A) -- (3,4);`

Names can be assigned to points as a line is drawn:
`\draw (1,2) coordinate (A) -- (3,4) coordinate (B);`

Names can also be defined incrementally without drawing a line:
`\path (1,2) -- ++(30: 2) coordinate (C);`
defines C to be 2 units at an angle of 30^o from $(1, 2)$.

3.4 Drawing shapes

There are various commands to draw shapes, including:

- Rectangle, with the point A at the lower left corner, and B the upper right.

  `\draw (A) rectangle (B);`

- A grid of lines, again with A at the lower left corner and B the upper right.

  `\draw (A) grid (B);`

- Circle

  `\draw (A) circle (3);` Gives a circle, centre A, radius 3.

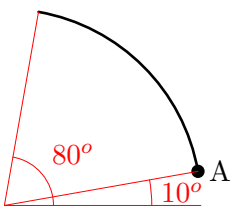
  `\node[draw, circle through=(B)] at (A) {};` This gives a circle centre A passing though B. This requires the use of the additional `through` tikz library which is included using the command `\usetikzlibrary{through}`.

- Arc, beginning at the point A and starting by making an angle of 10^o (relative to the positive x-axis), measured from the centre of the circle of which the arc is part, and ends when the angle subtended is 80^o. The radius of the arc is 2.

  `\draw (A) arc [start angle=10, end angle=80, radius=2];`

  Alternatively,

  `\draw (A) arc (10:80:2);`



The colour filling a closed object can be given using the `fill` option to `\draw`, for example `\draw[fill=blue]`.

3.5 Adding text

Text (or other LaTeX) can be added at a point on a diagram using the `node` command. For example,

```
\node at (3,4) {text};
```

or `\draw (3,4) node {text};`

Nodes can be included whilst drawing:

`\draw (0,0) -- (3,4) node{A};` draws a line from $(0, 0)$ to $(3, 4)$ and add the text "A" at the final point.

`\draw (0,0) -- (3,4) node {A} node[pos=0.5]{B};` draws a line from $(0, 0)$ to $(3, 4)$ and add the text "A" at the final point, and "B" at the point halfway along the line.

By default, the node text is centred on the point given. Alternatively, text can be placed near the point, rather than on it, by specifying the optional arguments
`left, right, above, below, below left` etc, for example

```
\node[above right] at (3,4) {text};
```

A colour for the text can also be given with the arguments of `node`.

3.6 Colours

As well as the usual colour names `red`, `green`, `blue` etc, shades of colours can be used: `red!20` is 20% of red. `red!20!blue` is 20% of red, the remaining 80% being of blue.

3.7 Plotting functions

A simple function $y = -x^2 + 4x + 8$ can be plotted using
`\draw[domain=-2:6, samples=100] plot (\x, {-\x*\x+4*\x+8});`

This plots the function over the domain $-2 \leq x \leq 6$, using 100 points.

The functions `sin`, `cos`, `sqrt` etc can be used in specifying the function to plot.

To restrict the line plotted to a certain region, for example, ensuring it does not go beyond the coordinate axis, a clipping region can be set. Since the clipping should only apply to the curve, it is contained in a `scope` block.

```
\begin{scope}
  \clip (-2, -1) rectangle (6, 5);
  \draw[domain=-2:6, samples=100] plot (\x, {-\x*\x+4*\x+8});
\end{scope}
```

***Note**: In practice, this can lead to the lines forming the boundary of the clipping window being seen, possibly due to rounding errors. It is better to restrict the domain of the function instead.*

Alternatively, TikZ can generate GNUPLOT code and then import the data produced by GNUPLOT. (To use this, you will obviously need GNUPLOT installed on your computer). For example

```
\draw[parametric, domain=-2:2, samples=50] plot[id=1] function{t,t*t};
```

plots the (trivial) parametric curve $(x, y) = (t, t^2)$ for $-2 \leq t \leq 2$ using 50 sampling points. A file `filename.1.gnuplot` (the name includes the `id` specified in the TikZ command, which could be text) is generated the first time LaTeX is run. Clicking on that file runs GNUPLOT which creates a `.table` data file. Running LaTeX again uses the `.table` file to create the plot. Some LaTeX systems might automate all this!

3.8 Calculating points.

In a complicated diagram, it is possible to calculate the position of points based on others. This requires the use of the `calc` tikz library which can be included using the command `\usetikzlibrary{calc}`. This is automatically included when using the OU styles.

- `($ (A) ! 0.25 ! (B) $)` is the point on the line AB, one quarter of the way from A to B.
- `($ (A) ! 2cm ! (B) $)` is the point on the line AB, 2 cm from A towards B. Note: distances are affected by figure scaling commands.
- `($ (A) ! 2cm !30:  (B) $)` start at A, move 2 cm towards B, then rotate 30^o about A.
- `($ (A) ! (C)! (B) $)` the projection of the point C onto the line AB. Note there is no space after `(C)`.

It is also possible to find the points of intersection of two objects, which requires the `intesections` library which can be included using `\usetikzlibrary{intersections}`.

For example

```
\draw[name path = mycircle] (0,1) circle (1);% Plot a circle, giving it the name mycircle
\draw[name path = myline]  (-1,-1) -- (2,2); % Plot a line, giving it the name myline
\path[name intersections={of=mycircle and myline}]; % Calculate the intersections

% Draw a red line between the intersection points
\draw[red] (intersection-1) -- (intersection-2);
```

4 OU tikz styles

To produce tikz figures to the OU styles, a LaTeX style has been developed: `outiks.sty`.

The use of the this style should be specified in the preamble of the LaTeX file, using `\usepackage{outikz}`, to replace `\usepackage{tikz}`.

The styles are listed and demonstrated in the file `outikz-styles-demo.tex` and `outikz-styles-demo.pdf`.

4.1 Installing `outikz.sty`

The style file is available from the GitHub respository:
https://github.com/rbrignall/OU-SUPPS.

To use the styles, there are two possible options

1. Place a copy of `outikz.sty` in the folder you are working in, with your figure LaTeXfiles. This is the simplest approach, but will need a copy of `outikz.sty` in each folder you work in.

2. Add `outikz.sty` to your LaTeX installation. Assuming you keep your local installation files in `C:\Local_TeX_Files`, copy `outikz.sty` to `C:\Local_TeX_Files\tex\latex\outikz\outikz.sty`, creating subfolders as needed.

   If you did not initially have a local files location, such as `C:\Local_TeX_Files` above, you need to tell LaTeX to look there. Assuming you are using MikTeX, open `Start->MikTex 2.9 ->Maintenance (Admin) -> Settings (Admin)`, select the `Roots` tab and add `C:\Local_TeX_Files` (or what-ever you called it) as a root path.

   In any case, you then need to refresh the LaTeX internal database. Assuming you are using MikTeX, open
   `Start->MikTex 2.9 ->Maintenance (Admin) -> Settings (Admin)`, and on the `General` tab press `refresh FNDB`.

4.2 Using `outikz.sty` for the first time

The first time you process a LaTeX file using `outikz.sty` you may need to install some (standard) packages that it uses.

If you are using a recent version of MiKTeX, the system should download and install all the required packages when you first process a file using them. If you are within the OU firewall, you need to ensure your proxy settings are correct (`wwwcache.open.ac.uk`, port 80) to permit this. These settings can be changed at:

`Start->MikTex 2.9 ->Maintenance (Admin) -> Update (Admin)` and click `connection settings`.

5 OU figure guidelines

When producing OU figures, please bear in mind the following.

- Text on diagrams should be 11pt Computer Modern. (Hence including the `[11pt]` option to `\documentclass`.)
- Make sure you use the correct colour for the correct module. The available colours are defined towards the start of `outikz.sty`.

  For ease of use, you might like to rename, for example `MST124red`, to be `red` by include the line `\colorlet{red}{MST124red}` in your figure file.
- Generally, the predefined colours should be used with the following priority: black, blue, red, green, purple.
- When a graph contains one curve/line it should be black. When a graph contains two curves/lines of equal importance, different colours can be used (eg, blue, red).
- Points and coordinates marked on a graph should be in the same colour as the line itself.
- When using (coloured) construction lines, and a main graph or curve, it is usually best to draw the main graph last, so it hides the ends of the construction lines.
- When shading an area, use 20% of one of the standard colours. Eg, to shade a rectangle green:
  `\draw[fill=green!20] (0,0) rectangle (2,2);`