

PGFPLOTS Intro and Tutorial

W. Ethan Duckworth ©2026

This “Intro and Tutorial” is focused on how to create graphs of the sort you see in a Calculus textbook. It is relatively brief and leaves out many options: Everything here, and much much more, can be found in the [pgfplots manual](#).

Contents

1	Example: A parabola and tangent line	3
2	Tikz commands, options, and keys	9
3	Creating your own settings	12
4	Sizing the axes	14
4.1	External versus internal lengths and coordinates	14
4.2	Setting external size using built-in styles	15
4.3	Setting external size manually	17
4.4	Setting external aspect ratio	19
4.5	Setting internal aspect ratio	21
5	Axis lines	23
6	Marking points and piecewise functions	24
6.1	Basic Piecewise graph	24
6.2	Marking a single point at a time	25
6.3	Marking multiple points	26
6.4	Intervals on the number line	27
7	Asymptotes	29
7.1	Basic approach	29
7.2	Unexpected wiggles with smooth	30
7.3	Fine tuning ymin etc.	31
8	Parametric	32
8.1	Basic use of parametric	32
8.2	Avoiding problematic Cartesian calculations	32
9	Math library	34
9.1	Math expressions in coordinates and domains	34
9.2	Declaring functions	35
9.2.1	Quadratic and tangent line	35
9.2.2	A piecewise graph	36
9.2.3	Delta-Epsilon pictures	37
10	Pushing the envelope in speed, efficiency and precision	39
10.1	Smooth is faster	39
10.2	Graphs with a large number of points	44
10.3	Numerical Precision	47

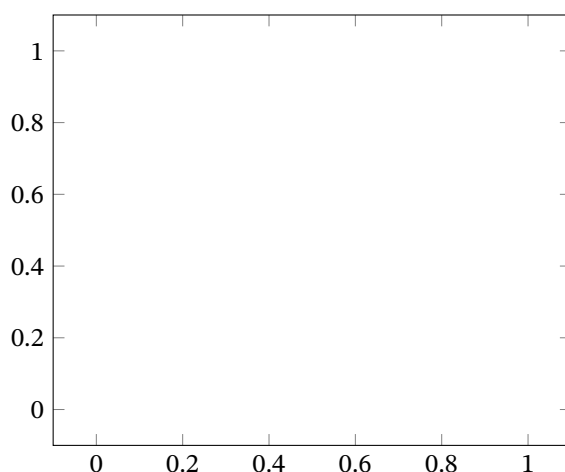
11 Custom Macros	48
12 Breaking up and externalizing	50
12.1 Breaking up	50
12.2 Externalizing	51
References and Further Resources	52
One page cheat-sheet	53

1 Example: A parabola and tangent line

Suppose you want to make a graph for your Calculus class of $f(x) = x^2$. You’ve heard that a package called `pgfplots` can do this easily with an environment called `axis` that is used within `tikzpicture`.¹ You give it a dry run without any functions added

```
% in the preamble
\usepackage{pgfplots}
\pgfplotsset{compat=1.18}

% in the regular document
\begin{tikzpicture}
\begin{axis}
\end{axis}
\end{tikzpicture}
```



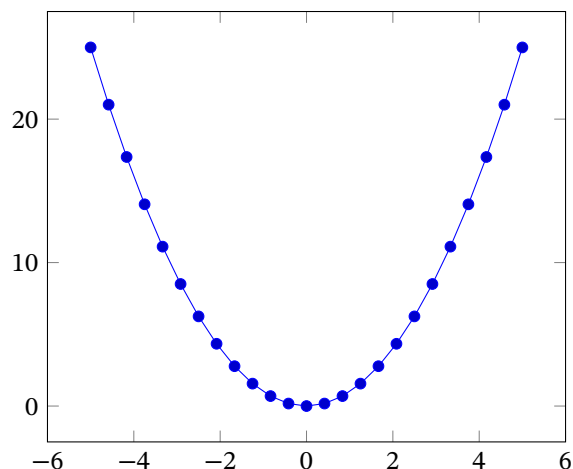
Not bad for an empty example,² but of course you want more. You look briefly at the manual and see that the command `addplot` can add plots within the axis.³

```
\begin{tikzpicture}
\begin{axis}
\addplot{x^2};
\end{axis}
\end{tikzpicture}
```

¹The package called `tikz` is a general purpose package for creating graphics in a user-friendly way. In my mind Tikz is pronounced as “ticks” like the tick marks you put on a graph. In the background it uses a programming/calculation layer called PGF. PGF stands for “portable graphics format” and it is a more detailed and complicated mathematical-graphical language that executes the user-friendly drawing commands in Tikz. Meanwhile, `pgfplots` consists of user-friendly commands that operate inside of Tikz, but in the background does a lot of calculations in the PGF layer.

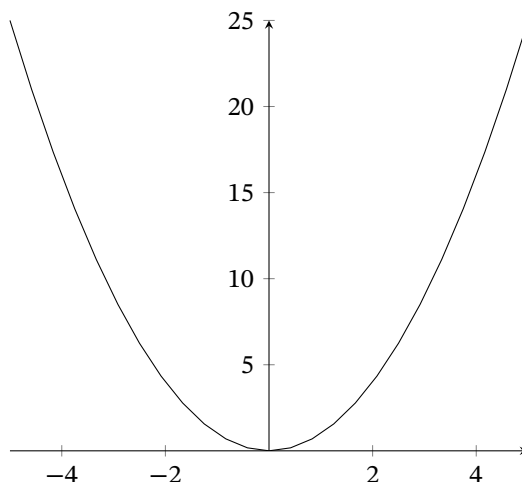
²Hopefully it only took you a couple of minutes. My first dry run of LaTeX (in 1995) took me close to an hour!

³The `pgfplots` manual uses the word “axis”, singular, to refer to the lines and labels that make up the figure above.



How nice that you didn't have to specify the values to plot, how big the graph should be, etc. On the other hand, maybe it doesn't look the way you want. You look in the manual and find some keys to change the appearance, such as `axis lines` and `no marks`. These keys are passed as options to `axis` and `addplot` respectively.

```
\begin{tikzpicture}
\begin{axis}[axis lines=middle]
\addplot[no marks]{x^2};
\end{axis}
\end{tikzpicture}
```



The `no marks` key had more of an impact than you wanted, you wanted to just remove the marks and keep the blue line. The previous use of `\addplots{x^2}` used the default pgfplots style.⁴ If we include `[` and `]` to `addplot` this turns off the default style options, leaving just a thin black line (the ultimate default). So we didn't need to say `no marks`; we could have said `addplot[]` and this would have turned the marks off and made a plot identical to the one above.

I would recommend using (1) a color for most plots, (2) a `thick` line, and (3) the `smooth` option for any plot that is curved (see a later section for more details about `smooth` and its benefits).

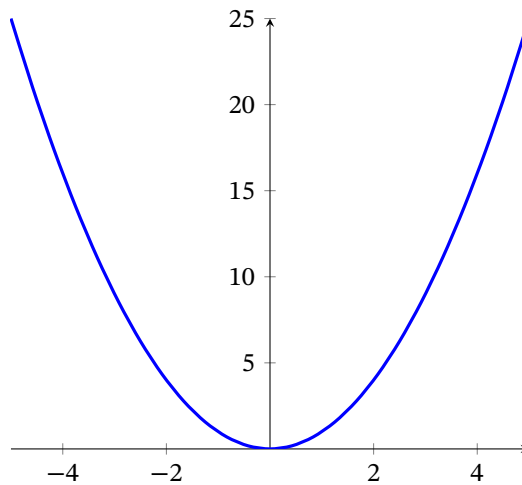
```
\begin{tikzpicture}
\begin{axis}[axis lines=middle]
```

⁴Actually the first `addplot` command without any options will use the first entry in a list of default styles, the second `addplot` command uses the second entry in this list, etc. For example, using defaults, the first plot will be blue with filled circle markers, the second one red with square markers, the third brown with filled \otimes , the fourth black with asterisks, etc.

```

\addplot[smooth, very thick, blue]{x^2};
\end{axis}
\end{tikzpicture}

```

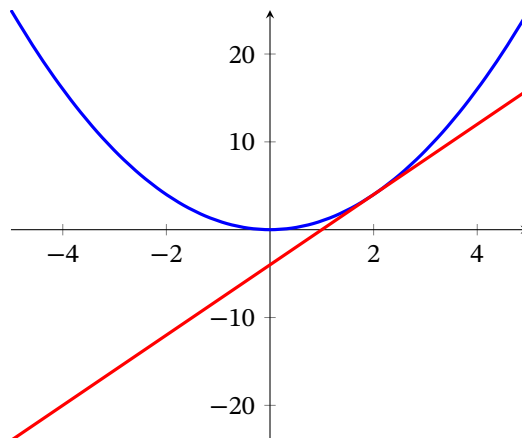


Next, we want to add a tangent line, and use colors to help distinguish this from the curve. We will calculate this ourselves and add it.

```

\begin{tikzpicture}
\begin{axis}[axis lines=middle]
\addplot[very thick,blue,smooth]{x^2};
\addplot[very thick,red]{4*(x-2)+4};
\end{axis}
\end{tikzpicture}

```



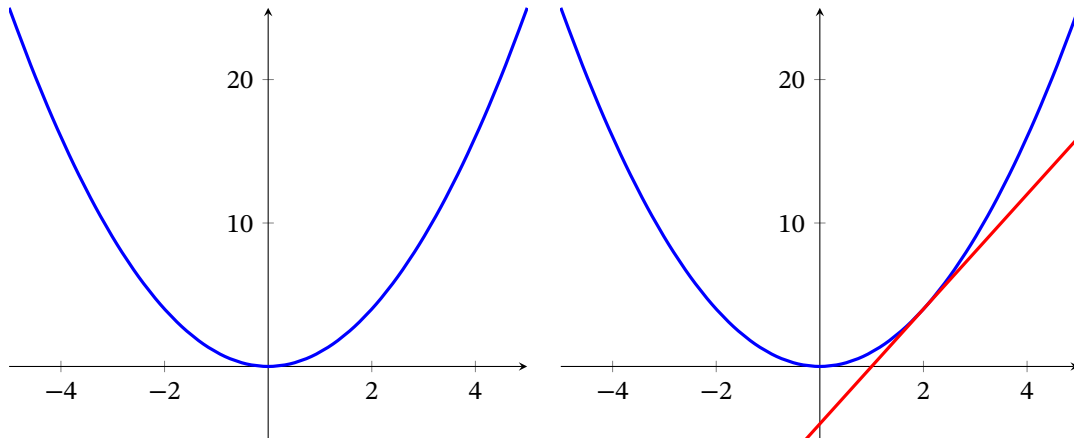
This is pretty good, but maybe we want the axis to have the same range of numbers as the one with just x^2 so that when we see them side by side it's easier to focus on the new feature, i.e. the tangent line. We need to either make the axis for the x^2 picture go down farther, or the axis with the tangent line not go down as far, or a little bit of both. We will manually set `ymin` instead of having pgfplots figure out what it should be (there are also keys for `xmin`, `xmax`, and `ymax`: my general philosophy is to leave these settings to the default or automatic mode until I need to change them).

```

\begin{tikzpicture}
\begin{axis}[ymin=-5,axis lines=middle]
\addplot[very thick,blue,smooth]{x^2};
\end{axis}
\end{tikzpicture}

\begin{tikzpicture}
\begin{axis}[ymin=-5,axis lines=middle]
\addplot[very thick,blue,smooth]{x^2};
\addplot[very thick,red]{4*(x-2)+4};
\end{axis}
\end{tikzpicture}

```

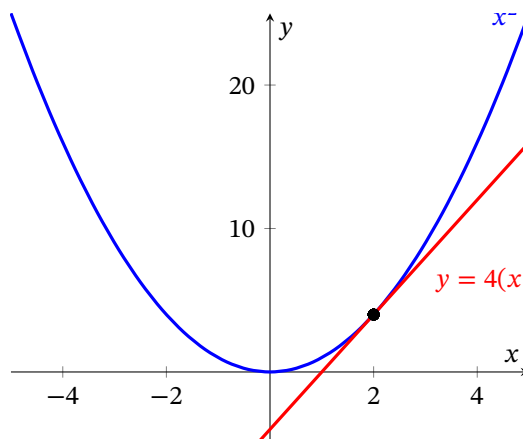


This is nice, and maybe you're fine stopping here. But perhaps you want to label the axes and the curves. We'll do the former with `xlabel` and `ylabel`, and the latter by adding text in nodes⁵ at the end of the paths created by `addplot`. Using the key `pos=0.8` means "add this node 80% of the way along the path created by `addplot`"

```

\begin{tikzpicture}
\begin{axis}[ymin=-5,axis lines=middle,
xlabel={$x$},ylabel={$y$}]
\addplot[very thick,blue,smooth]{x^2} node[left]{$x^2$};
\addplot[very thick,red]{4*(x-2)+4}
node[pos=0.8,below right]{$y=4(x-2)+4$};
\addplot[only marks] (2,4);
\end{axis}
\end{tikzpicture}

```



This partly worked, but the labels for the curves got cut off. You could say they were "clipped". By default pgfplots clips⁶ the final picture to the size of the axes: i.e. everything between `xmin`, `xmax`, `ymin` and `ymax`. Although

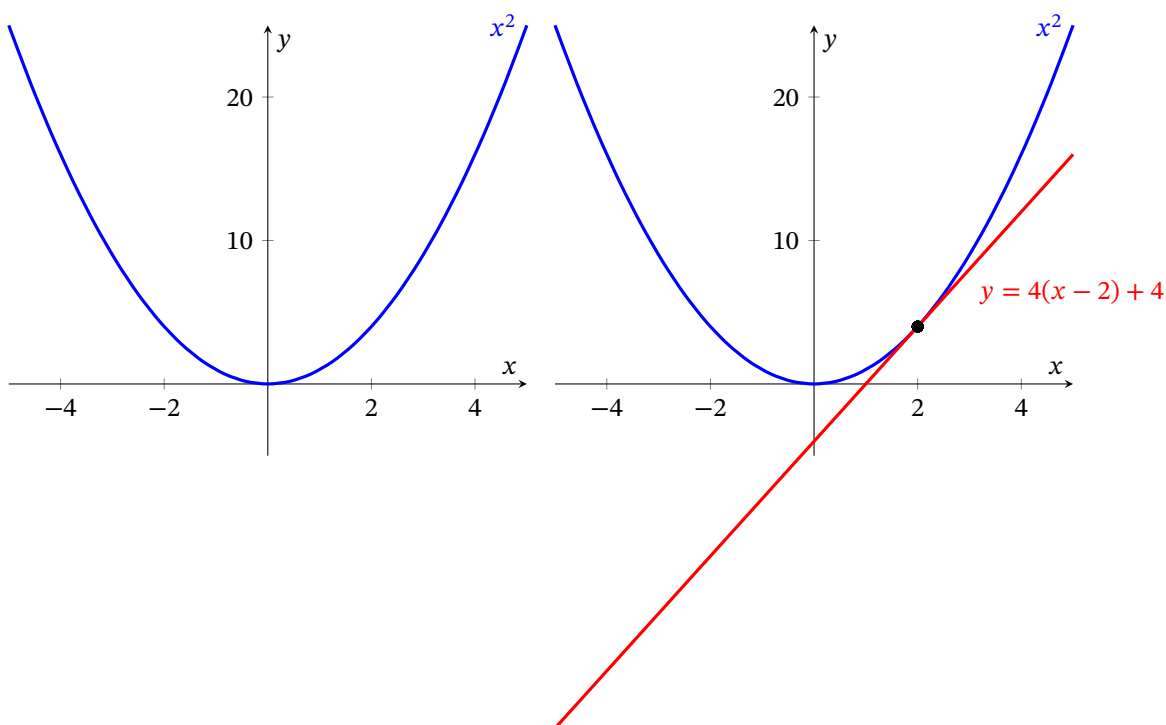
⁵The `node` command is not part of pgfplots, but part of Tikz/PGF. Basically it means "add something like a label or box to the path you are drawing." See the Tikz/PGF manual for more information.

⁶"Clip" means everything that is not in a certain part of the picture is cut off

pgfplots is smart enough to adjust `ymin` and `ymax` to fit the plots, it doesn't do so for the labels, and every approach I've found will sometimes require at least one additional, manual step. Sometimes, all you have to do is turn off the clip behavior by setting `clip=false` on the axis. This works perfectly for the parabola alone, but not perfectly for the parabola and tangent line:

```
\begin{tikzpicture}[baseline=0cm]
\begin{axis}[ymin=-5,axis lines=middle,
xlabel={\mathit{x}},ylabel={\mathit{y}},clip=false]
\addplot[very thick,blue,smooth]{x^2}
node[left]{\mathit{x}^2};
\end{axis}
\end{tikzpicture}

\begin{tikzpicture}[baseline=0cm]
\begin{axis}[ymin=-5,axis lines=middle,
xlabel={\mathit{x}},ylabel={\mathit{y}},clip=false]
\addplot[very thick,blue,smooth]{x^2}
node[left]{\mathit{x}^2};
\addplot[very thick,red]{4*(x-2)+4}
node[pos=0.8,below right]{\mathit{y}=4(\mathit{x}-2)+4};
\addplot[only marks] (2,4);
\end{axis}
\end{tikzpicture}
```



The problem here is that we decided we *didn't* want that much of the tangent line below the x -axis to show up. Earlier we fixed this by setting `ymin=-5` which clipped off anything in the graph below that, but now are telling it not to clip that off.

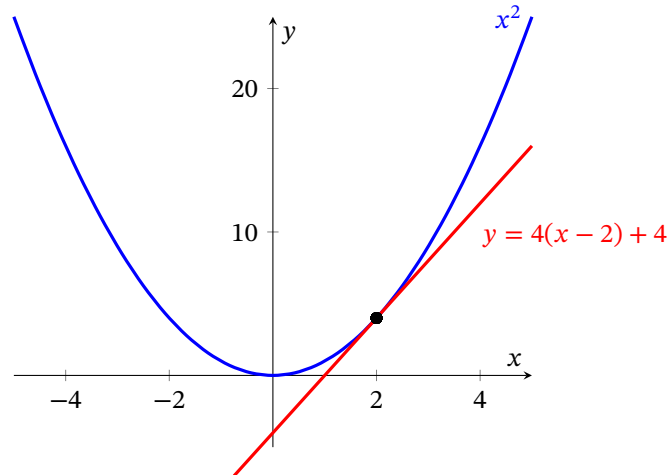
As I said above, I often have to make some manual changes to get labels to work, and I didn't just mean setting `clip = false`, although sometimes that's all that's needed. There are three additional steps I would consider, each of which can always be made to work, though which is easier may depend on the circumstance: (1) setting `clip = false`, and manually adjusting the domain of each function to make it fit, (2) setting `clip = false`, but using a key called `restrict y to domain` to eliminate y -values that don't fit (this is kind of like using two kinds of clips), (3) setting `clip mode = individual`, which makes clips just apply to `\addplot` paths, and then add the labels in a separate step from the `\addplot` path (again this is like using a different kind of clip).

```
% manually adjust domain for the line
\begin{tikzpicture}
\begin{axis}[ymin=-5,axis lines=middle,
xlabel={\mathit{x}},ylabel={\mathit{y}},clip=false]
\addplot[very thick,blue,smooth]{x^2} node[left]{\mathit{x}^2};
\addplot[very thick,red,domain=-0.75:5]{4*(x-2)+4}
```

```

node[pos=0.8,below right]{$y=4(x-2)+4$};
\addplot[only marks] (2,4);
\end{axis}
\end{tikzpicture}%

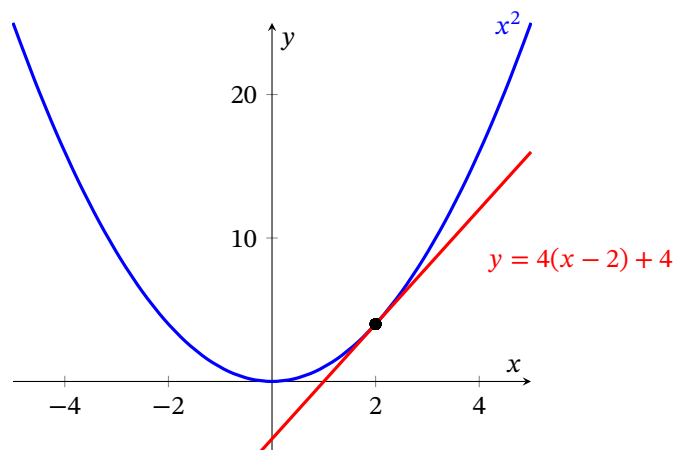
```



```

% clip only the addplot paths, but then add the labels separately
\begin{tikzpicture}
\begin{axis}[ymin=-5,axis lines=middle,
xlabel={$x$},ylabel={$y$},clip mode = individual]
\addplot[very thick,blue,smooth]{x^2}; % this will get clipped
\addplot[very thick,red]{4*(x-2)+4}; % this will get clipped
\addplot[only marks] (2,4);
\draw[blue] (5,25) node[left]{$x^2$}; % not clipped
\draw[red] (4,10) node[below right]{$y=4(x-2)+4$}; % not clipped
\end{axis}
\end{tikzpicture}%

```



```

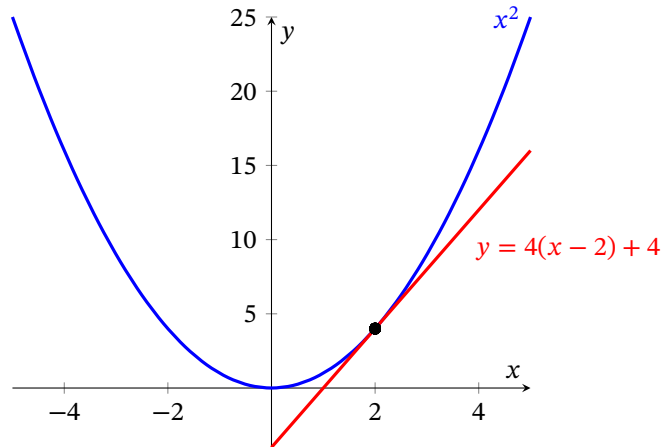
% restrict y to domain
\begin{tikzpicture}
\begin{axis}[restrict y to domain = -5:25,
axis lines=middle,

```

```

xlabel={x$},ylabel={y$},clip=false]
\addplot[very thick,blue,smooth]{x^2} node[left]{$x^2$};
\addplot[very thick,red]{4*(x-2)+4}
    node[pos=0.75,below right]{$y=4(x-2)+4$};
\addplot[only marks] (2,4);
\end{axis}
\end{tikzpicture}

```



Regarding these three approaches I don't have a general recommendation as to when to use which, but I do have some thoughts. I like being able to add `node` to the end of the `\addplot` path because then the label picks up the color used by `\addplot` and also because its position is directly connected to that plot: if I change the function I'm using, say shifting it up, the label will automatically change too. So usually I start with this. Sometimes the label will be clipped and I simply adjust the `pos=0.8` setting to move it so it falls within the axis.

If this doesn't work, probably my next bet will be to use `clip = false` combined with `restrict y to domain`, though this may mean I have to decide what the best range of y-values is (rather than something automatically being done).

Finally, you will probably sometimes have to adjust domains, maybe not to fit within a given axis, but to fit the rest of the picture. But I always do this last because it involves guessing and checking (which means waiting for a latex typeset run).

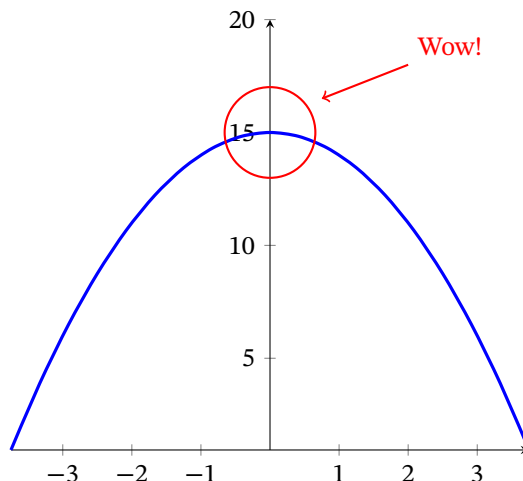
2 Tikz commands, options, and keys

`pgfplots` is built on top of Tikz, which has hundreds of commands for drawing things, and you can combine Tikz commands with `pgfplots` commands.

```

\begin{tikzpicture}
\begin{axis}[
    axis lines=middle,
    clip=false,ymax=20,
    restrict y to domain=0:20
]
\addplot[smooth,very thick,blue]{-x^2+15};
\draw[red,thick] (0,15) circle (0.6cm);
\draw[red,thick,->] (2,18) node[above right]{Wow!} -- (0.75,16.5);
\end{axis}
\end{tikzpicture}

```

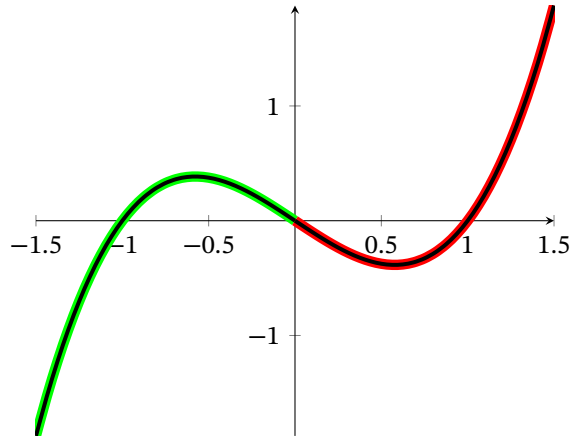


You can include any Tikz commands within the `axis` environment, and the coordinates used in that command will equal the numbers that `axis` is using (provided you are using a new enough version of `pgfplots` where any version since 1.11, released in 2014, is new enough). In the example above, the circle is centered at the *logical coordinate* (also called “axis coordinate”) (0, 15) where the 0 and 15 equal the numbers shown in the axes. Perhaps it seems obvious that it would work this way, but note: (1) There is a lot of scaling going on to turn the 15 on the y-axis above into an actual distance on the paper that is closer to 5cm. (2) The numbers do *not* match up that way if you put that same `\draw` command outside of `axis`. (3) The numbers do *not* match up that way if you put “cm” after the coordinates (which is why we *did* put “cm” in the radius for the circle; this is a radius of a “real” 0.5 centimeters, not a radius of 0.5 in the coordinate system used in the axes.) See the section on sizing axes for more about this distinction.

In addition to adding Tikz commands inside of `axis` you can add Tikz options/keys to the `\addplot` command. The following might be useful keys (some we’ve seen already)

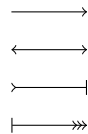
- colors, for example `blue`, `red`, etc. You can enter these directly or as `color = blue`, etc.
- line thicknesses: `ultra thin`, `very thin`, `thin`, `semithick`, `thick`, `very thick`, `ultra thick`, `line width = <dimension>`
- line styles: `dashed`, `dotted` (both also have `loosely` and `densely` varieties, like `densely dashed`). You can also create a double line with `double`

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle]
\addplot[very thick, double =black, double distance = 1.5pt,
smooth, green, domain=-1.5:0]{x^3-x};
\addplot[very thick, double =black, double distance = 1.5pt,
smooth, red,domain=0:1.5]{x^3-x};
\end{axis}
\end{tikzpicture}
```



- Arrow tips and tails: `<`, `>` and `|` can be put at either end, and also stacked up:

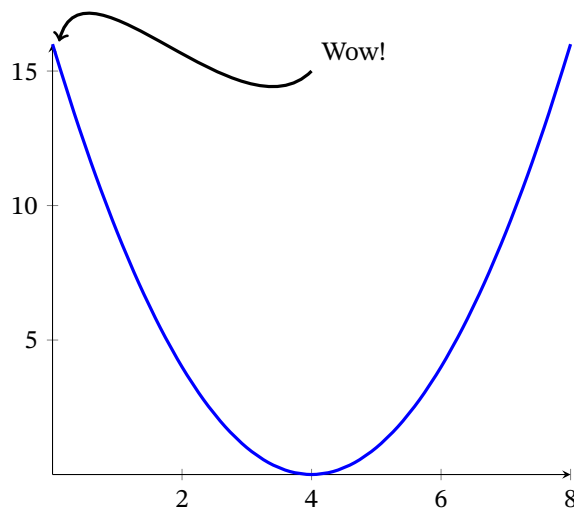
```
\begin{tikzpicture}[yscale=0.5]
\draw[->] (0,0) -- (1,0);
\draw[<->] (0,-1) -- (1,-1);
\draw[>-|] (0,-2) -- (1,-2);
\draw[|->>>] (0,-3) -- (1,-3);
\end{tikzpicture}
```



Note that there are a *huge* number of additional arrow tips and tails, but for many of these you'll need to load the `arrows.meta` package in Tikz.

- Bends:

```
\begin{tikzpicture}
\begin{axis}[axis lines=middle, clip = false]
\addplot[very thick,smooth,blue,domain=0:8]{(x-4)^2};
\draw[->,out = 225,in=75,very thick]
(4,15) node[above right]{Wow!} to (0.1,16.1);
\end{axis}
\end{tikzpicture}
```



Here `out` refers to what angle the path will make “coming out” of a point (in this case coming out of the point next to “Wow!”), `in` refers to what angle the path will make “coming in” to a point (in this case the point (0, 15)), and `to` replaces the more common `--` so instead of a straight line (which `--` makes), the path “goes to” the given point.

- plot expression options: Although we use the `\addplot` command, which is part of the `pgfplots` package, the main thing it adds compared to plain Tikz is automatic scaling of the final picture, automatic options for the lines that show the axes, and automatic options for tick marks, tick labels, labels, etc. In particular, most of the options we pass to `\addplot` are from Tikz, including the following
 - `domain=5:10` will plot a formula using 25 x -values starting at 5 and finishing at 10.
 - `samples = 100` will change the number of x -values used in the domain to 100.
 - `samples at = {5,7,9,15}` will cause the function formula to be evaluated at $x = 5$, $x = 7$, $x = 9$ and $x = 15$. It overrides the `domain` and `samples` options.
 - `coordinates = {(1,2) (3,4) (10,15)}` will make a plot through the points (1, 2), (3, 4) and (10, 15).
 - `only marks` means the plot will not connect points with a line or curve, it will only make marks at each point.
 - `smooth` means the plot will make a smooth curve through the points (where the points are either given in a coordinate list, or by sampling a function).

3 Creating your own settings

I tend to want the same, or similar, settings on most plots, and `pgfplots` makes it easy to create and use these. From the manual “Use `\pgfplotsset{⟨key⟩/.style = {⟨key-value-list⟩}}` to (re)define a style `⟨key⟩` in the namespace `/pgfplots.`”

What this means is you can do something like this:

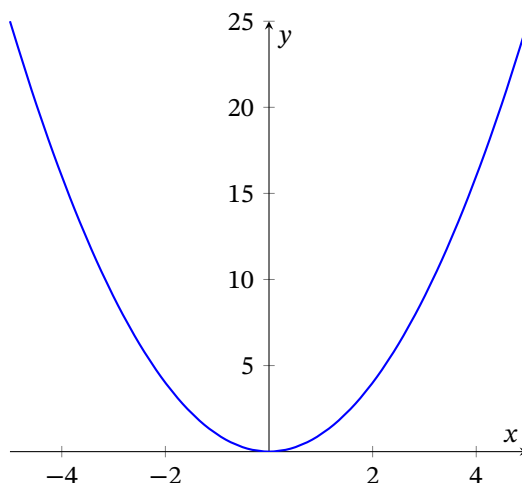
```
% probably in preamble
\pgfplotsset{
  duckplot/.style={thick,blue,smooth},
  duckaxis/.style={axis lines = middle, xlabel={\$x\$}, ylabel={\$y\$}}
}
% in main document
```

```

\begin{tikzpicture}
\begin{axis}[duckaxis]
\addplot[duckplot]{x^2};
\end{axis}
\end{tikzpicture}

```

and then use that in plots



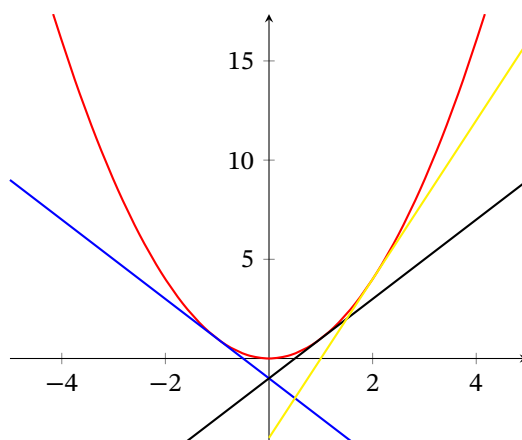
You can also change the default settings so you don't need to use options `duckaxis` and `duckplot` in the above commands.

```

% probably in preamble
\pgfplotsset{
  every axis/.style = {axis lines = middle},
  cycle list name = color list, % internal list of colors
  every axis plot/.style = {thick,smooth}
}

% in main document
\begin{tikzpicture}
\begin{axis}[restrict y to domain = -5:20]
\addplot{x^2};
\addplot{-2*(x+1)+1};
\addplot{2*(x-1)+1};
\addplot{4*(x-2)+4};
\end{axis}
\end{tikzpicture}

```



Making your own styles is an excellent idea: it will make your code easier to read and produce more consistent results (e.g. all your graphs will be blue and very thick without you forgetting one). None the less, I have not used them in this document at all, because I want each example to be as easy for a new user to figure out, without having to puzzle over what `duckplot` means, etc.

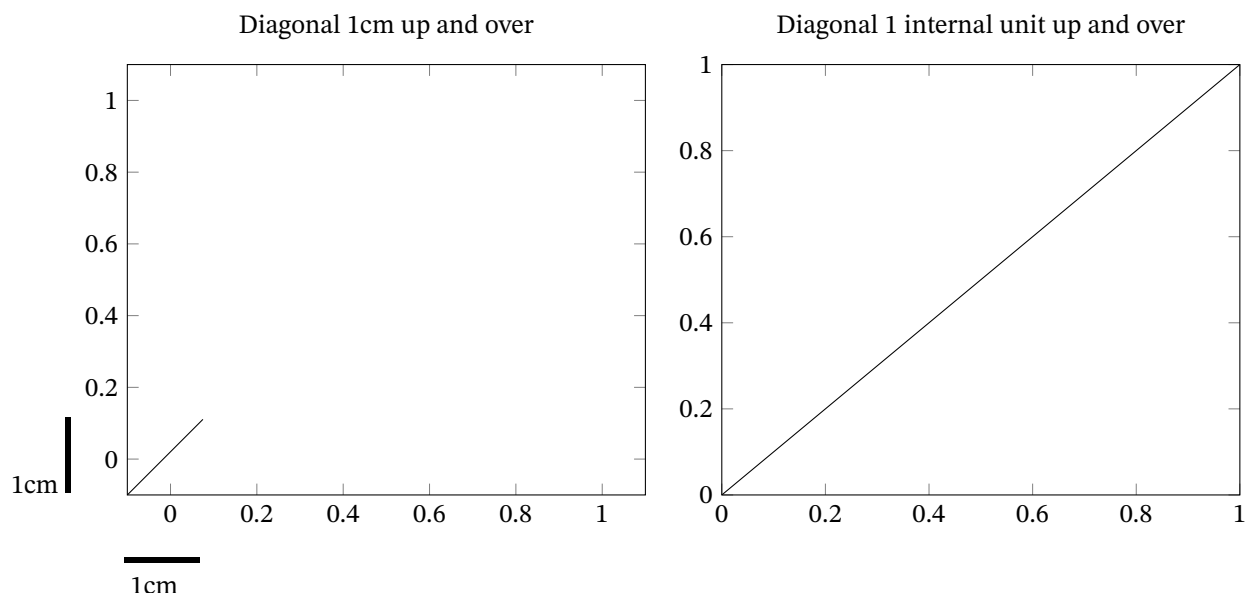
4 Sizing the axes

4.1 External versus internal lengths and coordinates

As mentioned above, the origin and coordinates within `axis` will in general not be the same as the coordinates and origin within `tikzpicture` but outside `axis`. This is inevitable if we expect coordinates in `tikzpicture` to be based in 1 centimeter units, but coordinates within `axis` to match the numbers displayed in the axes. The next example illustrates this difference:

```
\begin{tikzpicture}
\begin{axis}[title={Diagonal 1cm up
and over}]
\end{axis}
\draw (0,0) -- (1,1);
\end{tikzpicture}
```

```
\begin{tikzpicture}
\begin{axis}[xmin = 0, xmax=1, ymin=0, ymax=1,
title={Diagonal 1 internal unit up and over}]
\draw (0,0) -- (1,1);
\end{axis}
\end{tikzpicture}
```



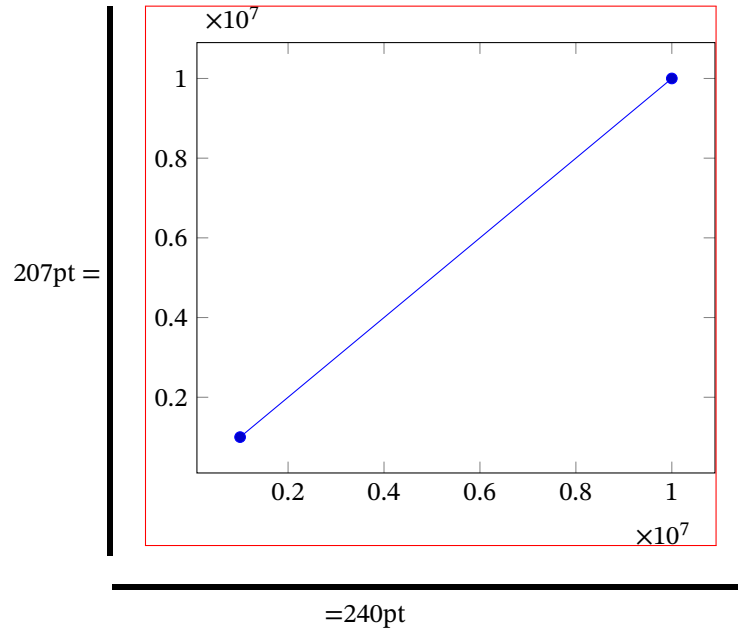
We’ve added two markers, created with `\rule` outside of the `tikzpicture` so you can see what a real centimeter is. Between `axis` and `tikzpicture` (i.e. outside `axis` but inside `tikzpicture`) a coordinate of 1 means 1cm from the origin, where 1cm is what we would get if we printed it out and put a ruler on the result (assuming we have not used a `scale` option, or similar change, in `tikzpicture`). But inside `axis` a coordinate of 1 uses the numbers shown in the coordinate axes, and therefore a length of 1 is also measured using these coordinates. When we need to distinguish between these two ways of measuring lengths we’ll call the former, “external” and the latter “internal”. In other words, the “external width” of a figure is what we could get by measuring it with a ruler, or what TeX will see in terms of a box on the page, the lengths marked by `\rules` above, and the “internal width” is what we would get using the numbers on the axes. External dimensions can be set manually directly using the `width` and `height` keys, and internal size can be manually directly set by using `xmin`, `xmax`, `ymin` and `ymax`.

The figure below has an external size of approximately 240pt × 207pt (that’s width × height) and internal size of approximately 10,000,000 × 10,000,000.

```

\begin{tikzpicture}
\begin{axis}[tick scale binop={\times}]
% pgfplots knows 1e6 notation for 10^6
\addplot coordinates {(1e6,1e6) (1e7,1e7)};
\end{axis}
\end{tikzpicture}

```



Note: according to the `pgfplots` manual the external dimensions are *approximately* 240pt \times 207pt (the external box is shown in red, and the measurements of 240pt and 207pt are marked with `\rule` commands that are not part of or inside `tikzpicture`). “Approximately” means that the actual size uses the following calculation: external size of the black axis box (just the box itself not the numbers) is $240 - 45 \times 207 - 45$ where 45pt is a default guess for the size of the labels, titles and ticks. Thus the red box has size $240 - 45 + \text{dec}_x \times 207 - 45 + \text{dec}_y$ where dec_x and dec_y are the actual sizes of the labels, titles and ticks (i.e. the “decorations”).

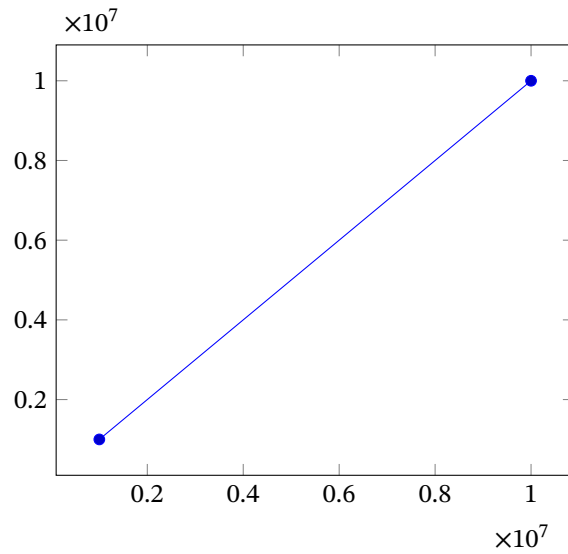
4.2 Setting external size using built-in styles

To change the external size of plots it’s recommend you use (or start with) one of the following:

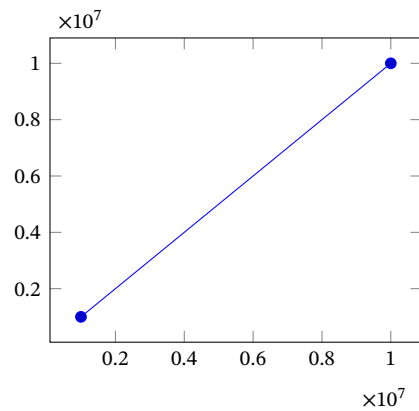
```

% normalsize =
% 240pt x 207cm = 8.435cm x 7.2752cm = 3.321in x 2.864in
\begin{tikzpicture}
\begin{axis}[tick scale binop={\times}]
\addplot coordinates {(1e6,1e6) (1e7,1e7)};
\end{axis}
\end{tikzpicture}

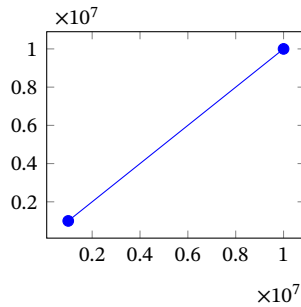
```



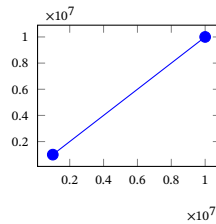
```
% small =
% = 6.50cm x 5.60cm = 2.52in x 2.20in
\begin{tikzpicture}
\begin{axis}[small,tick scale binop={\times}]
\addplot coordinates {(1e6,1e6) (1e7,1e7)};
\end{axis}
\end{tikzpicture}
```



```
% footnotesize =
% = 5cm x 4.31cm = 1.97in x 1.70in
\begin{tikzpicture}
\begin{axis}[footnotesize,tick scale binop={\times}]
\addplot coordinates {(1e6,1e6) (1e7,1e7)};
\end{axis}
\end{tikzpicture}
```



```
% tiny =
% = 4cm x 3.45cm = 1.57in x 1.36in
\begin{tikzpicture}
\begin{axis}[tiny,tick scale binop={\times}]
\addplot coordinates {(1e6,1e6) (1e7,1e7)};
\end{axis}
\end{tikzpicture}
```



Note that in each case the internal dimensions are the same, but the external dimensions are getting smaller. Note too that it's not just the external dimensions that get smaller: various style settings are adjusted so that the tick marks, and tick labels, all get smaller. The result should be more legible and better appearance than if you started with `normalsize` and then simply set `width` and `height` directly to something like 4cm and 3.45cm.

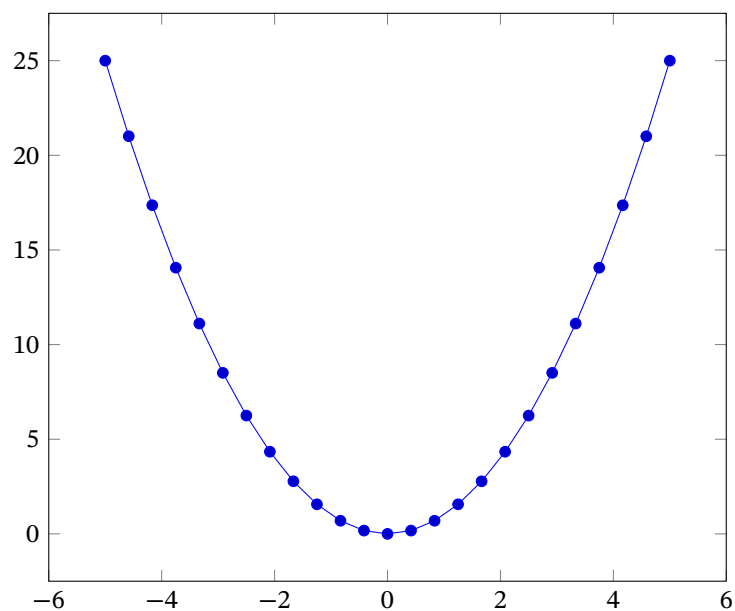
In each case, if you want to tweak the size, it's recommended you first set it to the closest option out of `normalsize`, `small`, `footnotesize` and `tiny` and then set `width` manually, and then `pgfplots` will auto-scale `height` so that the aspect ratio is the same as the `normalsize`, i.e. the aspect ratio equals $240/207 \approx 1.16$. For instance, `small, width = 5.75cm`. You can set both `width` and `height` in which case you get whatever aspect ratio you have set. Apparently the package has no direct provision for making *larger* plots.

4.3 Setting external size manually

So it remains to show you how to make things bigger than `normalsize` and how to change the aspect ratio. Here are three ways to make a bigger plot, specifically one that is close to 25% larger: (1) One we manually set `width` and `height`, (2) We scale the plot logically (meaning all points get farther apart, but letters and lines don't magnify), (3) we scale it visually (meaning it's like we put it under a magnifying glass, making letters and numbers bigger and lines thicker)

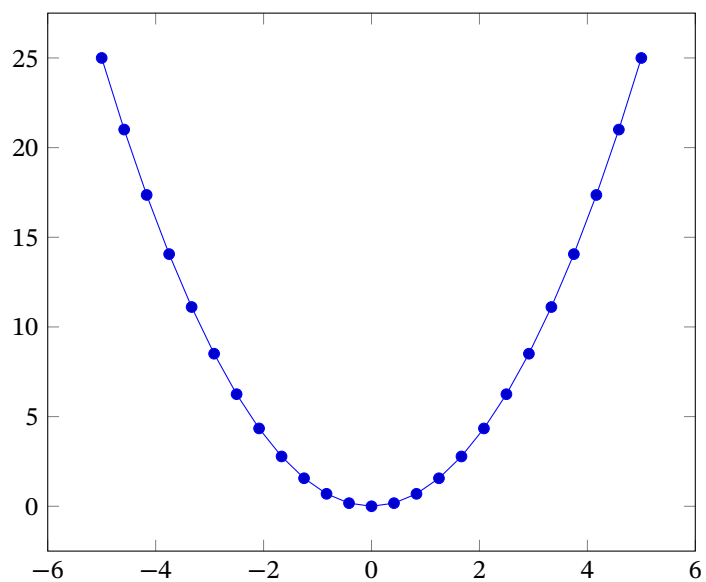
```
\begin{tikzpicture}
\begin{axis}[width=300pt, height=258.75pt,
title = {Manually setting width and height}]
\addplot{x^2};
\end{axis}
\end{tikzpicture}
```

Manually setting width and height



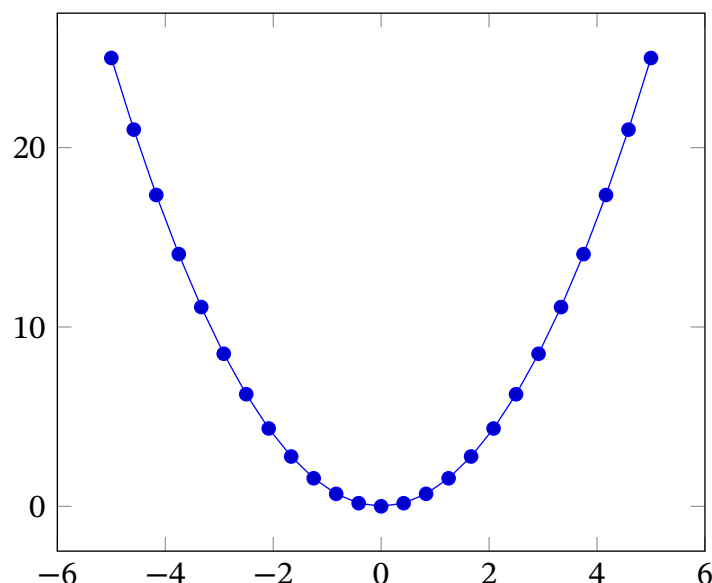
```
\begin{tikzpicture}
\begin{axis}[scale=1.25,
title = {Manually setting logical scale}]
\addplot{x^2};
\end{axis}
\end{tikzpicture}
```

Manually setting logical scale



```
\begin{tikzpicture}[scale=1.25]
\begin{axis}[title = {Manually setting visual magnification}]
\addplot{x^2};
\end{axis}
\end{tikzpicture}
```

Manually setting visual magnification



The first two are vaguely identical (probably the difference is how the distances for tick marks, and tick labels are handled by `pgfplots`). The disadvantage of the first one is that I had to both (1) guess what to set the width, or remember that the original width is 240pt and then increase that by 25%, and (2) if I wanted to preserve the aspect ratio then I had to calculate the height with a calculator (which is what I did).

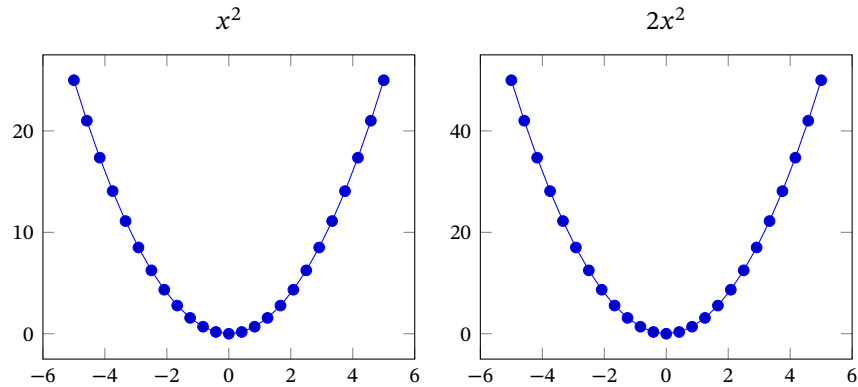
Note in the last one everything got larger, including the numbers and the line thickness. This could be an advantage if you are producing something for a reader with a visual impairment.

Warning: it's pretty easy to forget (or not notice) the difference in how to input the logical magnification and visual magnification: In the former we put `scale=1.25` inside the `axis` options, and the latter we put it inside the `tikzpicture` options.

4.4 Setting external aspect ratio

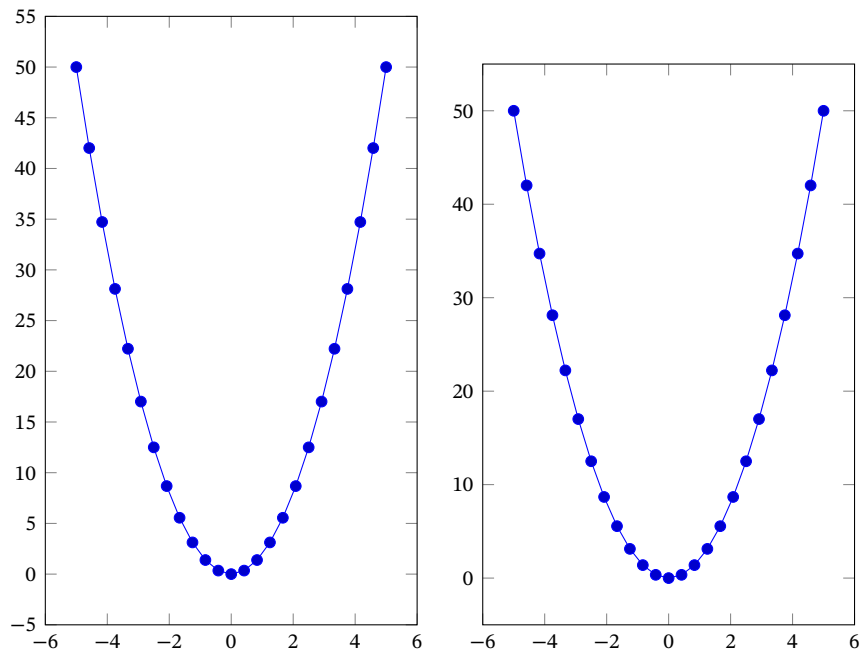
Sometimes you might want to override the default setting for the external aspect ratio, like for the following graphs:

```
\begin{tikzpicture}
\begin{axis}[small,title={\mathit{x}^2}]
\addplot{x^2};
\end{axis}
\end{tikzpicture}%
~~~
\begin{tikzpicture}
\begin{axis}[small,title={\mathit{2x}^2}]
\addplot{2*x^2};
\end{axis}
\end{tikzpicture}
```



If you were teaching students about vertical stretches, it sure might be nice to have that second graph scaled to be twice as tall as the first. Here are two ways to get this done, the first probably being the best

```
\begin{tikzpicture}
\begin{axis}[small,y post scale=2]
\addplot{2*x^2};
\end{axis}
\end{tikzpicture}%
~~%
\begin{tikzpicture}
\begin{axis}[small,height=9cm]
\addplot{2*x^2};
\end{axis}
\end{tikzpicture}
```



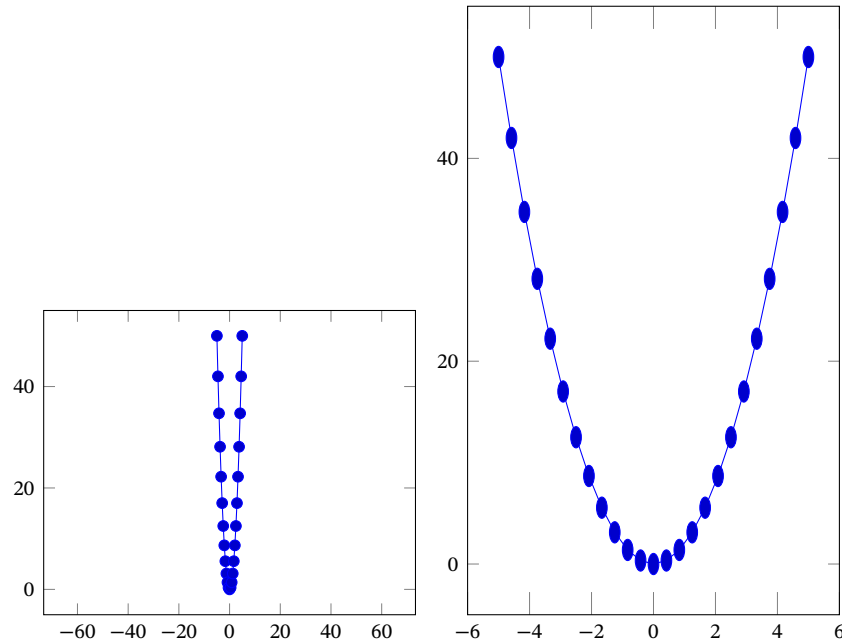
The only small problem with the first way is remembering the key `y post scale` (I get this confused with the standard Tikz key `yscale` which does *not* work, see below). The problem with the second way is that I need to guess (or remember and calculate) the number to set equal to height.

Here are two ways that do *not* work. The first because the `unit vectors` and therefore the `unit vector ratio` for `axis` refers to the *internal axis dimensions*.

```

\begin{tikzpicture}
\begin{axis}[small,unit vector ratio = 1 2]
\addplot{2*x^2};
\end{axis}
\end{tikzpicture}%
~~%
\begin{tikzpicture}
\begin{axis}[small,yscale=2]
\addplot{2*x^2};
\end{axis}
\end{tikzpicture}

```



Note that most of the scaling descriptions in the `pgfplots` manual are for *internal* lengths. Thus `scale mode` and `unit vector ratio` and `axis equal` and `x=` and `y=`, and similar discussions about “unit vectors” and “unit vector lengths” are all changes to the internal lengths. However, the `post scale` options affect the external aspect ratio because they are applied after all the other (internal) scaling has been done.

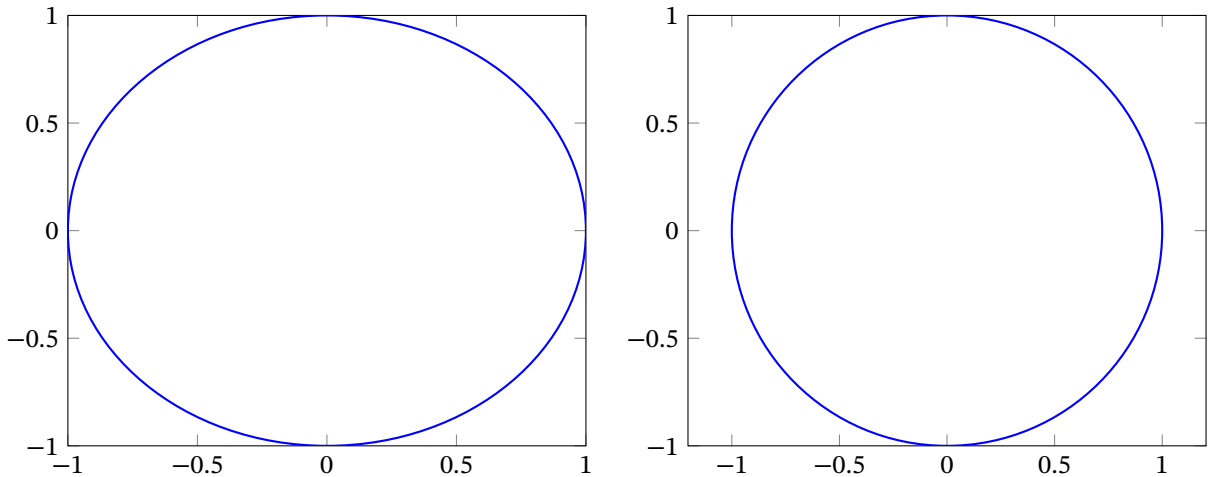
4.5 Setting internal aspect ratio

Note that there are certainly times where you might want to change the internal aspect ratio (although clearly not in the above example with $2x^2$). For instance to make a circle look like a circle, or to create a deliberate rescaling in internal units:

```

\begin{tikzpicture}
\begin{axis}[xmin = -1, xmax=1, ymin=-1,ymax=1]
\draw[thick,blue] (0,0) circle (1);
\end{axis}
\end{tikzpicture}%
~~%
\begin{tikzpicture}
\begin{axis}[axis equal,xmin = -1, xmax=1, ymin=-1,ymax=1]
\draw[thick,blue] (0,0) circle (1);
\end{axis}
\end{tikzpicture}

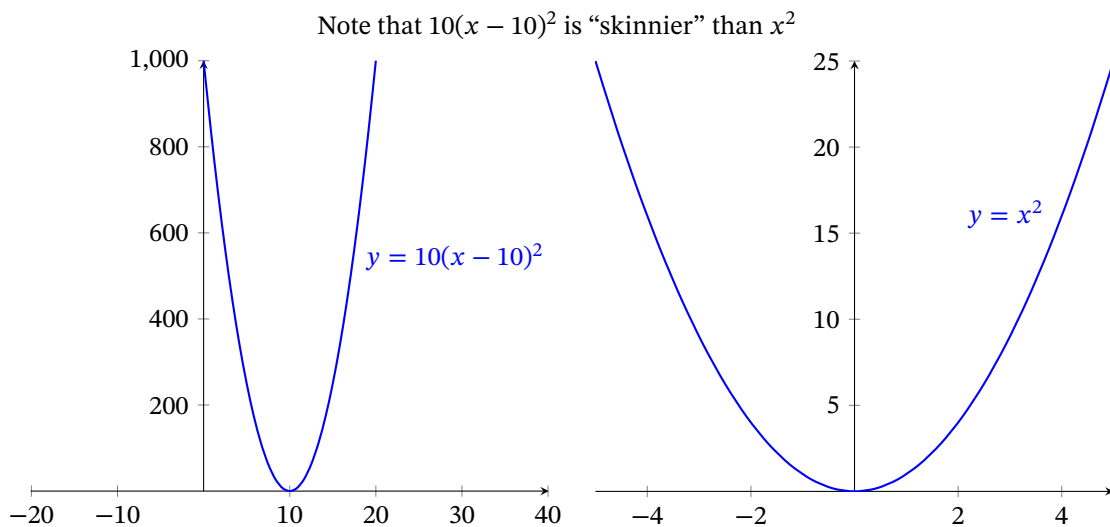
```



Note that the keys `axis equal` and settings for `xmin`, `xmax` are somewhat contradictory (given the fact that `pgfplots` is not going to automatically change the *external* dimensions of this picture), so `pgfplots` tries the best it can, but in the end `axis equal` wins and you can see the picture extends beyond the `xmin` and `xmax` settings.

Here's an example where we purposely do want to make one graph skinnier:

```
\begin{center}
Note that  $10(x-10)^2$  is ‘skinnier’ than  $x^2$ \\
\begin{tikzpicture}
\begin{axis}[unit vector ratio = 20 1,
axis lines=middle]
\addplot[smooth,thick,blue, domain=0:20]
{10*(x-10)^2} node[pos=0.8,below right]{$y=10(x-10)^2$};
\end{axis}
\end{tikzpicture}%
~~%
\begin{tikzpicture}
\begin{axis}[axis lines=middle]
\addplot[smooth,thick,blue] {x^2} node[pos=0.8, above left]{$y=x^2$};
\end{axis}
\end{tikzpicture}
\end{center}
```



Here we manually set the internal aspect ratio because otherwise `pgfplots` does too good of a job auto-scaling these pictures and they end up looking the same! (Also, we did not scale the external picture, unlike an earlier example because, for whatever reason, we wanted the figures to be the same size.)

5 Axis lines

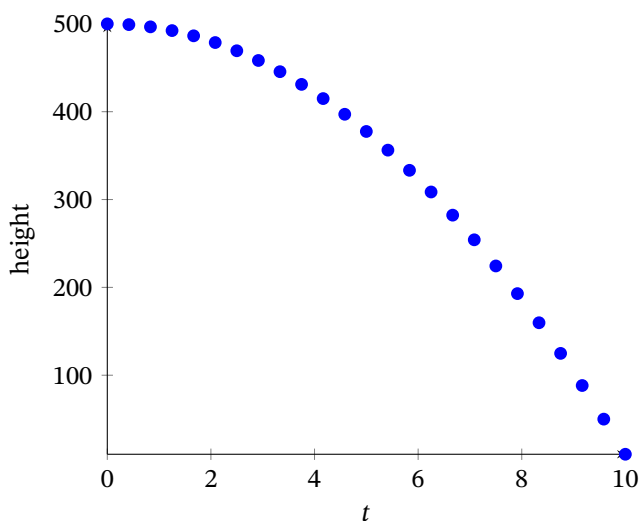
There are times where you might want an option for the axis lines somewhere besides an outer box (the default) and through the middle. The full range of options is

`axis lines = box|left|middle|center|right|none`

where “|” means “or”.

As usual, there are some other style changes that take place when you choose one of these. For instance

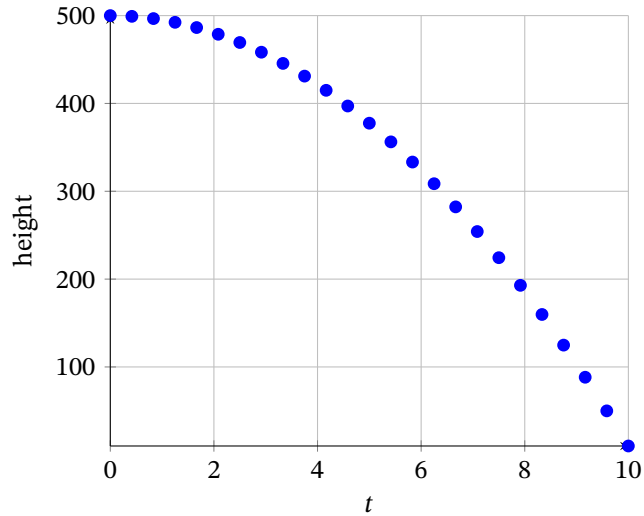
```
\begin{tikzpicture}
\begin{axis}[axis lines = left, xlabel={t}, ylabel={height}]
\addplot[only marks, domain = 0:10, thick, blue,variable=t]{-4.9*t^2+500};
\end{axis}
\end{tikzpicture}
```



Note that it wrote “height” sideways.

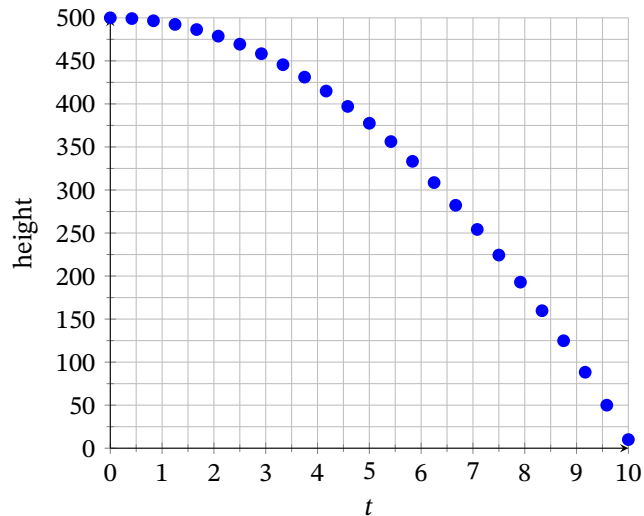
Maybe you’d like to have a grid in this and ask students to estimate the average velocity between two of the points

```
\begin{tikzpicture}
\begin{axis}[axis lines = left, grid,
xlabel={t}, ylabel={height}]
\addplot[only marks, domain = 0:10, thick, blue,variable=t]{-4.9*t^2+500};
\end{axis}
\end{tikzpicture}
```



Maybe you'd like a denser grid

```
\begin{tikzpicture}
\begin{axis}[axis lines = left,
grid=both, % marks both major and minor ticks
xtick distance = 1, % x ticks every integer
ytick distance = 50, % y ticks multiples of 50
minor tick num=1, % one minor tick between each pair of major ticks
ymin=0,
xlabel={t}, ylabel={height}]
\addplot[only marks, domain = 0:10, thick, blue,variable=t]{-4.9*t^2+500};
\end{axis}
\end{tikzpicture}
```



6 Marking points and piecewise functions

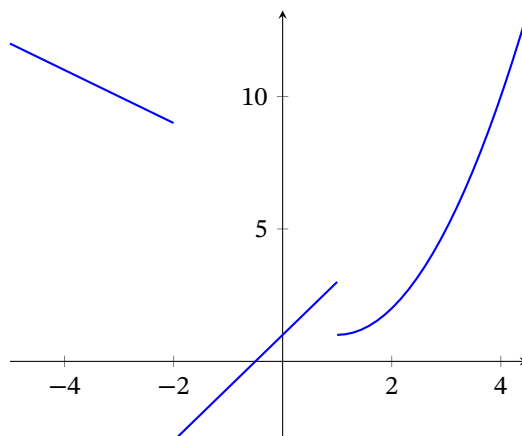
6.1 Basic Piecewise graph

Making a piecewise function is as easy as using more than one `\addplot` command.

```

\begin{tikzpicture}
\begin{axis}[axis lines = middle]
\addplot[thick,blue,domain=-5:-2]{-x+7};
\addplot[thick,blue,domain=-2:1]{2*x+1};
\addplot[thick,blue,domain=1:5]{(x-1)^2+1};
\end{axis}
\end{tikzpicture}

```



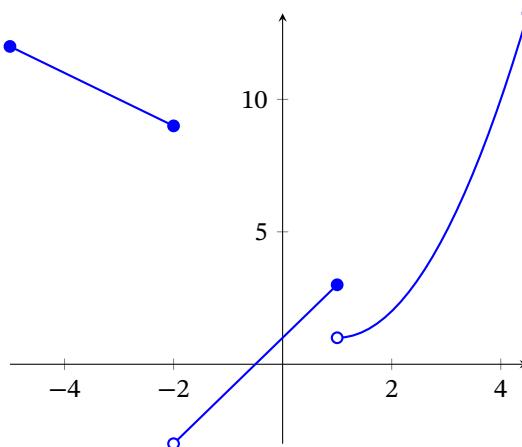
6.2 Marking a single point at a time

It's customary in textbooks to mark the ends of the pieces in a graph like the one above with either open or closed circles, which we'll do with the `mark` key. We'll add marks in two ways: inside the same `\addplot` command that creates the graph of the function, and outside of this in a separate `\addplot` command.

```

\begin{tikzpicture}
\begin{axis}[axis lines = middle]
\addplot[thick,blue,domain=-5:-2,mark = *, mark indices={1,25}]{-x+7};
\addplot[thick,blue,domain=-2:1,mark = *, mark indices=25]{2*x+1};
\addplot[thick,blue,domain=1:4.5,mark = *, mark indices=25]{(x-1)^2+1};
\addplot[thick,blue,only marks,fill=white] coordinates {(-2,-3) (1,1)};
\end{axis}
\end{tikzpicture}

```



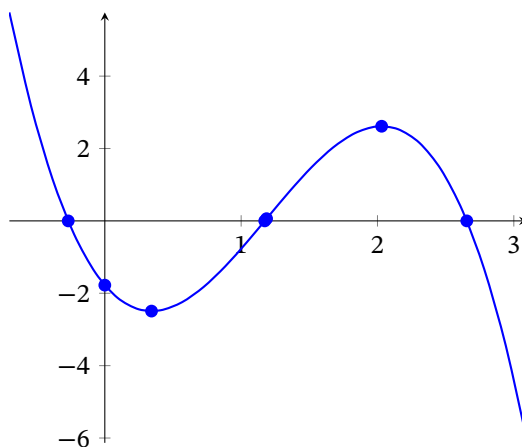
Here's how `mark indices` works: if a function is sampled (calculated) at x_1, x_2, \dots , then the index for point x_i is i . Recall also that the default number of samples is 25, so `mark indices = {1,25}` in the example above marks the 1st point and last point of the graph.

The command `coordinates` expects a list of ordered pairs. The command `fill = white` fills the current path, which in the example above is just the marks for the coordinate list. (Warning: in this case `fill = white` just filled the marks, but if you include it in the original `\addplot` command, it might work the way you want, but it might fill more than you want (see the example at the end of the next subsection)).

6.3 Marking multiple points

You may want to mark multiple points on a single graph, and odds are this will be easier to do by adding a separate `\addplot` command. Suppose you want to mark the x - and y -intercepts, and the max/mins, and the inflection point on the following cubic: $f(x) = -2.124x^3 + 7.561x^2 - 4.433x - 1.776$. I'll assume that we have calculated their locations already at $x \in \{-0.2684, 0.3426, 1.1731, 1.1866, 2.0306, 2.6551\}$. It would be nice to be able to using a single `\addplot` command to graph this cubic and to mark these specific points, but this is not easily possible because we would have to create a list for `samples at` then included these points, and 10–20 others, and then use `mark indices` to select those we want to mark. Instead we'll create a duplicate `\addplot` command, use `only marks` and copy and paste the function expression into the second `\addplot`.⁷

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle]
\addplot[thick,smooth,blue,domain = -0.7:3.1]
    {-2.124*x^3 + 7.561*x^2 - 4.433*x - 1.776};
\addplot[thick,smooth,blue,only marks,
    samples at = {-0.2684, 0, 0.3426, 1.1731, 1.1866, 2.0306, 2.6551}]
    {-2.124*x^3 + 7.561*x^2 - 4.433*x - 1.776};
\end{axis}
\end{tikzpicture}
```



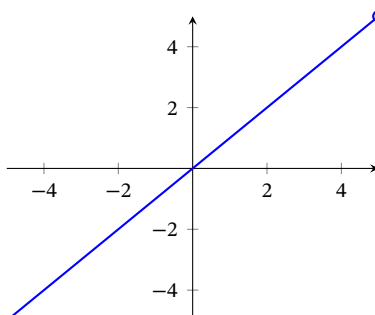
We close this subsection with examples that shows two minor variations on the above methods. Suppose we have just a single graph and want to mark just a single open point at the end of the graph. We might be able to do this just by adding `fill = white` in the `\addplot` command

⁷I requested a feature in the Tikz Github that would have given an easy alternative to this, but it was denied as being superfluous. It would have been something like a key for `marks at = ...` which would have worked like `samples at = ...`, i.e. it would override, for marks alone, any other mark settings or samples and just place marks at the indicated values.

```

\begin{tikzpicture}
\begin{axis}[small,axis lines=middle]
\addplot[thick,blue,fill=white,mark=*,
mark indices = 25]{x};
\end{axis}
\end{tikzpicture}

```

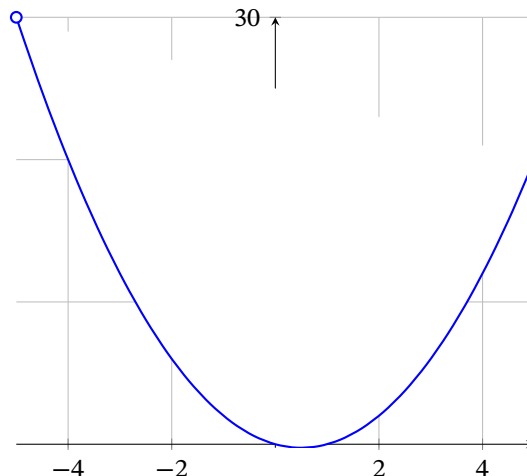


But this use of `fill` applies not just to the mark, it also applies to the whole path, which sometimes leads to undesired results

```

\begin{tikzpicture}
\begin{axis}[grid,axis lines = middle]
\addplot[thick,blue,smooth,mark=*,
mark indices = 1,fill=white] {x^2-x};
\end{axis}
\end{tikzpicture}

```



There are two solutions here: make the `fill = white` only apply to the mark, or use a separate `\addplot` command, but since it's just one point, there's a shortcut available:

```

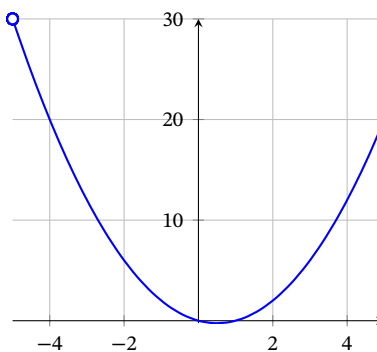
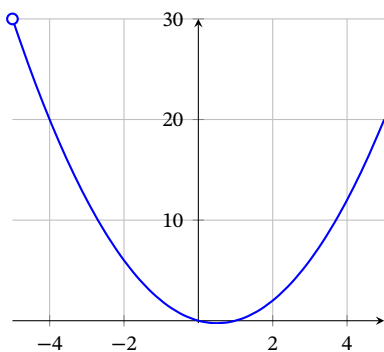
\begin{tikzpicture}
\begin{axis}[small,grid,
axis lines = middle]
\addplot[thick,blue,smooth,mark=*,
mark indices = 1,
mark options={fill=white}]
{x^2-x};
\end{axis}
\end{tikzpicture}

```

```

\begin{tikzpicture}
\begin{axis}[small,grid,axis lines = middle]
\addplot[thick,blue,smooth] {x^2-x};
\addplot[thick,blue,mark=*,fill=white] (-5,30);
\end{axis}
\end{tikzpicture}

```



6.4 Intervals on the number line

We can use the some of the same steps as above to graph intervals on the real number line.

```

\begin{tikzpicture}
\begin{axis}[axis y line = none, axis lines= middle,
enlarge x limits=0.3]
\addplot[very thick, blue,mark=*] coordinates {(7,0) (10,0)};
\end{axis}
\end{tikzpicture}

```



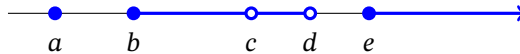
The command `enlarge x limits=0.3` expands the limits of the x-axis by 30% past what it would be otherwise (which would end very near 7 and 10).

Here is a more complicated example

```

\begin{tikzpicture}
\begin{axis}[axis y line = none,axis lines= middle,
enlarge x limits={lower=0.3},
every inner x axis line/.append style = {-},
hide obscured x ticks = false,
typeset ticklabels with strut,
xtick={0,1,2.5,3.25,4},
xticklabels={$a$, $b$, $c$, $d$, $e$}
]
\addplot[very thick, blue] coordinates {(1,0) (3.25,0)};
\addplot[very thick, blue,->] coordinates {(4,0) (6,0)};
\addplot[very thick, blue, only marks] coordinates {(0,0) (1,0) (4,0)};
\addplot[very thick, blue, only marks,fill=white] coordinates {(2.5,0) (3.25,0)};
\end{axis}
\end{tikzpicture}

```



`enlarge x limits={lower=0.3}` expands the axis beyond the points we've plotted, but only at the lower end.

`every inner x axis line/.append style = {-}` causes the x-axis to *not* have an arrow head (because we'll add that with the thick blue line)

`hide obscured x ticks = false` allows for the tick mark at 0 to be shown (usually `pgfplots` hides tick marks at 0 because usually there is a y-axis there!)

`typeset ticklabels with strut` causes all the labels for the tick marks to be aligned along the implied baseline of these letters (otherwise they appear to drift up and down)

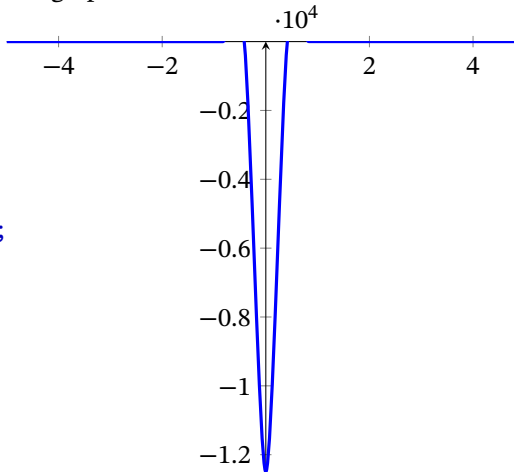
Note: in Section 11 we define some commands that make it much easier to create a number line like this.

7 Asymptotes

7.1 Basic approach

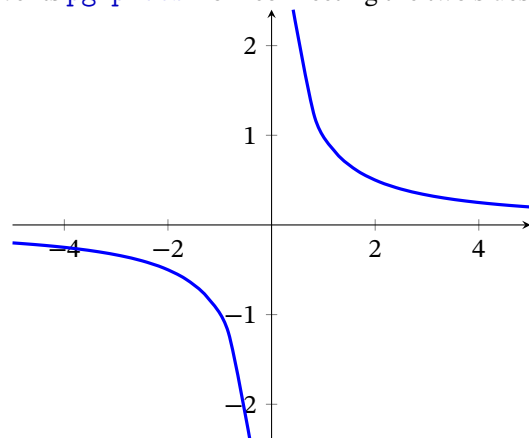
Pgfplots tries to auto-size the y-limits of the axis to fit the graph. But this doesn't work well for asymptotes:

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle]
\addplot[very thick, smooth, blue]{1/x};
\end{axis}
\end{tikzpicture}%
```



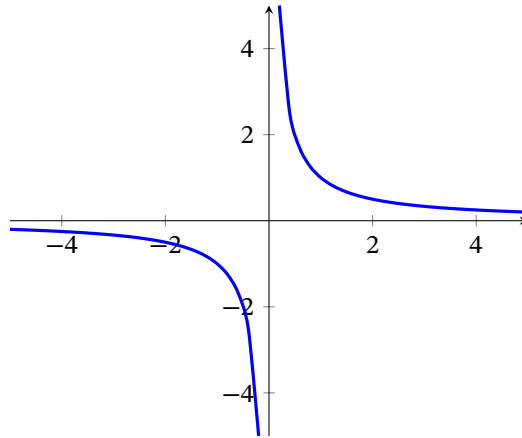
The best way to fix this is to set `restrict y to domain`: this will discard any y-value outside of the given range, and will set a key `unbounded coords=jump` which prevents `pgfplots` from connecting the two sides.

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle,
restrict y to domain =-5:5]
\addplot[very thick, smooth, blue]{1/x};
\end{axis}
\end{tikzpicture}
```



We could make this graph a little better in two related ways. First, you may note that the graph didn't get that close to $y = -5$ and $y = 5$, which we were probably expecting given how we set `restrict y to domain`. This is because with the default number of samples at 25, the middle three samples are $x \approx -0.4167, 0, 0.4167$. The middle number evaluates to `inf`, which is discarded, and the first and last evaluate to $\approx \pm 2.4$, which are plotted, but not that close to ± 5 . The second issue, which is a little hard to see and so may not matter much, is that the graph is relatively straight as it approaches the y-axis. Increasing the number of samples will help both issues:

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle, restrict y to domain =-5:5]
\addplot[very thick, smooth, blue, samples=51]{1/x};
\end{axis}
\end{tikzpicture}
```

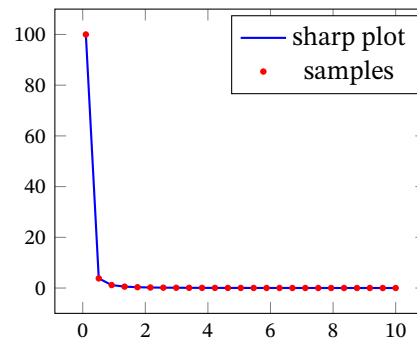
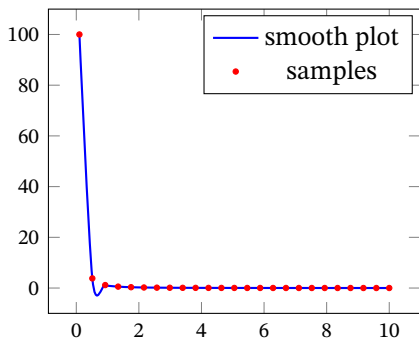


7.2 Unexpected wiggles with `smooth`

Asymptotes provide an example of how a curved graph is not always better with the `smooth` option turned on.

```
\begin{tikzpicture}
\begin{axis}[small,
  mark options={mark size=1pt,color=red}]
\addplot[domain=0.1:10,thick,smooth,
  blue]{1/x^2};
\addplot[domain=0.1:10,mark=*,
  only marks]{1/x^2};
\legend{smooth plot,samples}
\end{axis}
\end{tikzpicture}
```

```
\begin{tikzpicture}
\begin{axis}[small,
  mark options={mark size=1pt,color=red}]
\addplot[domain=0.1:10,thick,
  blue]{1/x^2};
\addplot[domain=0.1:10,mark=*,
  only marks]{1/x^2};
\legend{sharp plot,samples}
\end{axis}
\end{tikzpicture}
```



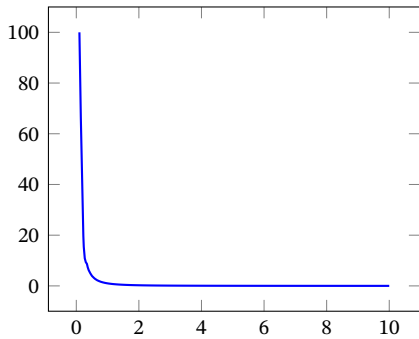
(Note: the only reason we have used two `\addplot` commands is to have the marks get a separate entry in the legend; otherwise, the `mark options` could be combined with `mark = *` in a single `\addplot` to give us the same graph.)

Both of these graphs look poor, but I'd say the first one looks worse. The problem is that `pgfplots` (actually `Tikz`) is trying to make the slope on the second segment of this graph match the slope on the first segment, and that slope is aiming down below where the real graph should go. In any case, both graphs are improved if we simply use more samples; I felt the `smooth` one needed a marginally *greater* number of samples than the `sharp` graph, but the difference in number is probably not that significant.

```

\begin{tikzpicture}
\begin{axis}[small/
\addplot[domain=0.1:10,thick,smooth
blue,samples=80]{1/x^2};
\end{axis}
\end{tikzpicture}

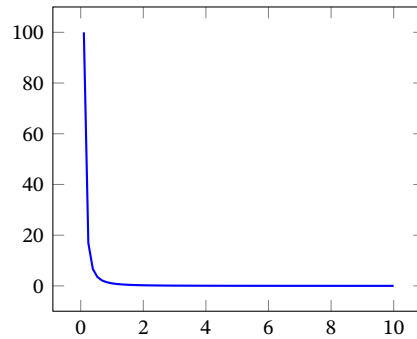
```



```

\begin{tikzpicture}
\begin{axis}[small]
\addplot[domain=0.1:10,thick,
blue,samples=70]{1/x^2};
\end{axis}
\end{tikzpicture}

```



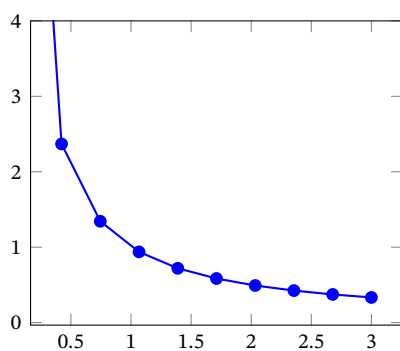
7.3 Fine tuning ymin and ymax versus restrict y to domain

There may be times you want to use `ymin` and `ymax` instead of `restrict y to domain`: the former options *do* (with care) calculate the values outside of the indicated range and add them to the path, which will influence how the graph is shaped within the visible window

```

\begin{tikzpicture}
\begin{axis}[small,ymax=4]
\addplot[domain=0.1:3,
thick,blue,samples=10,
mark=*]{1/x};
\end{axis}
\end{tikzpicture}

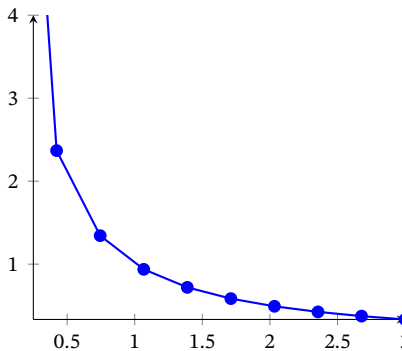
```



```

\begin{tikzpicture}
\begin{axis}[small,
axis lines = middle,
xmin=0.25, ymax=4]
\addplot[domain=0.1:3,
thick,blue,samples=10,
mark=*]{1/x};
\end{axis}
\end{tikzpicture}

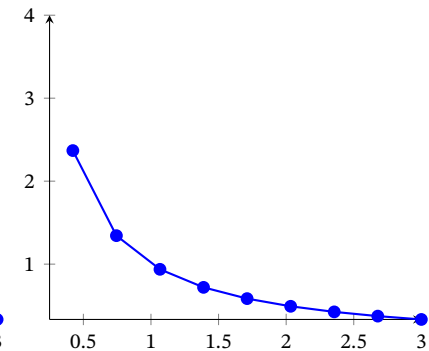
```



```

\begin{tikzpicture}
\begin{axis}[small,
axis lines=middle,
restrict y to domain = 0:4,
xmin=0.25,ymax=4]
\addplot[domain=0.1:3,
thick,blue,samples=10,
mark=*]{1/x};
\end{axis}
\end{tikzpicture}

```



All three graphs show only 9 of the 10 samples requested, but the first two also show the path extending off the edge of the axes, where the 10th sample point was calculated and added to the path. By contrast, the graph with `restrict y to domain` throws away the 10th sample point before `pgfplots` forms the whole path. Somewhat strangely, the axis in the second graph doesn't operate exactly the same as in the first graph: we have to add either an `xmin` key or `enlarge x limits` for the path to extend past the 9th sample.⁸

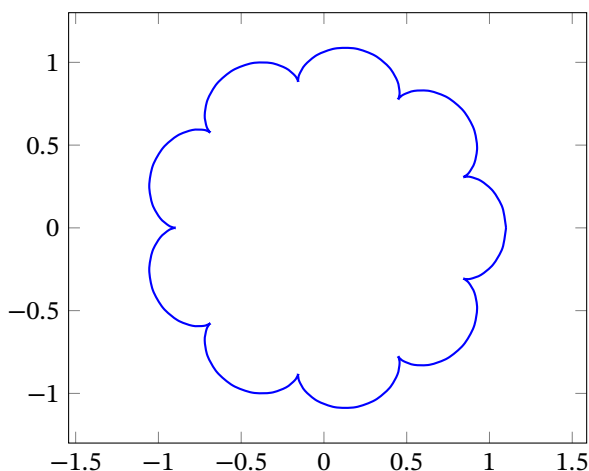
⁸This is either a bug in `pgfplots`, or maybe just an inconsistency. With the axis setting of a box, there is an implicit `enlarge limits`. The fact that this expands the x-axis allows the path to extend past the 9th point and towards the 10th point. We can verify this behavior by adding `enlarge limits = false` to the box axis, and seeing that the path ends early, or by adding `enlarge x limits` to the second

8 Parametric

8.1 Basic use of parametric

To make a parametric graph, use the `\addplot` command as follows: (1) change the variable to `t`; (2) Give it a math expression in two coordinates, i.e. put the functions in an ordered pair like this `(... , ...)`.

```
\begin{tikzpicture}
\begin{axis}[axis equal, trig format plots = rad]
\addplot[thick,smooth,blue,variable = t,
domain=0:2*pi,samples=100]
( {cos(t)+0.1*cos(10*t)}, {sin(t)+0.1*sin(10*t)} );
\end{axis}
\end{tikzpicture}
```



In addition to the changes needed here for parametric mode, note we used “`2*pi`” for the domain (see more about this in Section 9), and we “forced” the trig functions to be in radian mode. The default in Tikz and `pgfplots` is to use degrees, because those seem more natural for drawing things (e.g. if you wanted to draw an angle of 25°), the option above changes that, but only for the expressions in `\addplot` commands.⁹

8.2 Avoiding problematic Cartesian calculations

We showed earlier, in Section 7, that sometimes asymptotes don’t graph as nicely as we might want using the `smooth` option combined with Cartesian calculations, i.e. graphing $y = f(x)$ by evaluating at a list of x -values. Sometimes this can be avoided by using parametric equations. Looking at the graph below on the left, we can see that to avoid the erroneous wiggle we should have the x -values sampled much closer together near the y -axis. One way to accomplish this is to replace $y = 1/x^2$, $0.1 \leq x \leq 10$ with $x = 1/t$, $10 \geq t \geq 0.1$, $y = t^2$ (which gives the same graph).

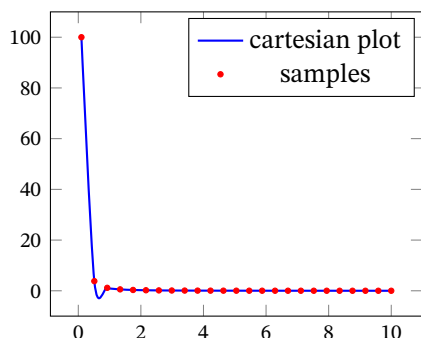
axis instead of changing the `xmin` value, and we’ll still see the top of the graph. By contrast, in the third graph, no change (other than removing `restrict y to domain`) will allow us to see the path continuing in the direction of the 10th sample point.

⁹There is another way to make these functions work in radian mode: there is a postfix operator `(...)r` that causes it’s argument to be interpreted as radians, so that we could have written `cos((t)r)+0.1*cos((10*t)r)` etc.

```

\begin{tikzpicture}
\begin{axis}[small,
  mark options={mark size=1pt,color=red}]
\addplot[domain=0.1:10,thick,
  smooth,blue]{1/x^2};
\addplot[domain=0.1:10,mark=*,
  only marks]{1/x^2};
\legend{cartesian plot,samples}
\end{axis}
\end{tikzpicture}

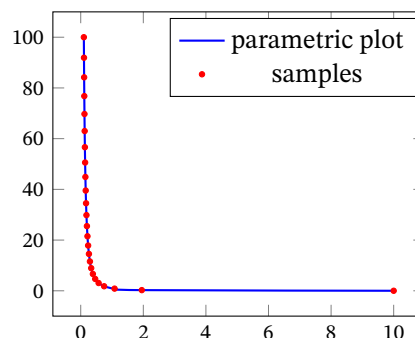
```



```

\begin{tikzpicture}
\begin{axis}[small,
  mark options={mark size=1pt,color=red}]
\addplot[domain=0.1:10,thick,
  blue,smooth,variable = t]({1/t}), {t^2});
\addplot[domain=0.1:10,mark=*,
  only marks, variable = t]({1/t}), {t^2});
\legend{parametric plot,samples}
\end{axis}
\end{tikzpicture}

```



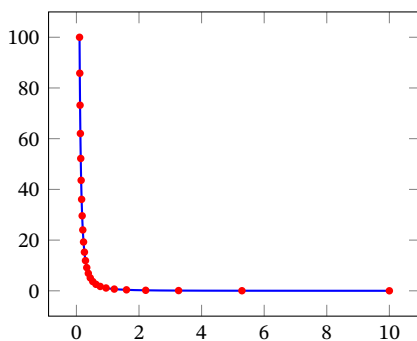
(As in similar examples in Section 7, we have used two `\addplot` commands simply to create two legend entries: otherwise, we could simply use the first `\addplot` command and include `mark=*` in it. We will use just one `\addplot` in the remaining examples.)

Note however two things about fixing the graphs of asymptotes in this way: (1) the appearance of the erroneous wiggle is a geometric property of the points that are calculated using the samples, which in turn depend on the domain used, the number of points sampled, and how those points are spread out. In general, to prevent this problem we wish for more points “near” the asymptote. (2) As a consequence, it’s possible that some choices of parameterizing the Cartesian equation will work (in the sense of getting rid of the erroneous wiggle), some will work even better (in the sense of spreading the samples out) and some will simply reproduce the wiggle, but sideways:

```

\begin{tikzpicture}
\begin{axis}[small,mark options={mark
  size=1pt,color=red}]
\addplot[domain={1/sqrt(10)}:{1/sqrt(0.1)},
  thick, mark=*,blue,smooth,variable = t]
  ({1/t^2}), {t^4});
\end{axis}
\end{tikzpicture}

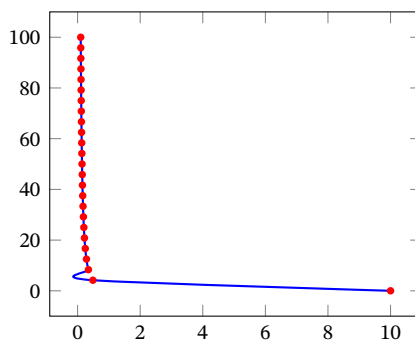
```



```

\begin{tikzpicture}
\begin{axis}[small,mark options={mark
  size=1pt,color=red}]
\addplot[domain=0.01:100,thick,
  mark=*,blue,smooth,variable = t]
  ({1/sqrt(t)}), {t});
\end{axis}
\end{tikzpicture}

```



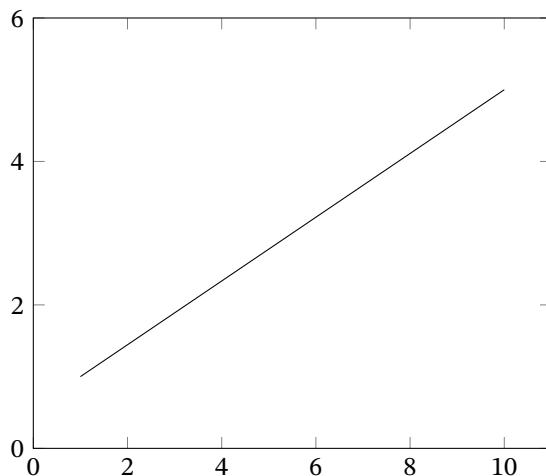
I suspect a good way to choose sample points would be to have them equally spaced along the curve. This could be done easily if we could parameterize $y = 1/x^2$ using arc-length, but unfortunately there is no elementary formula for this (as in, it's probably not possible using a closed form combination of elementary functions). (Note: you can use something like $x = 1/t^{10}$, $y = t^{20}$ and this is marginally better than $x = 1/t^2$, $y = t^4$ shown above, but most of the points still cluster around the “elbow” of the graph.)

9 Math library

9.1 Math expressions in coordinates and domains

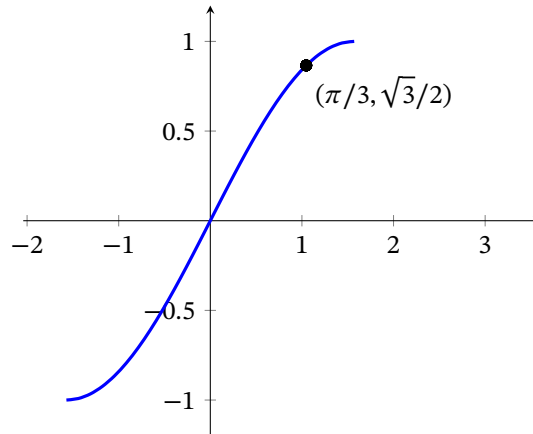
`pgfplots` loads a math library that is part of the Tikz bundle. Among other things it allows you to make calculations within coordinates and domains:

```
\begin{tikzpicture}
\begin{axis}[xmin=0,xmax=11,ymin=0,ymax=6]
\draw (1,1) -- (2*5,6-2/2);
\end{axis}
\end{tikzpicture}
```



Sometimes you need to enclose the math expression in `{...}` to make sure it gets parsed (when in doubt, enclose the material in `{...}`), but the manual says it's necessary when the expression contains `(...)`.

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle,
xmin=-pi/2,xmax=pi,ymin=sin(-90),ymax=sin(90),
trig format plots = rad,enlargetimits]
\addplot[very thick,smooth,blue,domain=-pi/2:pi/2]{sin(x)};
\addplot[mark=*, (pi/3,{sqrt(3)/2})] % note this is parametric coordinate format
node[below right]{ $(\pi/3, \sqrt{3}/2)$ };
\end{axis}
\end{tikzpicture}
```



From the `pgfplots` manual: the operations and functions include “+ , - , * , / , `abs` , `round` , `floor` , `mod` , `<` , `>` , `max` , `min` , `sin` , `cos` , `tan` , `deg` (conversion from radians to degrees), `rad` (conversion from degrees to radians), `atan` , `asin` , `acos` , `cot` , `sec` , `cosec` , `exp` , `ln` , `sqrt` , the constants `pi` and `e` , `^` (power operation), `factorial` , `rand` (random between -1 and 1), `rnd` (random between 0 and 1).” In fact, there are a fair number more functions, see the [Tikz/PGF manual](#).

9.2 Declaring functions

As part of the math package Tikz provides (and which is inherited by `pgfplots`), you can define functions (in the mathematical sense) and use them in plots. You can also define constants, which are essentially functions without any inputs. The crucial command is `declare function`, and the crucial syntax to watch out for is that *each function or constant you define needs to end with “;”*. You can put `declare function` in either the optional arguments to `tikzpicture` or to `axis`. I tend to put it in `tikzpicture` just because in my mental space, it comes as early as possible in the overall picture (I’d put it in the “preamble” of the picture if this existed), and isn’t really part of what I’m drawing.

9.2.1 Quadratic and tangent line

This is a good way to graph functions and tangent lines, because you can easily change the point of tangency:

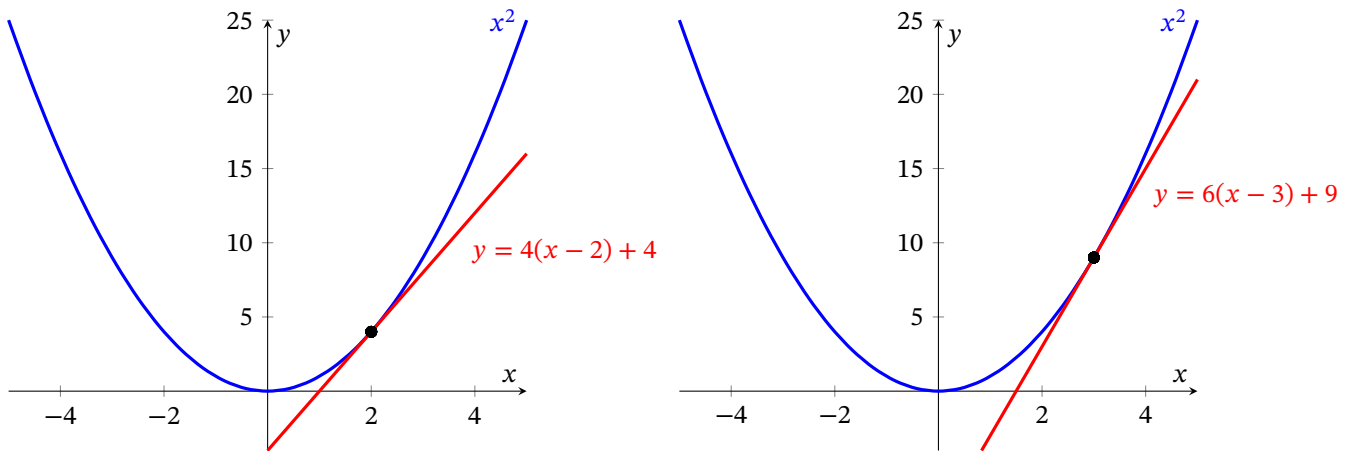
```
\begin{tikzpicture}[declare function = {
a = 2; % ** everything ** is calculated in terms of this
f(\x) = \x^2;
fp(\x) = 2*\x;
m = fp(a);}]
% OK, now I'm showing off, just to format label below
\pgfmathtruncatemacro{\m}{m}
\pgfmathtruncatemacro{\a}{a}
\pgfmathtruncatemacro{\fa}{f(a)}
%
\begin{axis}[restrict y to domain = -5:25,
axis lines=middle,
xlabel={\x$},ylabel={\y$},clip=false]
\addplot[very thick,blue,smooth]{f(x)} node[left]{$x^2$};
\addplot[very thick,red]{fp(a)*(x-a)+f(a)}
node[pos=0.75,below right]{$y=\m(x-\a)+\fa$}; % the label
\addplot[only marks] ({a},{f(a)});
\end{axis}
\end{tikzpicture}%
~~

\begin{tikzpicture}[declare function = {
a = 3; % this is the ** only ** change from above
```

```

f(\x) = \x^2;
fp(\x) = 2*\x;
m = fp(a);}]
\pgfmathtruncatemacro{\m}{m}
\pgfmathtruncatemacro{\a}{a}
\pgfmathtruncatemacro{\fa}{f(a)}
%
\begin{axis}[restrict y to domain = -5:25,
axis lines=middle,
xlabel={\x$},ylabel={\y$},clip=false]
\addplot[very thick,blue,smooth]{f(x)} node[left]{$x^2$};
\addplot[very thick,red]{fp(a)*(x-a)+f(a)}
node[pos=0.75,below right]{$y=\m(x-\a)+\fa$};
\addplot[only marks] ({a},{f(a)});
\end{axis}
\end{tikzpicture}

```



I did show off a little in that example to get the variables stored in a format that I could use in the label. Ordinarily those variables and functions (i.e. `a`, `f`, `fp`, `m`) are only going to be usable inside coordinates (and `domains`, and maybe a few other places ... but not in “regular” LaTeX places that are interpreted as something to print). But the PGF math engine can use them, and so I looked up the command for storing them into macros which could then be used elsewhere. Beware: the internal PGF math engine is not really user friendly, and this is one of the only times I’ve ever used it. Ordinarily I would just type in the label for the tangent line as $y=4(x-2)+4$.

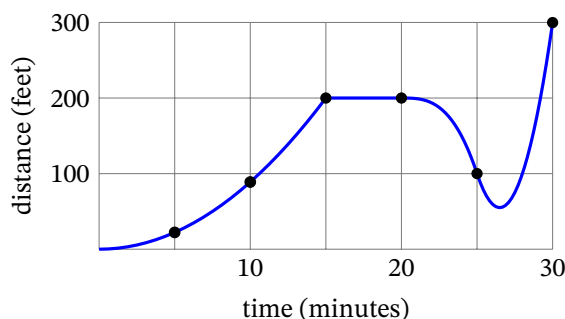
9.2.2 A piecewise graph

In 2023 a member of my department asked me for help in re-creating an example from a book about a function defined by a graph (the problem probably asked for an estimate of velocity at each marked point). At the time I didn’t know `pgfplots` and so I created it in “raw” Tikz. As a result a few things were harder: (1) I created a style for marking points, (2) I had to scale the numbers in the graph to more suitable numbers in Tikz, (3) I added labels to the axes myself, (4) I repeated a math formula in two different parts. Now that I know `pgfplots` I can create the result more easily:

```

\tikzset{point/.style={fill=black,circle,
    inner sep =1.5pt}}
\begin{tikzpicture}
\draw[gray,thin] (0,0) grid (6,3);
\draw[blue, very thick]
  plot[domain=0:3] (\x, {2/9*(\x^2)})
  node[point]{}
  -- (4,2)
  node[point]{}
  plot[domain=4:5] (\x, {-1*((\x-4)^3)+2})
  node[point]{}
  plot[domain=5:6] (\x, {5*(\x^2)-53*\x+141})
  node[point]{};
\draw plot[only marks,mark=*,samples at ={1,2}]
  (\x, {2/9*(\x^2)});
\foreach \x/\label in {2/10,4/20,6/30}
  \draw (\x,0) node[below] {$\label$};
\foreach \y/\label in {1/100,2/200,3/300}
  \draw (0,\y) node[left] {$\label$};
\node[rotate=90] at (-1,1.5) {distance (feet)};
\node[below=15pt] at (3,0) {time (minutes)};
\end{tikzpicture}

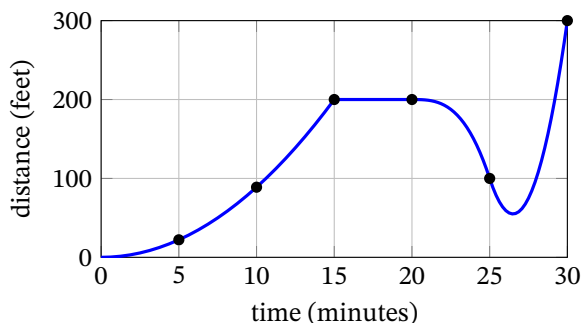
```



```

\begin{tikzpicture}
\begin{axis}[
  grid, x post scale=0.9, y post scale=0.55,
  every axis plot/.style={blue,very thick},
  xmin=0, xmax=30, ymin=0, ymax=300,
  declare function = {f(\x)=8/9*\x^2;},
  xlabel={time (minutes)},
  ylabel={distance (feet)}
]
\addplot [domain=0:15] {f(x)};
\addplot [domain=15:20] {200};
\addplot [domain=20:25] {-4/5*(x-20)^3+200};
\addplot [domain=25:30] {20*(x^2-53*x+705)};
\addplot [only marks, mark options={black,
  mark size = 1.5pt}] coordinates {(5,{f(5)})
  (10,{f(10)}) (15,200) (20,200) (25,100) (30,300)};
\end{axis}
\end{tikzpicture}

```



To reiterate, for me the improvement here is not so much that the code is more compact (though it is), it's that it took less thought on my part: it was easier. It was easier to work with the coordinates I could see, things like (15, 200) rather than scaling those in my head to a smaller Tikz coordinate. It was easier not to worry about adding the tick labels. It was easier to use the function $f(x)$ (which I could have done in Tikz, but didn't know about at the time).

9.2.3 Delta-Epsilon pictures

Next, we illustrate how useful these function-programming features are, by showing a picture that includes information for a δ - ϵ definition of the limit of a function .

```

\begin{tikzpicture}[
  declare function = {
    a=3;
    f(\x) = -(\x-1)^2+16;
    L = f(a);
    eps = 2.8; % chosen to be visible
    del=0.5; % chosen to fit within eps
    x1 = sqrt(16-(L-eps))+1;
    x2 = sqrt(16-(L+eps))+1;
  }
]
\begin{axis}[

```

```

axis lines = middle, xmin=0, enlargelimits,
xtick=\empty, ytick=\empty, clip = false
]

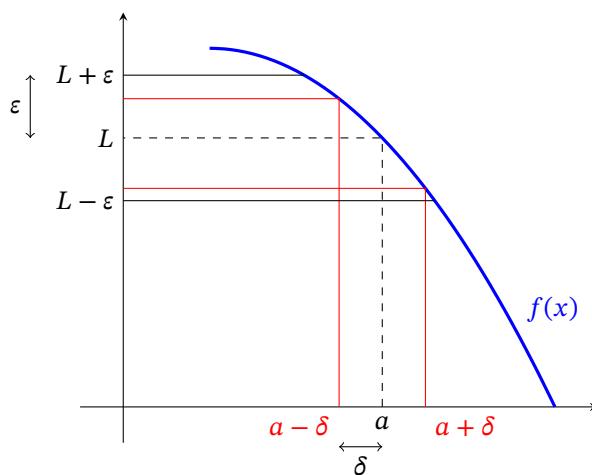
% the actual function
\addplot[domain=1:5, smooth, very thick, blue]{f(x)}
node[pos=0.8, above right]{$f(x)$};

% the parts coming from the x-axis
\draw[dashed] (a,0) node[below]{$a$} -- (a,{f(a)}) --
(0,{f(a)}) node[left]{$L$};
\draw[red] ({a-del},0) node[below left]{$a-\delta$}
-- ({a-del},{f(a-del)}) -- (0,{f(a-del)});
\draw[red] ({a+del},0) node[below right]{$a+\delta$}
-- ({a+del},{f(a+del)}) -- (0,{f(a+del)});

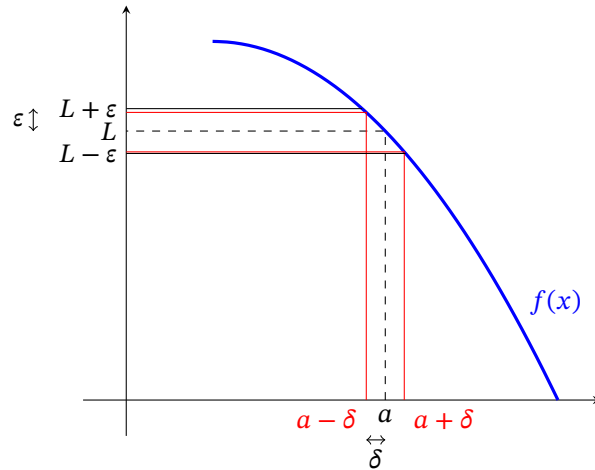
% the parts on the y-axis
\draw (0,{L-eps}) node[left]{$L-\epsilon$} -- (x1,{L-eps});
\draw (0,{L+eps}) node[left]{$L+\epsilon$} -- (x2,{L+eps});

% labeling the distance for delta and epsilon
\draw[<->] ({a-del},-15pt) -- node[below]{$\delta$} (a,-15pt);
\draw[<->] (-35pt,L) -- node[left]{$\epsilon$} (-35pt,{L+eps});
\end{axis}
\end{tikzpicture}

```



Notice a couple of things: (1) It was very convenient to have a function that we could use to calculate $f(x)$. (2) It was very convenient to have a formula for the x -values corresponding to $f^{-1}(L + \epsilon)$ and $f^{-1}(L - \epsilon)$ (we called them `x1` and `x2` above). (3) You can immediately get similar version of this picture where we choose to make δ and ϵ smaller, because the formulas for $f(x)$ and $f^{-1}(L \pm \epsilon)$ will calculate the new values needed in the graph. In the following graph, all we changed compared to the previous code was to set `eps=1` and `del=0.22`. (If we wanted to make ϵ smaller, we would probably need to change a little more in the picture, such as the labels for $L \pm \epsilon$, etc.)

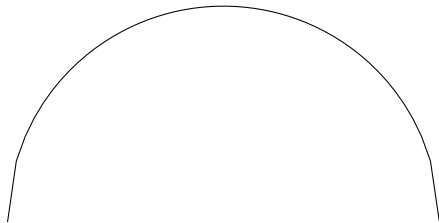


10 Pushing the envelope in speed, efficiency and precision

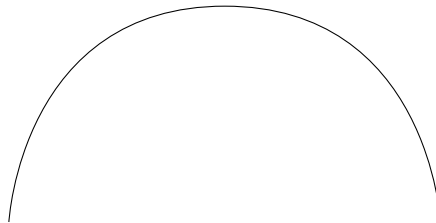
10.1 Smooth is faster

In `pgfplots` and Tikz it is usually faster to use relatively fewer points to plot something and to turn on smoothing, rather than having no smoothing and calculating lots of points (as you might do in Matlab for instance). Shown below are two ways to make a semicircle (of course, the point is not about the semicircle *per se*, it is simply a stand-in for plotting functions).

```
\begin{tikzpicture}
\begin{axis}[axis lines=none, axis equal]
\addplot[domain=-1:1,samples=50]
      {\sqrt{1-(x)^2}};
\end{axis}
\end{tikzpicture}
```

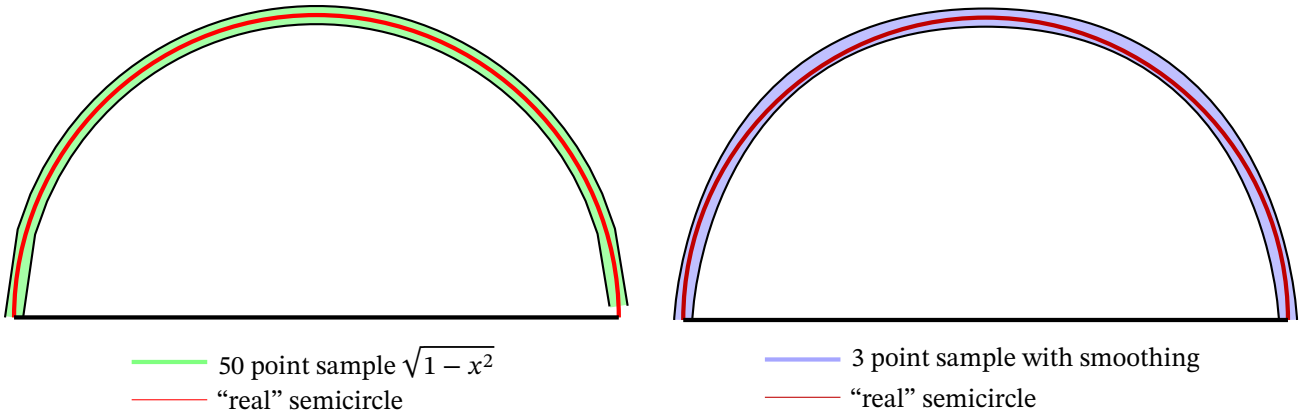


```
\begin{tikzpicture}
\begin{axis}[axis lines=none,axis equal]
\addplot[samples at ={-1,0,1},smooth,
      tension=1.7] {\sqrt{1-(x)^2}};
\end{axis}
\end{tikzpicture}
```



Probably both pictures are good enough, especially when you remember our real goal isn't to draw semicircles but to plot other functions, and for those, a small inaccuracy in shape probably won't be noticeable.

If you want to see more closely how these shapes compare to a “real” semicircle, the graphs below will help. In each case, the red semicircle is made by asking Tikz to draw an arc using the command `arc(<angle>:<angle>:<radius>)`, which I'll assume is more or less perfect. The green path on the left (including the thin black boundary of that path) is made by calculating $\sqrt{1-x^2}$ at 50 points. The right end doesn't quite touch the horizontal, and both ends have a short straight segment. The blue path on the right is made using 3 points and smoothing. The ends of the path are a little too far “in”, and the “shoulders” are a little too far out (as compared to the red arc which I assume is perfect).



So we can see some minor imperfections in each case. Maybe which version you prefer would be a matter of stylistic preference (for me, I don’t like seeing the straight segments on the sample-based approach).

But the biggest difference between the two approaches is speed. Making 1000 copies of the first graph took 26 seconds on a MacBook Air (M1):

```

\foreach \n in {1,2,...,1000}
{\begin{tikzpicture}
  \draw[domain=-1:1,samples=50] plot (\x,{sqrt(1-(\x)^2)});
\end{tikzpicture}}\linebreak[3]

```

and this took 2.3 seconds:

```

\foreach \n in {1,2,...,1000}
{\begin{tikzpicture}
  \draw[samples at ={-1,0,1},smooth,tension=1.7] plot (\x,{sqrt(1-(\x)^2)});
\end{tikzpicture}}\linebreak[3]

```

(Both sets of commands were tested in the command line with “`time pdflatex myfile`”).

This is a rather extreme example, for instance a semicircle only needs three points but most functions will need more. And I had to tweak the tension setting, which I would usually recommend omitting. So, I wouldn’t usually try to use fewer than the default value of 25 samples, but also I would at least consider adding `smooth` to make a better looking graph before increasing the samples too much.

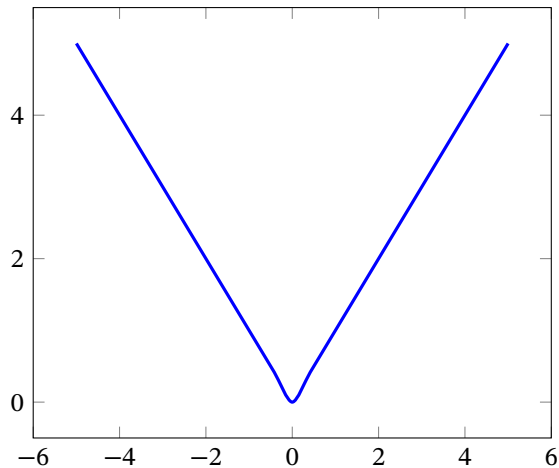
So, why not always use `smooth`? As I said in the beginning of this guide, I would recommend trying it for a *curve* (though see also Section 7 where we saw that `smooth` can introduce an erroneous wiggle in graphs with asymptotes). It’s not so good for absolute value:

```

\begin{tikzpicture}
\begin{axis}[title={plot with smooth}]
\addplot[very thick,smooth,blue]{abs(x)};
\end{axis}
\end{tikzpicture}%

```

plot with smooth

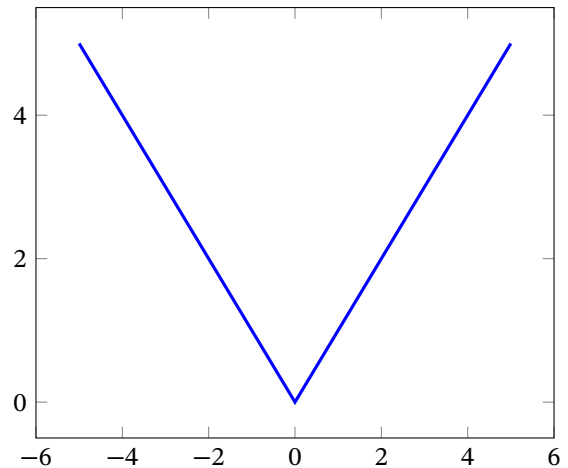


```

\begin{tikzpicture}
\begin{axis}[title={plot without smooth}]
\addplot[very thick,blue]{abs(x)};
\end{axis}
\end{tikzpicture}

```

plot without smooth



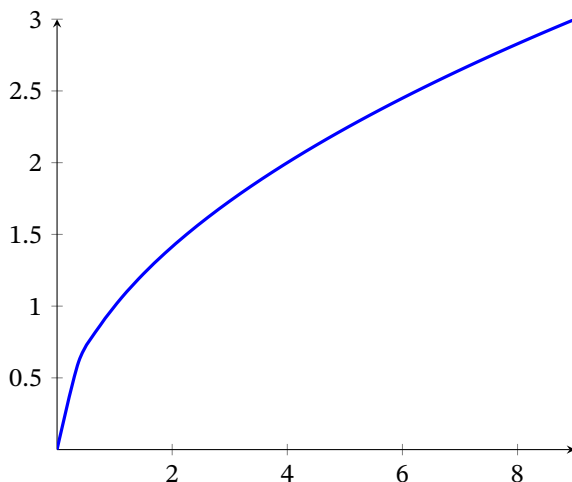
How do you know when you have reason to increase the number of samples? Basically, look at the graph and see what you think. I find the square root graph doesn't look correct at the origin with the standard number of samples:

```

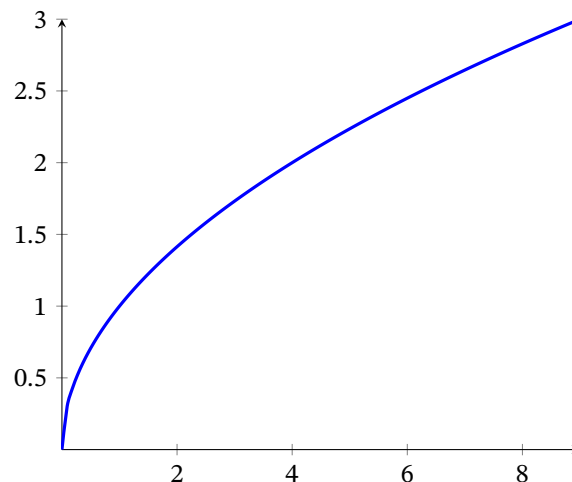
\begin{tikzpicture}
\begin{axis}[axis lines = middle,title={Default samples}]
\addplot[very thick,blue,smooth,domain=0:9]{sqrt(x)};
\end{axis}
\end{tikzpicture}%
~~~%
\begin{tikzpicture}
\begin{axis}[axis lines = middle,title={100 Samples}]
\addplot[very thick,blue,smooth,domain=0:9,samples=100]{sqrt(x)};
\end{axis}
\end{tikzpicture}

```

Default samples

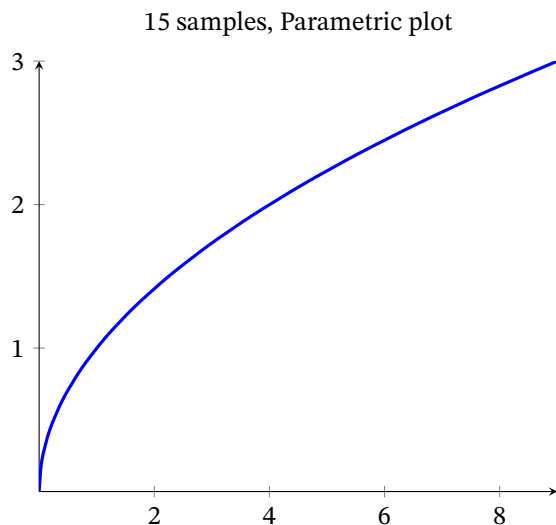


100 Samples

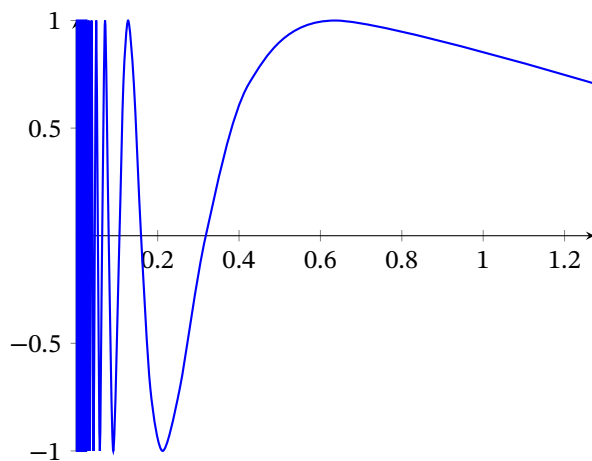


It irks me that I need 100 samples (four times the default) to (*almost*) make the artifact at the origin go away. On the other hand, I need even more points in Matlab, because there's no smoothing there. The problem is that the first and second derivatives of \sqrt{x} are both large at the origin. If I *really* don't want to use this many points, I can do a parametric graph and make y the independent variable, and then I can get by with even fewer samples than the default:

```
\begin{tikzpicture}
\begin{axis}[axis lines = middle,title={15 samples, Parametric plot}]
\addplot[very thick,blue,smooth,domain=0:3,variable=t,samples=15]({t^2},{t});
\end{axis}
\end{tikzpicture}
```



One canonical example of a difficult to graph function is the so-called topologist's sine curve $y = \sin(1/x)$. When I plot this in Matlab, on the interval $[0, 4/\pi]$, with its adaptive command `fplot` the resulting figure uses 1682 x -values! Can we do better? How much better? To do a good job we should have the x -values more densely chosen as $x \rightarrow 0^+$. As is often the case (maybe always?) a parametric approach works a lot better. Here's my final result, and then I'll explain how I got it.



To be clear, I worked hard to make that graph as good as I thought it could be, and to make it as efficient as possible in terms of number of points that are sampled. I'm not recommending you work this hard in general to produce a nice enough looking graph.

In terms of making it parametric, I replaced $y = \sin(1/x)$ with $x = 1/t$ and $y = \sin(t)$. This makes it easy to specify t -values that correspond to meaningful parts of the curve, e.g. $t = \pi/2, 3\pi/2, 5\pi/2, \dots, (2n+1)\pi/2$ will give us a sequence of the max and mins of this curve. Just using these points would give us a curve that is efficient, simple to describe, and at least matches the correct max and mins. In the end, I added some multiples of $\pi/4$ at the beginning, i.e. to produce the right “half” of the curve.

Now, what about the left end? As we get close enough to the y -axis the blue lines blur together, and at that point I will (1) just draw a filled blue rectangle, (2) stop calculating values for $\sin(1/x)$ (where “stop” means I’m thinking of values moving from the right to the left.) Let’s find the x -value where the blue rectangle should meet the $\sin(1/x)$ curve. It should be where two successive peaks of $\sin(1/x)$ have overlapping lines, so that the separate peaks are almost indistinguishable from each other.

The lines have thickness 0.8pt (because that’s the default value for `thick` paths), and that’s an external dimension, i.e. what we should see if we print this out and measure it. Thus, two peaks will be indistinguishable when they have a distance apart, in internal-axis-dimensions, less than or equal to 0.8pt. So, what are the internal dimensions of 0.8pt? Let’s use “iu” for internal unit, i.e. the distance from $x = 0$ to $x = 1$ in the axes. Then we have (see Section 4)

$$\begin{aligned} 1.27 \text{ iu} + 45 \text{ pt} &= 240 \text{ pt} \\ 1.27 \text{ iu} &= 195 \text{ pt} \\ 0.8 \text{ pt} &= \frac{1.27}{195} \times 0.8 \text{ iu} = 0.0052 \text{ iu} \end{aligned}$$

In other words, if x_1 and x_2 are the x -values of two successive peaks, then the peaks would be indistinguishable for $x_1 - x_2 < 0.0052 \text{ iu}$.

So now we ask, what are the first pair of x -values that satisfy this condition? That will tell us the x -value for the right edge of the blue rectangle, and it will tell us what n should be where the peak occurs at $x = \frac{2}{(4n+1)\pi}$. Thus we wish to solve the following

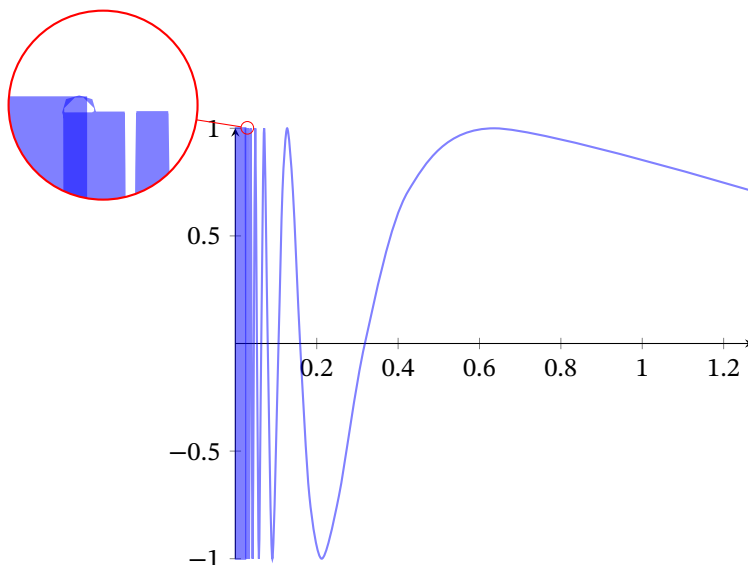
$$\begin{aligned} \frac{2}{(4n+1)\pi} - \frac{2}{(4n+5)\pi} &< 0.0052 \\ \frac{4n+5 - (4n+1)}{(4n+1)(4n+5)} &< \frac{\pi}{2}(0.0052) \\ \frac{(4n+1)(4n+5)}{4} &> \frac{2}{(0.0052)\pi} \\ (4n+1)(4n+5) &> \frac{8}{(0.0052)\pi} \\ n &\geq 5.8 \end{aligned}$$

When $n = 6$ we find $x = \frac{2}{(4 \cdot 6 + 1)\pi} \approx 0.02564$. This is the right edge of the blue rectangle we used in the graph above. And $n = 6$ means that in our parametric plot $t = (4 \cdot 6 + 1)\pi/2 = 25\pi/2 = 25 \times 90^\circ = 2250^\circ$ is the last t -value we use. Here’s the code

```
\begin{tikzpicture}
  \begin{axis}[axis lines = middle,xmin=0,clip=false]
    % y-value on rectangle increased to cover the thickness of the
    % line of the sine curve as it "turns the corner"
    \draw[blue,fill=blue] (0,-1.0025) rectangle (0.02546,1.0025);
    \addplot[thick,blue,variable = t,
      samples at = {
        45, 90, ..., 540, % multiples of 45,
        630, 720, ..., 2250}, % multiples of 90
      smooth] ({1/rad(t)},{sin(t)}) circle (0.01pt);
  \end{axis}
\end{tikzpicture}
```

Here’s a magnification of the top of the curve right at the $x = 0.02546$ boundary (I’ve set `opacity=0.5` on both the rectangle and the sine curve, so we can see how they overlap). Reading from the right, we see two peaks on

$\sin(1/x)$ that are not touching the rectangle, and then the next one is almost entirely covered up by the blue rectangle:

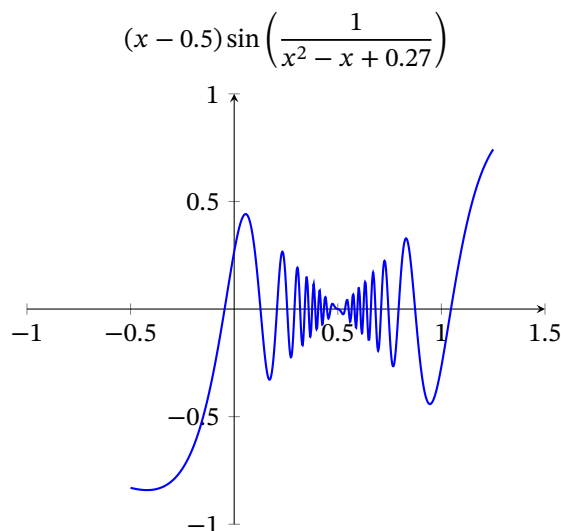


10.2 Graphs with a large number of points

Up to this point I have perhaps overemphasized the goal of being efficient in how many points we need to sample to make a nice graph. There are certainly times when we simply have to throw a pretty large number of points into our computer, and if this does take too long there are ways to help with that too (see Section 12).

The following three graphs are interesting examples where I needed to use 200–300 points to get a result that I thought looked nice:

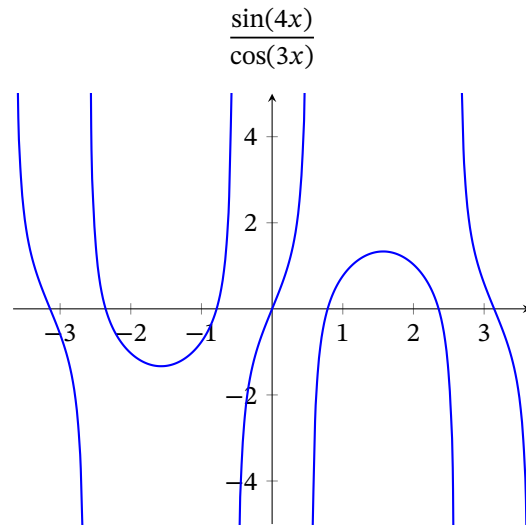
```
\begin{tikzpicture}[
  declare function = {
    f(\x) = (\x-0.5)*sin(1/(\x^2-\x+0.27));
  }
  \begin{axis}[axis lines = middle,
    xmin = -1,xmax=1.5,ymin=-1,ymax=1,
    trig format plots = rad,
    title={
      $(x-0.5)\sin\left(\frac{1}{x^2-x+0.27}\right)$
    }
  ]
  \addplot[thick,smooth,blue,
    domain=-0.5:1.25,samples=300]{f(x)};
\end{axis}
\end{tikzpicture}
```



```

\begin{tikzpicture}[
declare function = {
f(\x) = sin(4*\x)/cos(3*\x);
}]
\begin{axis}[axis lines = middle,
xmin = {-7*pi/6}, xmax={7*pi/6}, ymin=-5, ymax=5,
restrict y to domain = -15:15,
trig format plots = rad,
title={\frac{\sin(4x)}{\cos(3x)}}
]
\addplot[thick,smooth,blue,
domain={-7*pi/6}:{7*pi/6},samples=200]{f(x)};
\end{axis}
\end{tikzpicture}

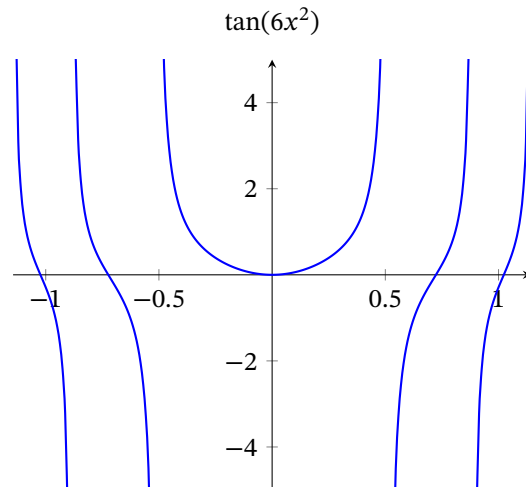
```



```

\begin{tikzpicture}[
declare function = {
f(\x)=tan(6*\x^2);
}]
\begin{axis}[axis lines = middle,
xmin = {-sqrt(5*pi/12)}, xmax={sqrt(5*pi/12)},
ymin=-5, ymax=5,
restrict y to domain = -10:10,
trig format plots = rad, title={\tan(6x^2)}
]
\addplot[thick,smooth,blue,
domain={-sqrt(5*pi/12)}:{sqrt(5*pi/12)},
samples=200]{f(x)};
\end{axis}
\end{tikzpicture}

```



The above examples seem reasonable to me to do within [pgfplots](#), but either for these, or for more extreme ones, it may be a good idea to generate the points in an external program, such as Matlab. Here's a demo, followed by a more meaningful example.

We enter the following in Matlab

```

x=linspace(-3,3,10);
y=x.^2;
[x',y']

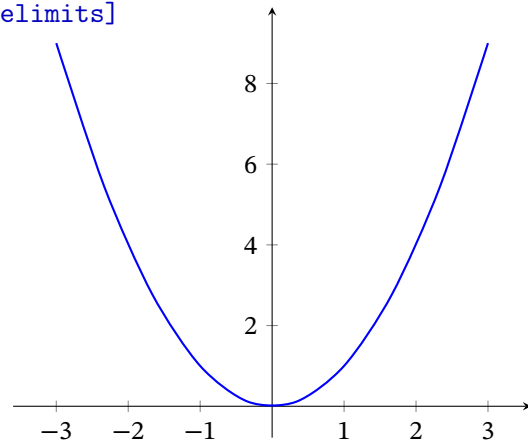
```

and then copy and paste the output into [pgfplots](#) as shown below

```

\begin{tikzpicture}
\begin{axis}[axis lines = middle,enlargelimits]
\addplot[thick,blue,smooth] table
{
-3.0000    9.0000
-2.3333    5.4444
-1.6667    2.7778
-1.0000    1.0000
-0.3333    0.1111
0.3333     0.1111
1.0000     1.0000
1.6667     2.7778
2.3333     5.4444
3.0000     9.0000
}; % must be on line by itself
\end{axis}
\end{tikzpicture}

```



Note that the format is quite simple: each point is defined by a pair of numbers, on a single line with no other material, separated by a space (or more than one spaces).

Of course for the previous example it is not necessary to use Matlab, but for the following example I felt it was. This is what I entered:

```

a=0.5;
b=5;
n=0:7;
x=linspace(-pi,pi,3000); % *3000* points!
f=@(x) (sum(a.^n.*cos(b.^n*pi.*x'), 2))';
fprintf('%g %g\n',[x; f(x)])

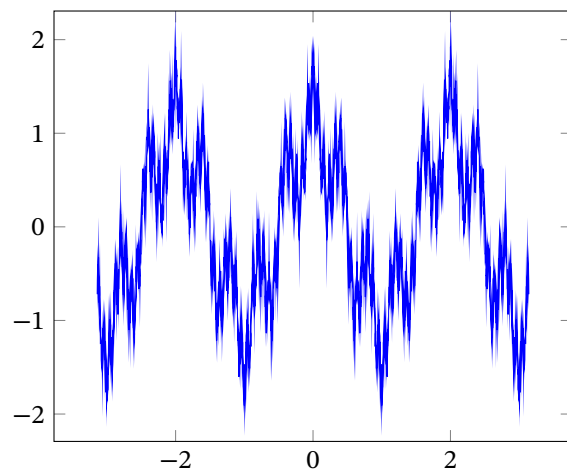
```

Then I copied and pasted the output into a table like above:

```

\begin{tikzpicture}
\begin{axis}
\addplot[very thin,blue,miter limit=100]
table {
-3.14159    -0.719407
-3.1395     -0.571447
% ...
% most lines omitted
% ...
3.1395      -0.571447
3.14159     -0.719407
};
\end{axis}
\end{tikzpicture}

```



Of course, most times you will not want to copy and paste 3000 lines of numbers into your tex-file. In this case, you should save the output in a plain text file, such as `weierstrass_func.txt` and then use the following code:

```

\begin{tikzpicture}
\begin{axis}
\addplot[very thin,blue,miter limit=100] table {weierstrass_func.txt};
\end{axis}
\end{tikzpicture}

```

10.3 Numerical Precision

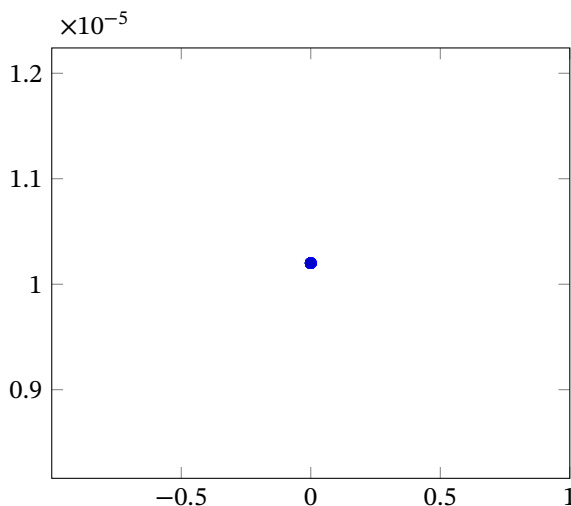
The quality of most graphs is determined by the number of points used and whether or not the `smooth` option is included. But there may be times where you run into issues with numerical precision. PGFPLOTS uses `fpu`, a floating point package *written in TeX* for its calculations. Although `fpu` provides tools for working with a large range of values (“It offers the full range of IEEE double precision computing in TeX.”[8]) it has a somewhat limited precision: “The FPU has a uniform relative precision of about 4–5 correct digits.”[8] We can see what this means here:

```
The fpu internal calculation of
$1-3(4/3-1)$ is
\pgfkeys{/pgf/fpu}
\pgfmathparse{1-3*(4/3-1)}%
\pgfmathprintnumber{\pgfmathresult}.
\pgfkeys{/pgf/fpu=false}
```

The fpu internal calculation of $1 - 3(4/3 - 1)$ is $1.02 \cdot 10^{-5}$.

Of course, the more accurate answer would be 0.
If you prefer to see this graphically here you go

```
\begin{tikzpicture}
\begin{axis}[tick scale binop=\times]
\addplot (0, {1-3*(4/3-1)});
\end{axis}
\end{tikzpicture}
```



Note that the formula $1 - (3(4/3 - 1))$ above is from Cleve Moler, who describes it this way “Before the IEEE standard, this code was used as a quick way to estimate the roundoff level on various computers.”[7, p.40] In other words, this number is $\varepsilon(1)$, i.e. the distance to the “next” floating point number in the system used to do the calculation. This means in `pgfplot`’s `fpu` module we have $\varepsilon(1) \approx 1.02 \times 10^{-5}$, which is another way of phrasing the result quoted above from the `pgfplots` manual about the relative precision. (In single precision floating point systems we have $\varepsilon(1) \approx 1.2 \times 10^{-7}$ and in double precision we have $\varepsilon(1) \approx 2.2 \times 10^{-16}$.) So the precision in PGF and `pgfplots` is a little worse than single precision.

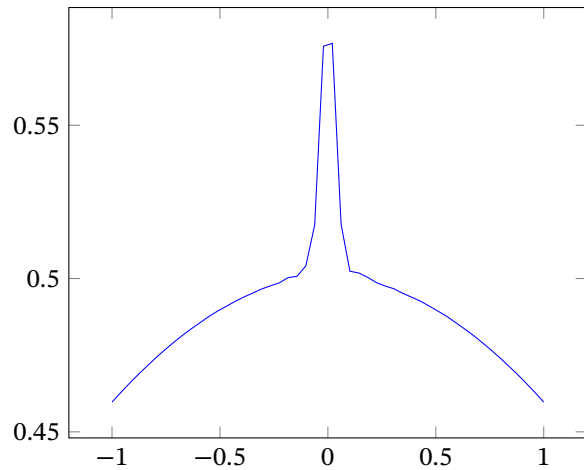
Using the Lua backend (i.e. running “lualatex”) will give increased accuracy for *some* calculations (see [8, p.534]), to a degree that some graphics will look different with lualatex than pdflatex. But, experimentally, for me it produced no difference in the calculations shown above.

A classical example of a basic graph that reveals inaccuracy in calculations is $\frac{1 - \cos(x)}{x^2}$. Here’s how it looks for `pgfplots`:

```

\begin{tikzpicture}
\begin{axis}[trig format plots=rad,
  restrict y to domain =-1:1]
\addplot[blue,domain=-1:1,
  samples=50]{(1-cos(x))/x^2};
\end{axis}
\end{tikzpicture}

```



Based on what we learned in the previous subsection, the right way to handle this is probably to do the calculation externally, e.g. in Matlab. Note, if we do this we do *not* need many sample points, the issue here is precision not sampling the graph. Here's what I entered in Matlab

```

x=linspace(-1,1,10);
y=(1-cos(x))./x.^2;
[x',y']

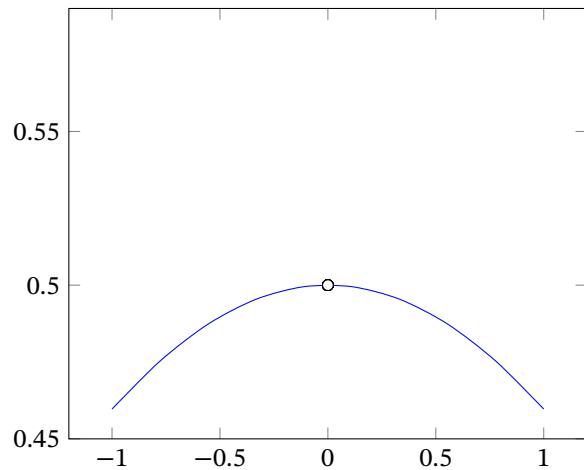
```

and the results of using this in `pgfplots`

```

\begin{tikzpicture}
\begin{axis}[ymin=0.45,ymax=0.59]
\addplot[blue,smooth] table {
  -1.0000    0.4597
  -0.7778    0.4753
  -0.5556    0.4873
  -0.3333    0.4954
  -0.1111    0.4995
   0.1111    0.4995
   0.3333    0.4954
   0.5556    0.4873
   0.7778    0.4753
   1.0000    0.4597
};
\addplot[mark=*,fill=white] (0,0.5);
\end{axis}
\end{tikzpicture}

```



11 Custom Macros

Like in LaTeX more generally, you can create your own custom macros (i.e. commands) to make certain combinations of commands in `pgfplots` easier to use. We've seen one version of this already with defining styles, but here we do something more complicated: create commands to automate typesetting real number lines. Here's a demonstration of the commands:

```

\begin{numline}[small]
\interval{7}{10}{o}{o};
\end{numline}

```



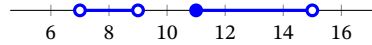
```

\begin{numline}[small,xtick={7,10}]
\interval{7}{9}{o}{o};
\end{numline}

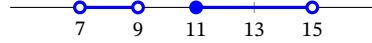
```



```
\begin{numline}[small]
\interval{7}{9}{o}{o};
\interval{11}{15}{c}{o};
\end{numline}
```



```
\begin{numline}[small,
xtick={7,9,...,15}]
\interval{7}{9}{o}{o};
\interval{11}{15}{c}{o};
\end{numline}
```



```
\begin{numline}
\interval{2}{4}{o}{o};
\interval{6}{8}{c}{o};
\interval{-1.5}{1}{<}{o};
\interval{9}{12}{c}{>};
\end{numline}
```



```
\begin{numline}[xtick={1,2,4,6,8,9}]
\interval[red]{2}{4}{o}{o};
\interval{6}{8}{c}{o};
\interval{-1.5}{1}{<}{o};
\interval{9}{12}{c}{>};
\end{numline}
```



You can see that often the code takes care of everything you might need, but sometimes you'll want to adjust the tick marks, or change colors, etc. You can pass the usual options to `numline` that you would otherwise pass to `axis`, and you can pass options to `\interval` that you would otherwise pass to `\addplot`.

In fact, `\begin{numline}` equals

```
\begin{tikzpicture}
\begin{axis}
```

with some additional optional settings of the same kind that we used for number lines in Section 6.

The command

```
\interval<num1><num2><opt1><opt2>;
```

means this: the interval is drawn between `<num1>` and `<num2>`; the ends are marked with `o` or `●` or `→` or `←` according as `<opt1>` and `<opt2>` are "o", "c", ">" or "<" respectively (where `<opt1>` determines the mark at `<num1>`, and `<opt2>` for `<num2>`).

Note: the commands are somewhat smart, but one thing they don't do (yet?) is automatically make the `→` or `←` long enough so that it's guaranteed to extend past the end of the x -axis, so sometimes you'll need to adjust the number you have that marks "where infinity ends" i.e. where `→` or `←` is located.

To define these commands I needed some fancier code: some more detailed settings from `pgfplots` and some internal \TeX control commands. You should feel free to skip reading the definitions of these commands (unless this is your kind of thing!).

Here is the code for `numline`

```
1 \newenvironment{numline}[1] []
2 {
3   \begin{tikzpicture}[>=stealth]
4   \begin{axis}[axis y line = none,
5     axis lines= middle,
6     enlarge x limits=0.3,
7     scatter/classes={
8       o={mark=*,fill=white},
9       c={mark=*} % <- no comma here
10    },
```

```

11 clip = false, % false so infinite intervals can extend past end
12 enlarge x limits = 0.3,
13 every inner x axis line/.append style = {-},
14 hide obscured x ticks = false,
15 #1]
16 }
17 {
18 \end{axis}
19 \end{tikzpicture}
20 }

```

Lines 7–10 define a class of points for use in a “scatter plot” (which is what we’ll use for the intervals). Line 13 redefines the style for the x-axis so that it does not end with an arrow. Line 14 is used so that we can make a tickmark at $x = 0$ (ordinarily `pgfplots` would hide a tick mark here because ordinarily the y-axis would be there). Finally, `#1` in line 15 is how we pass options from `numline` back to `axis`.

The code for `\interval` is here

```

\newcommand{\interval}[5] [] {
  \ifx #4<
    \ifx #5> % line <->
      \errmessage{ERROR: The interval <-> is not defined in this command}
    \else % line <-
      \addplot[very thick,blue,scatter, scatter
        src = explicit symbolic,#1] coordinates {(#3,0) [#5]};
      \draw[very thick,<-,blue] (#2,0) -- (#3,0);
    \fi
  \else
    \ifx #5> % line ->
      \addplot[very thick, blue,scatter, scatter src
        = explicit symbolic,#1] coordinates { (#2,0) [#4]};
      \draw[very thick,->,blue] (#2,0) -- (#3,0);
    \else % - line
      \addplot[very thick, blue,scatter, scatter src
        = explicit symbolic,#1] coordinates { (#2,0) [#4] (#3,0) [#5]};
    \fi
  \fi
}

```

The commands `\ifx ... \else ... \fi` are if-then-else statements and are not part of `pgfplots`; they are part of \TeX , i.e. the original basic program, and so I won’t attempt to describe them further. The new parts for us are the options in `\addplots`. The option `scatter` creates a scatter plot, which doesn’t do much in this case except allow us to define classes of different types of points, i.e. the classes labeled “c” and “o”. The option `scatter src = explicit symbolic` tells `pgfplots` that the points will be given explicitly, and the classes for each marker will be identified symbolically, meaning by the symbols “c” and “o”.¹⁰

12 Breaking up and externalizing

12.1 Breaking up

Using `pgfplots` with lots of plots that have lots of points can slow a document down. The oldest, most basic approach to improving a slow document is to break your document up into different parts and then, when you’re editing and making lots of changes, just typeset one part at a time. The standard way to do this is to use `\includeonly` and `\include`, something like this

¹⁰In theory it would be nice to have “<” and “>” also treated as labels for classes in the scatter plot. But adding an arrow head to a path is not as simple as marking a point: Tikz has to change exactly how the line is drawn as it hits the arrow head. Thus I handled these options with if-then statements and extra commands to draw the arrows.

```

\includeonly{section1} % only section 1 will be typeset

\begin{document}
\include{section1}
\include{section2}
\include{section3}
% etc.
\end{document}

```

When you're ready to produce the whole document, just put a comment symbol % in front of `\includeonly` and the whole document will be typeset. I have done that for this document and it's helped a lot: this *whole* document takes 6.1 s but this section alone takes 0.7 s.

12.2 Externalizing

Odds are you can skip this section unless you are making a document that is hundreds of pages and/or has hundreds of figures. As mentioned in Section 10, speed can start to be an issue in using `pgfplots` if you have a document that has lots of figures and they have a large number of `samples`. In a program like Matlab you might not hesitate at all in running a plot with 300, or even 1000 points. But TeX, after all, was written during 1978–1990, uses fixed point arithmetic and was designed to calculate vertical and horizontal placement of boxes: it's not really designed to be fast at 1000 calculations of floating point numbers.

As an example, the current version of this document is 53 pages, has 96 figures, and takes 6.1 s on my current MacBook Air (M4, 2026). At that rate it would take about 2 min for a 1000 page book, which is probably too long to be convenient. But the PGF package includes a library (sub-package) written by the author of `pgfplots` that helps out.

When one uses the `externalize` library this is what happens: When Tikz finds a `tikzpicture` environment it checks to see if it is new in that document, and/or if it has changed since the last time the document was typeset. If so, it creates a temporary document that contains just that picture, typesets it, and saves the PDF that has just that picture. Then it inserts that PDF into the main document. On subsequent typesetting runs, it checks again, and assuming nothing has changed in that picture, it simply inserts the PDF into the main document. Thus, on subsequent runs, the amount of time needed is roughly just the same as a simple call to `\includegraphics`. This can represent a huge speed increase.

Here's an example of how I use it

```

% in preamble
\usetikzlibrary{external}
\tikzexternalize
\tikzsetexternalprefix{tikzfigs/}

% in main document
\tikzsetnextfilename{3p5_example_1}
\begin{tikzpicture}
...
\end{tikzpicture}

```

Note that to activate `externalize` you need to do two things: load `external`, and issue the `\tikzexternalize` command. This way you can keep the package loaded, but comment out `\tikzexternalize` to turn off externalizing, and just run things like normal.

It's not necessary to use `\tikzsetexternalprefix` but it's a good idea, because it puts the files created by `external` in a separate folder. It's not necessary to use `\tikzsetnextfilename`, without it the library will automatically create a file name for the external image, but this automatic feature is fragile in the sense that if you insert a new picture (or delete an old one) then the automatic file names are thrown off (they are named with a number like “`picture_0`”), and so I use `\tikzsetnextfilename` (actually I created my own shorter command name which is just an alias for this command).

After doing the above you need to typeset the file with a command that has extra permissions; for Linux and MacOS systems one can use the following:

```
pdflatex -shell-escape mainfile
```

(Warning: `mainfile` CANNOT have spaces in its name). On subsequent runs you can use your usual typeset command.

For further details the reader should refer to the Tikz/PGF user manual.

References and Further Resources

- [1] https://www.overleaf.com/learn/latex/Pgfplots_package. Overleaf introduction to pgfplots.
- [2] <https://github.com/hengxin/cheat-sheets/blob/master/latex/latex-pgfplots-cheat-sheet.md>. A short intro by a Github user.
- [3] <https://www.underleaf.ai/learn/pgfplots/pgfplots-for-beginners>. Underleaf introduction to pgfplots.
- [4] <https://www.useoctree.com/learn/pgfplots>. A company or organization webpage with a short intro.
- [5] Tamara Kolda, *Unlocking LaTeX Graphics*, <https://latex-graphics.com>. A full length book about Tikz/PGF and PGFPLOTS.
- [6] Stephen Kottwitz, *LaTeX Graphics with Tikz*, <https://www.packtpub.com/en-us/product/latex-graphics-with-tikz-9781804618233>. A full length book about LaTeX and Tikz.
- [7] Cleve Moler, *Numerical Computing with MATLAB*, https://www.mathworks.com/moler/index_ncm.html. A textbook in numerical computing, by Cleve Moler, the founder of Matlab and Mathworks, the company that sells Matlab.
- [8] Dr. Christian Feuersänger, *Manual for Package PGFPLOTS*, <https://ctan.org/pkg/pgfplots>. Definitive reference manual for PGFPLOTS.
- [9] Till Tantau, *Tikz & PGF: Manual for Version 3.1.11a*, <https://ctan.org/pkg/pgf>. Definitive reference manual for Tikz and PGF.

PGFPLOTS Cheat Sheet for Calculus-type graphs

Addplot

```
\addplot coordinates {<coordinate list>;  
\addplot table [<column selection>]{<table>;  
\addplot {<math expression>;  
\addplot[variable = t] ( {<math>}, {<math>} );
```

Coordinate list example:

```
{(1,2) (10,12) (25,30)}
```

Inline table example

```
{  
1 1  
2 4  
3 9  
}; % must be by itself on last line
```

Math expression example

```
\addplot{e^{-x^2}};
```

Coordinates

Cartesian coordinates

```
(length1,length2)
```

Polar coordinates

```
(angle:length)
```

Coordinate math expression examples:

```
(pi/2, {sin(pi/2)}) or (1e6, 2e7)
```

Options for axis

Box, x-axis, y-axis, etc

```
axis lines=box|left|middle|center|right|none  
axis x line=box|top|middle|center|bottom|none  
axis y line=box|left|middle|center|right|none
```

Labels

```
xlabel = <text>, ylabel = <text>
```

Title

```
title = <text>
```

Ticks

```
ticks = minor|major|both|none  
xtick = \empty|<number list>  
ytick = \empty|<number list>  
xtick distance = <dim> % dist between ticks  
ytick distance = <dim>  
minor tick num = <num> % ticks between ticks  
xticklabels=<text list>  
yticklabels=<text list>
```

(WARNING: xticklabel is an unrelated command)

Grids

```
grid=minor|major|both|none
```

Ticklabel scale format

```
tick scale binop=<binary operator>
```

Axis external sizing options

Size styles

```
normalsize: 8.4 cm × 7.3 cm = 3.3 in × 2.9 in  
small: 6.50 cm × 5.60 cm = 2.52 in × 2.20 in  
footnotesize: 5 cm × 4.31 cm = 1.97 in × 1.70 in  
tiny: 4 cm × 3.45 cm = 1.57 in × 1.36 in
```

Setting width and height

```
\begin{axis}[width=<dim>, height = <dim>]
```

Scaling whole image

```
\begin{tikzpicture}[scale=1.25] % visual mag  
\begin{axis}[scale=1.25] % logical mag
```

Changing external aspect ratio

```
\begin{axis}[y post scale=2] % change height  
\begin{axis}[height=9cm] % change height
```

Clipping

```
clip = true|false  
clip mode = global|individual  
restrict y to domain = <min>:<max>
```

Axis internal sizing options

Setting internal limits

```
xmin = <number>, xmax = <number>,  
ymin = <number>, ymax = <number>
```

Changing internal aspect ratio

```
\begin{axis}[axis equal]  
\begin{axis}[unit vector ratio = <num> <num>]
```

Options for Addplot (mostly from Tikz)

Line thickness:

```
ultra thin|very thin|thin|semithick|thick  
|very thick|ultra thick|linewidth=<dim>
```

Line style:

```
solid|dashed|dotted|dash dot|dash dot dot  
|densely dashed|loosely dashed|<etc>
```

Color Example:

```
red!50!black,ultra thick
```

Line doubling example

```
double=white,thick
```

Line shape

```
sharp|smooth
```

Where functions are evaluated

```
domain = <min>:<max>  
samples = <whole number> % num of values used  
samples at = {<numbers>} % manual override
```

Marks

```
only marks % don't connect points  
marks = none  
mark = *|+|x|o|star|square|oplus|diamond|<etc>  
mark size = <dim>  
mark indices = {<index list>} % which points  
mark options = {<style declarations>}
```

Style declarations example

```
mark options = {scale=2,thick,fill=white}
```

Math options

Setting trig functions in plot commands to use radians

```
trig format plots = rad
```

Defining constant and function example:

```
declare function = { a = 5; % <- semicolon!  
f(\x) = (\x-a)^2; % <- semicolon!  
}
```

Defining styles

Storing a style in a name

```
\pgfplotsset{duckplot/.style={thick,blue,smooth}}  
\addplot[duckplot]{x^2};
```

Setting a style for whole document

```
\pgfplotsset{  
every axis/.style = {axis lines = middle},  
cycle list name = color list,  
every axis plot/.style = {thick,smooth}  
}
```

Externalize

```
\usetikzlibrary{external}  
\tikzexternalize  
\tikzsetexternalprefix{tikzfigs/}  
\tikzsetnextfilename{3p5_example_1}
```

Command line

```
pdflatex -shell-escape mainfile
```

(Note: mainfile CANNOT have spaces in name)