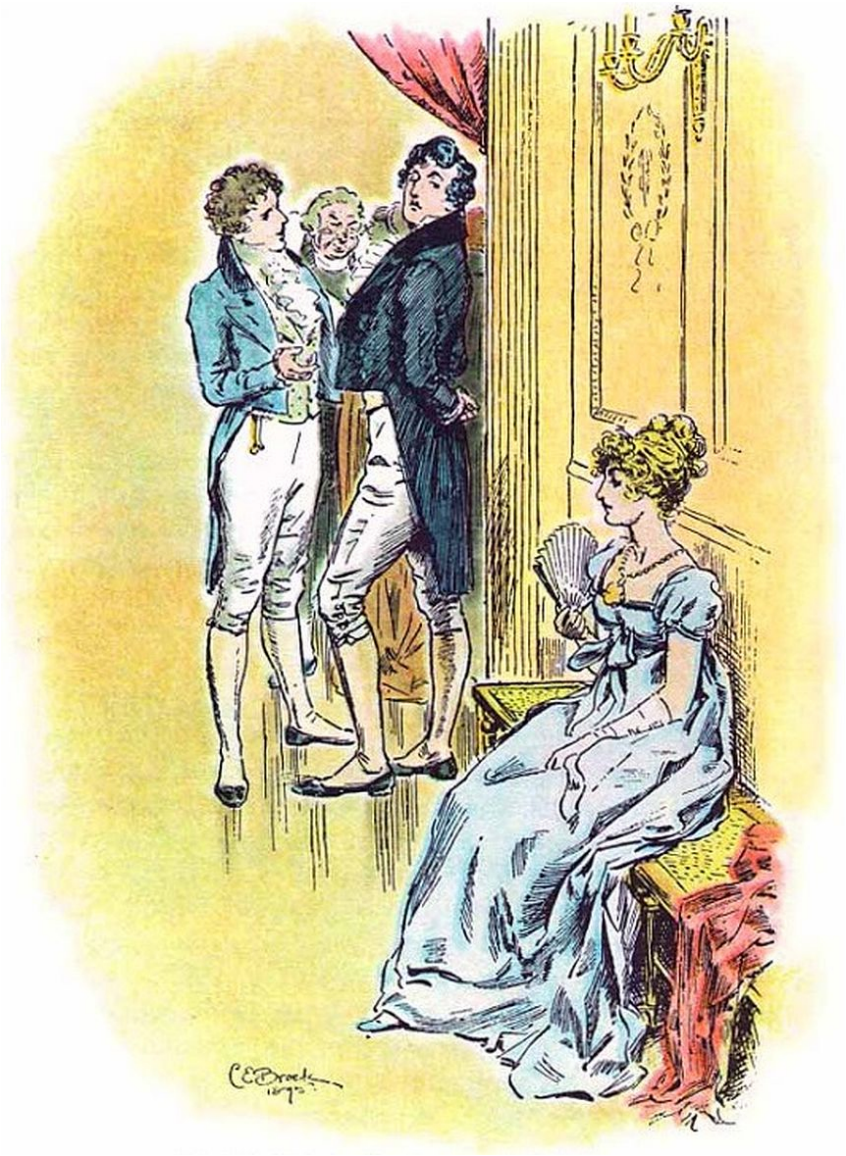
First impressions of *Asymptote*



Jim Hefferon

Cover image C E Brock. Public domain, via Wikimedia Commons.

Preface

Asymptote is a language for drawing mathematical figures. It outputs vector graphics and fits well with TEX, LATEX, and friends. It is great with two-dimensional graphics but it also sparkles with three dimensions, including that it extends to 3D algorithms from METAFONT and METAPOST that are elegant and that give beautiful curves.

Last year I made a careful set of lecture slides for Calculus I and II, and I drew the figures with *Asymptote*. It is wonderfully well-suited to the job.

However, recently I mentioned this success to someone, who later told me that they had an online search for *Asymptote* and found a technical reference, a long tutorial, and a gallery with many graphics, but really no quick overview. Rather than tackle what was there, they stuck with what they were using.

Hence this document. It is short, adopting a few familiar Calculus graphics. It gives a feel for what you can do with *Asymptote*, without being too much (although I believe that the examples use every feature of *Asymptote* that I used for my lectures). You can work through it in an afternoon. If you do elect to try *Asymptote* then its web site has many more advanced resources.

I hope that it is a help.

Remark: This is an introduction. Often I will do something one way, showing one option, when there are many ways to do it. For instance, you can include *Asymptote* source in your LATEX document but here I have the source as standalone files. Another example is that I will take the *Asymptote* files to be in a `asy/` subdirectory, while of course you can organize your work in many different ways. Showing only one option is just a question of going shorter.

Jim Hefferon
University of Vermont
2024-Sep-29

Chapter 1: A first graphic

Making an *Asymptote* input file is like making a LaTeX input file, so you already have a feel for the basics. To start, use your favorite editor to open a file `asy/unit_circle.asy`.

```
jim@millstone:~/Documents/asy_tut/src$ cd asy
jim@millstone:~/Documents/asy_tut/src/asy$ emacs unit_circle.asy
```

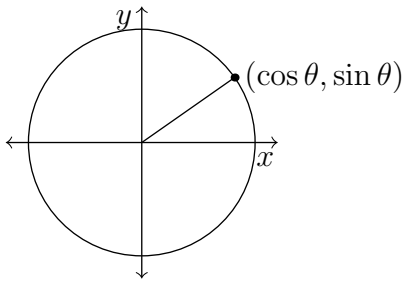
Type this text into the file and save it, or copy it from this document's online source.

```
// unit_circle.asy
settings.outformat="pdf";  // Output PDF file

import graph; // Module for plotting routines, including xaxis() and yaxis()

unitsize(1.5cm);  // One x unit or y unit will be 1.5cm.

// Draw a unit circle
draw(unitcircle);

// Draw a generic angle
real theta = radians(35);
pair generic_pt = (cos(theta), sin(theta));
draw((0,0)--generic_pt);
dot("$(\cos\theta,\sin\theta)$",generic_pt,E);

// Draw the axes
xaxis("$x$",xmin=-1.2,xmax=1.2,Arrows(TeXHead));
yaxis("$y$",ymin=-1.2,ymax=1.2,Arrows(TeXHead));
```

Get the output file `unit_circle.pdf` by running *Asymptote* on the source.

```
jim@millstone:~/Documents/asy_tut/src/asy$ asy unit_circle
```

You can see the result in any PDF viewer.



What do we have here? The input `unit_circle.asy` illustrates a number of things. Globally, it shows that you must declare variables such as line 12's `theta`, that commands end with a semicolon, and that comments are preceded by a double slash, `//`.

Locally, line 2's `settings.outformat` variable fixes the format of output files to PDF so we needn't remember each time to do that from the command line. Line 4's `import` is like a LaTeX `\usepackage{...}` in that it gives us access to a module, a collection of related commands and data. In this case it gives us commands to make plots. In this drawing we only use its axis-making commands but the second chapter has more about plots.

In line 6 the `unitsize(1.5cm)` command means that if we describe a point (x, y) then *Asymptote* will interpret it as the location $x \cdot 1.5$ cm and $y \cdot 1.5$ cm from the origin.

Line 9 is self-explanatory. Lines 12 through 15 draw the line segment from the origin to the point labeled $(\cos\theta, \sin\theta)$. (On line 15, in addition to drawing the dot, *Asymptote* labels it and the `E` puts that label east of the dot.)

Finally, lines 18 and 19 draw the axes. These commands have many options, most of which we did not use here, and we will see more of them later.

One more thing. *Asymptote* gets the point label $(\cos\theta, \sin\theta)$ and the axis labels by putting the given strings in a small file, running LaTeX on it, and then extracting the result back into the output graphic. So your labels have access to all of LaTeX's capabilities.

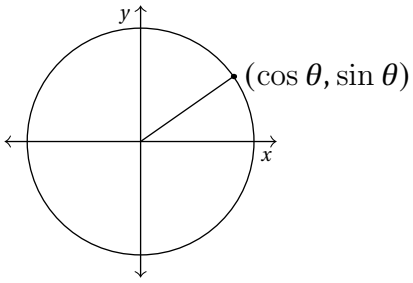
Adjustments After you get a graphic draft there are always some tweaks.

First, here the axis labels are too big. We will replace `"$x$"` with `"\scriptsize $x$"`. (We could instead omit the label by deleting the entire string and the comma after it, which illustrates that commands such as `"xaxis(..)"` can have a variable number of arguments. While you are working you may want to have open the *Asymptote* reference for a list of the options.)

Second, the dot showing the generic point on the unit circle is too big. In the revised source below we've adjusted the size by inserting `dotfactor = 4` in line 22 (the default factor is 6).

Finally, the $(\cos\theta, \sin\theta)$ label is in a different font than the other mathematics in this overview. We want that *Asymptote*, when making the small LaTeX document to create the in-graphic text, will use the same font setup as the main `.tex` file. So below we added a `texpreamble("..")`; the string is long so for readability we've spread it across multiple lines.

```
// unit_circle_after.asy
settings.outformat="pdf";

texpreamble("\usepackage{mathtools}
  \usepackage[utf8]{inputenc}
  \usepackage[osf,scaled=.92,loosest]{heuristica}
  \usepackage[heuristica,vvarbb,bigdelims]{newtxmath}
  \usepackage[T1]{fontenc}
  \renewcommand*\oldstylenums[1]{\textosf{#1}}");

import graph; // Module for plotting routines, including xaxis() and yaxis()

unitsize(1.5cm);  // One x unit or y unit will be 1.5cm.

// Draw a unit circle
draw(unitcircle);

// Draw a generic angle
real theta = radians(35);
pair generic_pt = (cos(theta), sin(theta));
draw((0,0)--generic_pt);
dotfactor = 4;
dot("$(\cos\theta,\sin\theta)$",generic_pt,E);

// Draw the axes
xaxis("\scriptsize $x$",xmin=-1.2,xmax=1.2,Arrows(TeXHead));
yaxis("\scriptsize $y$",ymin=-1.2,ymax=1.2,Arrows(TeXHead));
```



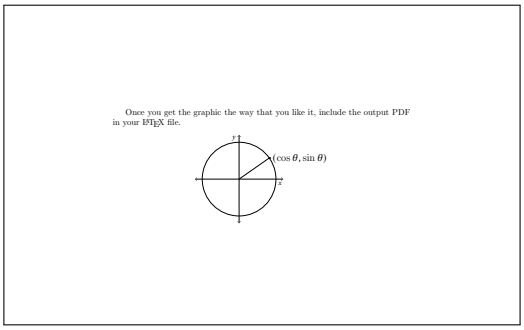
Include the graphic in a LaTeX file Open a new LaTeX file `main.tex`.

```
jim@millstone:~/Documents/asy_tut/src$ emacs main.tex
```

Enter this text or copy it from this document's source.

```
% main.tex  LaTeX demonstration file to use with Asymptote, Jim Hefferon
\documentclass{article}
\usepackage{graphicx}
% \usepackage{asymptote}  % If you use \asyinclude{..}
% \renewcommand{\asydir}{asy}  % Keep this doc's dir tidy, put stuff in asy/

\begin{document}
Once you get the graphic the way that you like it,
include the output PDF in your \LaTeX{} file.
\begin{center}
  \includegraphics{asy/unit_circle_after.pdf}
  % Can instead include the .asy source in the text body.
  % \asyinclude{asy/unit_circle_after.asy}
  % There is also an environment where you type the Asy source directly.
\end{center}
\end{document}
```

This document shows two ways to include the graphic. Line 11 is straightforward because after you've iterated through some adjustments to the figure then you use LaTeX's standard `\includegraphics{..}`. The other way, commented out, uses the `asymptote` LaTeX package to include the *Asymptote* source file with `\asyinclude{..}`. In this approach, getting the graphic is a three step process, where you run "`pdflatex main`" (of course you can instead use "`xelatex main`" or "`lualatex main`"), then you go into the `asy/` subdirectory and run "`asy <latex-filename>-1`" (here, "`asy main-1`"), then go back to the LaTeX file's directory and run "`pdflatex main`" once more. In either case here is the one-page output.



Chapter 2: Plots

We will draw this function.

$$f(x) = x + \frac{1}{x-1}$$

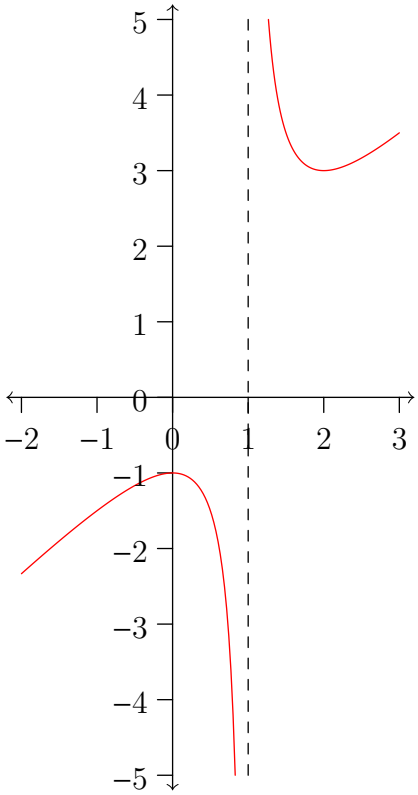
It goes infinite at $x = 1$ so we can't ask *Asymptote* to plot all x's. We will instead plot the x's where the associated y's are between -5 and 5. To find these we can use a computer algebra system such as *Sage* to solve $5 = x + 1/(x-1)$ and $-5 = x + 1/(x-1)$.

```
sage: x = var('x')
sage: solve( [5==x+(1/(x-1))], x )
[x == -sqrt(3) + 3, x == sqrt(3) + 3]
sage: solve( [-5==x+(1/(x-1))], x )
[x == -2*sqrt(2) - 2, x == 2*sqrt(2) - 2]
sage: round(-sqrt(3) + 3, ndigits=3)
1.268
sage: round(2*sqrt(2) - 2, ndigits=3)
0.828
```

That leads to this source file `asy/plot.asy`.

```
// plot.asy
settings.outformat="pdf";

texpreamble("\usepackage[utf8]{inputenc}
  \usepackage[osf,scaled=.92,loosest]{heuristica}
  \usepackage[heuristica,vvarbb,bigdelims]{newtxmath}
  \usepackage[T1]{fontenc}
  \renewcommand*\oldstylenums[1]{\textosf{#1}}");

import graph;

// Function to be plotted
real fcn(real x) {
  return( x+(1/(x-1)) );
}

unitsize(1cm);  // One x unit or y unit will be 1cm.

// Nominal plot boundaries
real xmin = -2; real xmax = 3;
real ymin = -5; real ymax = 5;

// Get the graph
real left_x = 2*sqrt(2)-2;    // From solving -5=x+(1/(x-1))
real right_x = 3-sqrt(3);     // From 5=x+(1/(x-1))
path f_left = graph(fcn, xmin, left_x);
path f_right = graph(fcn, right_x, xmax);

// Draw
draw((1,ymin)--(1,ymax), dashed); // Vert asymptote
draw(f_left, red);
draw(f_right, red);

// Draw the axes, with ticks
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks(Step=1),
      Arrows(TeXHead));
yaxis(ymin=ymin-0.2, ymax=ymax+0.2,
      LeftTicks(Step=1),
      Arrows(TeXHead));
```

Here is the resulting plot.



Adjustments As earlier, on seeing the draft graphic we make some tweaks, which helps give a sense of some available options, leading to the source `asy/plot_after.asy` below.

The axes go through the two 0's and the vertical asymptote passes through the 1. We can change the `xaxis(..)` command to say `RightTicks(Step=1, OmitTick(0,1))`, and similarly change `yaxis(..)`.

Although we limited the output range to between $y = -5$ and 5, the plot is still so tall that it is hard to fit on a page or slide. We make the y unit height be half of the x unit width by adding this command

```
scale(Linear, Linear(0.5))
```

(`Linear` is in contrast with a `Logarithmic` scale). The axes and graph now come out rescaled but we must also adjust the location of points, the ones defining the vertical asymptote line, using for example line 26's `Scale((1,ymin))`.

That tweak of the y axis causes its tick labels to be scrunched together, so we arrange that *Asymptote* labels only every fifth tick (the labeled ones are called major ticks and the others are minor ticks).

```
  yaxis(ymin=ymin-0.4, ymax=ymax+0.4,
      LeftTicks(Step=5, step=1, OmitTick(0), Size=3pt, size=2pt),
      Arrows(TeXHead));
```

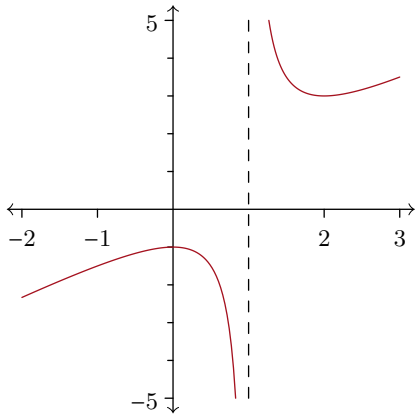
That command also sets the length of these major and minor ticks.

Here is `asy/plot_after.asy`. An explanation of line 3 is in the next section.

```
// plot_after.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

// Function to be plotted
real fcn(real x) {
```

```
  return( x+(1/(x-1)) );
}

unitsize(1cm);
scale(Linear, Linear(0.5)); // Rescale the y axis by half

// Nominal plot boundaries
real xmin = -2; real xmax = 3;
real ymin = -5; real ymax = 5;

// Get the graph
real left_x = 2*sqrt(2)-2;   // From solving -5=x+(1/(x-1))
real right_x = 3-sqrt(3);    // From 5=x+(1/(x-1))
path f_left = graph(fcn, xmin, left_x);
path f_right = graph(fcn, right_x, xmax);

// Draw
draw(Scale((1,ymin))--Scale((1,ymax)), dashed); // Vert asymptote
draw(f_left, HIGHLIGHT_COLOR);
draw(f_right, HIGHLIGHT_COLOR);

// Draw the axes, with ticks
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks(Step=1, OmitTick(0,1), Size=3pt),
      Arrows(TeXHead));
yaxis(ymin=ymin-0.4, ymax=ymax+0.4,
      LeftTicks(Step=5, step=1, OmitTick(0), Size=3pt, size=2pt),
      Arrows(TeXHead));
```

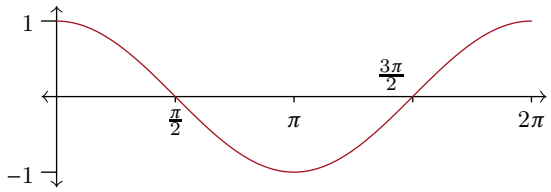
Here is the output.



Defaults Rather than copy and paste elements common across graphics such as the font commands or colors, we can put them in a separate file and import them, as in the prior source's line 3. The source of that file, `jh.asy`, is in the Appendix. (About the `"../../../asy/"` directory stuff: usually we set up *Asymptote* with a directory for common files and then just say `import jh`. But for this document we want that a user can compile without setup so the relative path is in the source.)

Ticks With plot ticks you often want something other than the default. We won't cover all of the options but there are a couple of things we have not yet seen that are especially useful.

On a trigonometric graph

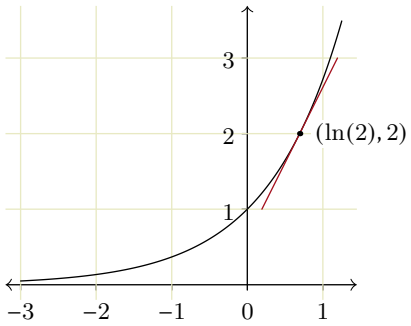


you don't want the x axis to say 1, 2, etc., you want $\pi/2$, π, etc. You also don't want "3.14," you want "π." This illustrates explicit ticks, on lines 19 and 21.

```
// cos.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

real fcn(real x) {
  return( cos(x) );  // Many real functions such as cosine are built in
}

unitsize(1cm);
real xmin = 0; real xmax = 2*pi;  // Asymptote defines pi as a convenience
real ymin = -1; real ymax = 1;

path f = graph(fcn, xmin, xmax);
draw(f, HIGHLIGHT_COLOR);

// Axes with custom ticks
real[] T = {pi/2, pi, 3*pi/2, 2*pi};  // Array of reals holds tick locations
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks("%", T, Size=2pt),  // LaTeX % comment means tick label is blank
      Arrows(TeXHead));
labelx("$\frac{\pi}{2}$",pi/2); // Put in a label by hand, rather than a decimal
labelx("$\pi$",pi);
labelx("$\frac{3\pi}{2}$",3*pi/2, NW);
labelx("$2\pi$",2*pi);
yaxis(ymin=ymin-0.2, ymax=ymax+0.2,
      LeftTicks(Step=1, OmitTick(0)),
      Arrows(TeXHead));
```

Note line 25's `NW`, which prints the $3\pi/2$ northwest of its tick.

Our other tick example has a graph paper effect, with lines in a light color extending across the graph. (I sometimes use this for lectures; here, to estimate by eye that at $y = 2$ the slope of the tangent line is 2.)



The source has a number of interesting features.

```
// exponential.asy
settings.outformat="pdf";
```

```
import "../../../asy/jh.asy" as jh;

import graph;

real fcn(real x) {
  return( exp(x) );
}
real tangent_fcn(real x) {
  return( 2*(x-log(2))+2 );
}

unitsize(1cm);
real xmin = -3; real xmax = 1.25;
real ymin = 0; real ymax = exp(xmax);

path f = graph(fcn, xmin, xmax, n=300);
path tan_line = graph(tangent_fcn, log(2)-0.5, log(2)+0.5);

draw(f);
draw(tan_line, HIGHLIGHT_COLOR);
dotfactor = 4;
dot(Label("$(\ln(2),2)$",filltype=Fill(white)), (log(2),2), 2*E);

// Axes making graph paper
pen GRAPHPAPERPEN=(0.25*LIGHT_COLOR+0.75*white)
  +squarecap;  // For graph paper lines

xaxis(axis=YEquals(ymax+0.2),
      xmin=xmin-0.5, xmax=xmax+0.5,
      p=nullpen,
      ticks=RightTicks("%", Step=1, OmitTick(0), extend=true, pTick=GRAPHPAPERPEN));
xaxis(axis=YEquals(ymin-0.2),
      xmin=xmin-0.5, xmax=xmax+0.5,
      p=nullpen,
      ticks=RightTicks("%", Step=1, OmitTick(0), extend=true, pTick=GRAPHPAPERPEN));

yaxis(axis=XEquals(xmin-0.2),
      ymin=ymin-0.5, ymax=ymax+0.5,
      p=nullpen,
      ticks=LeftTicks("%", Step=1, OmitTick(0), extend=true, pTick=GRAPHPAPERPEN));
yaxis(axis=XEquals(xmax+0.2),
      ymin=ymin-0.5, ymax=ymax+0.5,
      p=nullpen,
      ticks=LeftTicks("%", Step=1, OmitTick(0),  extend=true, pTick=GRAPHPAPERPEN));

// Axes in black with ticks
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks(Step=1, step=0, Size=2pt),
      Arrows(TeXHead));
yaxis(ymin=ymin, ymax=ymax+0.2,
      LeftTicks(Step=1, OmitTick(0), Size=2pt),
      Arrow(TeXHead));  // Arrow singular means no bottom arrow
```

The graph paper effect is due to the input in lines 26 through 46. The horizontal lines are a little clearer so we will cover them. They are created by the `yaxis(..)` commands in lines 39–46. These two vertical axes, one on the left and one on the right, are drawn with a `nullpen` so we don't see vertical black lines at `xmin-0.2` and `xmax+0.2`. What we do see are the ticks for those axes, extending back and forth between them in the color given by `GRAPHPAPERPEN`, because of the `extend=true`. These ticks have a null label because of the LaTeX comment character `"%"`. The y axis on the left produces the horizontal graph paper marks between $x =$ `xmin-0.2` and $x = 0$, while the one on the right generates the marks from $x = 0$ to $x =$ `xmax+0.2`. (The $x = 0$ comes from the y axis in lines 52–54.)

The commands from line 49 to the end produce the axes shown in black.

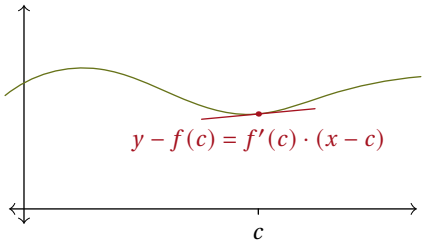
This is a long file but we will discuss a few fine points. One is that the $(\ln(2), 2)$ label has a white back-

ground obscuring some of the graph paper lines, from the `Label("$(\ln(2),2)$",filltype=Fill(white))` command. Another is that the 300 in line 19's `f = graph(fcn, xmin, xmax, n=300)` is there because *Asymptote* draws the graph by connecting dots that evaluate `fcn` at a finite number of points, and the default was too small so that the graphic had visible jaggies.

Finally, lines 18 and 19 as well as lines 27 and 28 make clear that essential to understanding *Asymptote* is understanding the ideas of `path` and `pen`. That's the next chapter.

Chapter 3: Paths and pens

Paths This plots a function that is generic in that it isn't derived from a simple expression such as $\cos x$ or $x+(1/(x-1))$. We will use it for the classic Calculus lesson illustrating that the curve is locally well-approximated by the line, by zooming in on a point of tangency.



Here is the source of that graphic.

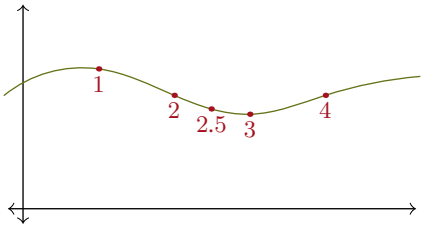
```
// zoom.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

path generic_fcn_plot = (-0.25,0)..(1,0.35)..(2,0)..(3,-0.25)..(4,0)..(5.25,0.25);

unitsize(1cm);
real xmin = 0; real xmax = 5;
real ymin = 0; real ymax = 2.5;

path f = shift(0,1.5)*generic_fcn_plot; // Transform curve by shifting upwards
real c = 3.1;    // x location of tangency
real c_time = times(f, c)[0];
pair c_point = point(f,c_time);  // Point of tangency
pair d = dir(f, c_time);    // Direction of tangent line
real t_line_fcn(real x) { return (d.y/d.x)*(x-c_point.x) + c_point.y; }
path t_line = graph(t_line_fcn, c-0.75, c+0.75);

draw(f, BOLD_COLOR);
draw(t_line, HIGHLIGHT_COLOR);
dotfactor = 4;
dot("$y-f(c)=f'(c)\cdot(x-c)$", c_point, 2*S, HIGHLIGHT_COLOR);

real[] T = {c};
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks("%", T, Size=2pt),
      Arrows(TeXHead));
labelx("$c$", c);
yaxis(ymin=ymin-0.2, ymax=ymax+0.2,
      Arrows(TeXHead));
```

Line 7's `generic_fcn_plot` is a `path`. It joins some points with smooth curves, using the `..` operator. (I often use this path so I included a copy in `jh.asy` as `GENERIC_FCN_PLOT`.) Earlier, when we drew a vertical asymptote line we instead connected two points with a `--` operator, which gives a line segment. There are other connectors but these two are the most common.

In line 17, *Asymptote*'s `dir(..)` command gives the direction of the tangent line as a unit vector. The two lines after that produce its graph. As part of this, line 18 uses that `d.y` is the second component of the pair `d` and `d.x` is its first component, so the tangent line's slope is the ratio `d.y/d.x`.

As to line 15's `c_time = times(f, c)[0]`, *Asymptote* joins the points with piecewise cubic Bézier curves, just as METAFONT and METAPOST do. These curves are parametrized by a variable called 'time'. By definition, the initial point $(0,-0.25)$ is at time 0, the next point $(1,0.35)$ is at time 1, etc. (To forestall any confusion: the time has nothing to do with the first coordinate, it comes from when the

point is specified in the path.) Intermediate points have intermediate times. This illustrates, showing some times.



The `times(..)` command returns an array of times where the path intersects the vertical line $x = c$. We extract the first one (in this case the only one) with the `[0]`. Then line 16's `c_point = point(f,c_time)` returns that point as a `pair`. (Incidentally, the time points need not be evenly spaced on a curve, meaning that there may be a different arc length between $t = 2.0$ and 2.5 than there is between $t = 2.5$ and 3.0.)

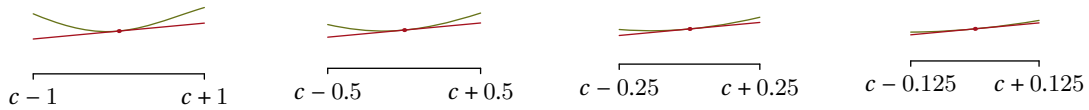
The source for the prior graphic shows two useful aspects of *Asymptote* that are new.

```
// zoom_times.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

unitsize(1cm);
real xmin = 0; real xmax = 5;
real ymin = 0; real ymax = 2.5;

path f = shift(0,1.5)*GENERIC_FCN_PLOT;  // Shift it up by 1.5 y units

draw(f, BOLD_COLOR);

dotfactor = 4;
real[] times = {1, 2, 2.5, 3, 4};  // Array of reals
for (real t : times) {
  pair pt_t = point(f, t);  // Point on curve at that time
  dot(format("$%0.02f$",t), pt_t, S, HIGHLIGHT_COLOR);
}

xaxis(xmin=xmin-0.2, xmax=xmax+0.2, Arrows(TeXHead));
yaxis(ymin=ymin-0.2, ymax=ymax+0.2, Arrows(TeXHead));
```

The first of those is in line 19. The `format("$%0.02f$",t)` turns the floating point number t into the string used in the label.

The other is in lines 17 through 20, where the code has an iteration. One strength of *Asymptote* is that it is a standard programming language, with clean constructs that are like those you use in other languages in your daily work. This iteration is over an array but an integer iteration `for(int i=0; i<4; ++i)` is in the next example.

That next example shows zooming in on the point of tangency in four steps. Here's the output.



The source `asy/zoom_iterate.asy` is more complex than the others that we have seen. One reason is that this one file produces four pictures, so that we needn't maintain four separate `.asy` files with lots of overlap. The four output files are produced in the loop between lines 17 and 47. Line 18 creates a new `picture` and line 46 outputs it. The files are named `zoom_iterate000.pdf` ... `zoom_iterate003.pdf`; the form of that name is given by the string `OUTPUT_FN`.

```
// zoom_iterate.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

path f = GENERIC_FCN_PLOT;  // Shorter to type
real c = 3.1;

// Find f(c) on f and get f'(c)
real c_time = times(f, c)[0];
pair c_point = point(f,c_time);
pair d = dir(f, c_time);
real t_line_fcn(real x) { return (d.y/d.x)*(x-c_point.x) + c_point.y; }

string OUTPUT_FN = "zoom_iterate%03d";
for (int i=0; i<4; ++i) {
  picture pic;  // Generate a new picture
  size(pic, 3cm, 0); // Will be 3cm wide, scaling units to make it so

  // Zoomed-in window spans c minus delta to c plus delta
  real delta = 1/2^(i);
  real xmin = c-delta; real xmax = c+delta;

  // Limits of f and tangent line to show
  real left_time = times(f, xmin)[0];
  real right_time = times(f, xmax)[0];
  path f_shown = subpath(f, left_time, right_time);
  path t_line = graph(t_line_fcn, xmin, xmax);

  // Shift f and tangent line close to x-axis, then draw
  transform f_trans = shift(0, 0.5*delta)*shift(0, -1*c_point.y);
  draw(pic, f_trans*f_shown, BOLD_COLOR);
  draw(pic, f_trans*t_line, HIGHLIGHT_COLOR);
  dotfactor = 3;
  dot(pic, f_trans*c_point, HIGHLIGHT_COLOR);

  // Just the x axis
  real[] T = {xmin, xmax};
  xaxis(pic, xmin=xmin, xmax=xmax,
        RightTicks("%", T, Size=2pt));
  labelx(pic, format("$c-%03f$",delta), xmin);
  labelx(pic, format("$c+%03f$",delta), xmax);

  // Produce PDF output file
  shipout(format(OUTPUT_FN,i), pic);
}
```

Besides using a single input to create multiple output files, there are two other things that are new here. One is line 19's `size(pic, 3cm, 0)`. This makes each output graphic be three centimeters wide and as tall as required, setting the size of the x and y units as needed to get that width. The result is a zooming-in on successively shorter intervals of the x axis.

The other new thing is that rescaling the units to make the entire figure three centimeters wide would put the plotted function very far above the x axis. So we have moved the function down near the axis. This transformation applies not just to the function but also to the tangent line and to the point $(c, f(c))$, so we have broken this transformation out as a separate thing, in line 32. Transformations are applied with the star operator, as on lines 33, 34, and 36.

In the next section we will see one more thing about paths, that if a path is closed then we can fill it.

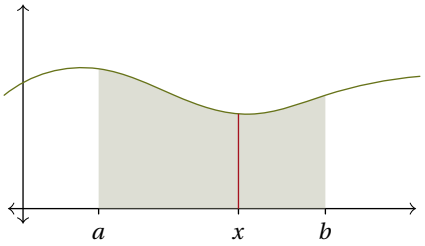
Pens When you draw something you need to specify some properties, such as its color or thickness if you are drawing a curve, or the font if you are writing text. *Asymptote* binds those properties together as

a pen.

This source gives a picture showing the area computed with $\int_a^b f(x)\,dx$.

```
// integral.asy
settings.outformat="pdf";
import "../../../asy/jh.asy" as jh;

import graph;

unitsize(1cm);
real xmin = 0; real xmax = 5;
real ymin = 0; real ymax = 2.5;

path f = shift(0,1.5)*GENERIC_FCN_PLOT;
real a = 1; real b = 4; // Limits of integration
real x = 2.85;    // Location of dx-thick slice
real x_time = times(f, x)[0];

// Give the sides of the region of interest
path left_side = (a,0)--(a,ymax);
path right_side = (b,0)--(b,ymax);
path bottom = (0,0)--(xmax,0);
path region = buildcycle(left_side, f, right_side, bottom);  // Return the boundary

// Draw those
fill(region, NEUTRAL_COLOR+opacity(0.5));
draw(f, BOLD_COLOR);
draw((x,0)--point(f,x_time), HIGHLIGHT_COLOR+squarecap);

// Make the axes
real[] T = {a, b, x};
xaxis(xmin=xmin-0.2, xmax=xmax+0.2,
      RightTicks("%", T, Size=2pt),
      Arrows(TeXHead));
labelx("$a$", a);
labelx("$x$", x);
labelx("$b$", b);
yaxis(ymin=ymin-0.2, ymax=ymax+0.2,
      Arrows(TeXHead));
```

Here is the resulting graphic.



The `buildcycle(left_side, f, right_side, bottom)` on line 20 is new. It takes paths surrounding the region of interest and constructs the path that is the region's boundary. (A more common way to make a cyclic path is to end with `cycle`, as with `path triangle = (0,0)--(0,1)--(1,0)--cycle`.)

Then line 23's `fill(region, NEUTRAL_COLOR+opacity(0.5))` covers the region using a pen that, in addition to its color, allows some of the material behind it to show through. Note that some PDF viewers have trouble with opacity so your results may vary but one viewer that gives good results is Adobe's Reader.

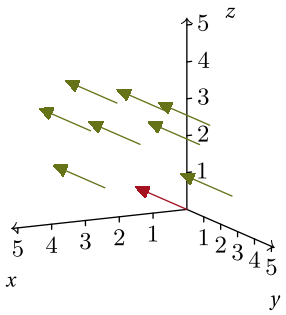
The *Asymptote* reference gives many options for pens. Another is on line 25 where we draw the slice using the pen `HIGHLIGHT_COLOR+squarecap` so that the slice bottom lies flat on the axis.

Chapter 4: 3D

A strength of *Asymptote* is its ability in three dimensions. It can easily draw what you want to draw. That includes doing projections, which can be tricky to get right by eye.

In addition, you can choose to make these graphics manipulable, so that you can use your mouse to turn them around or peek under them, and in general have an explore. This is great for a Calculus lecture so it is what I'm showing here. (Note that only some PDF viewers let you manipulate. For instance, Adobe's Reader works but the ones embedded in web browsers do not. To test your reader just click on the graphic below. You may be asked to let the reader run the code that does the manipulation.)

We start with a picture showing the displacement vector $(2,1,1)$ at a number of initial points.



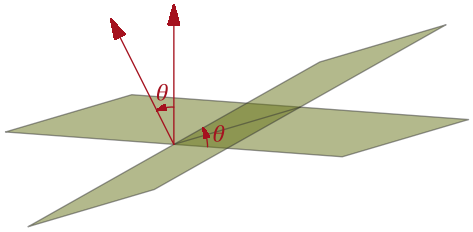
The input code shows two new things. First, to scatter the vectors about, in lines 34–36 they get a randomly-chosen initial point. The randomization uses the seed from line 24. To find that number I uncommented lines 21–23 and commented out line 24, compiled the `.asy` file a couple of times until I got a scatter that I liked, and then I froze the seed. With that, the loop in lines 33–44 creates a randomly placed vector and draws it if the entire vector will show, until there are such eight vectors.

```
// vectors.asy
settings.outformat = "pdf";
settings.tex = "pdflatex";  // For compiling in-pic text
settings.prc = true;  // Manipulable in a PDF file
settings.render = 0;  // Needed for "poster" image, i.e., picture to click on
import "../../../asy/jh.asy" as jh;

import fontsize;

import graph3;
projection default_projection = orthographic(1,2,0.5,up=Z);
currentprojection = default_projection;
currentlight = nolight;  // Avoid shadows

unitsize(0.5cm);
real xmin = 0; real xmax = 5;
real ymin = 0; real ymax = 5;
real zmin = 0; real zmax = 5;

// Randomize: uncomment these to get a good seed, then use it in srand(..seed..).
// int secs = seconds();
// write(format("Seed for srand is %d",secs));
// srand(secs);
srand(1716139271);

// Draw the vector in canonical position
triple displacement = (2,1,1);
draw((0,0,0)--displacement, HIGHLIGHT_COLOR, Arrow3(DefaultHead2));

// Draw more of the same vector, offset from the origin
int num_vector_target = 8;
int num_vectors = 0;
while (num_vectors < num_vector_target) {
  real x_init = xmin+(xmax-xmin)*unitrand();
```

```
  real y_init = ymin+(ymax-ymin)*unitrand();
  real z_init = ymin+(ymax-ymin)*unitrand();
  if ((x_init+displacement.x <= xmax)  // Only draw if vector falls in picture
      && (y_init+displacement.y <= ymax)
      && (z_init+displacement.z <= ymax)) {
    num_vectors = num_vectors+1;
    triple init = (x_init,y_init,z_init);
    draw(init--(init+displacement), BOLD_COLOR, Arrow3(DefaultHead2));
  }
}

// Axes; note it says OutTicks instead of LeftTicks or RightTicks
xaxis3("{\scriptsize $x$}", xmin=xmin, xmax=xmax+0.2,
       OutTicks(Step=1, OmitTick(0), Size=2pt),
       Arrow3(TeXHead2));
yaxis3("{\scriptsize $y$}", ymin=ymin, ymax=ymax+0.2,
       OutTicks(Step=1, OmitTick(0), Size=2pt),
       Arrow3(TeXHead2));
zaxis3("{\scriptsize $z$}", zmin=zmin, zmax=zmax+0.2,
       OutTicks(Step=1, OmitTick(0), Size=2pt),
       Arrow3(TeXHead2));
```

However, the really new stuff is the 3D stuff. It is surprisingly like the 2D constructs that we have seen. Line 10's `import graph3` gives access to *Asymptote*'s 3D routines, and extends them to axes and graph plotting. Some things are new, such as that instead of `pair`'s you want `triple`'s, and instead of `xaxis(..)` you say `xaxis3(..)`. But much of it is at least similar. (Lines 11 and 12 give the projection, essentially setting the location of the camera that is taking this picture. Even if we use a reader that allows us to manipulate the image, we still need a starting view.)

We next see something genuinely different from a 2D context, surfaces. This graphic illustrates that the angle between two intersecting planes is the same as the angle between their normal vectors.




```
// planes.asy
settings.outformat="pdf";
settings.tex="pdflatex";
settings.prc = true;
settings.render = 0;
import "../../../asy/jh.asy" as jh;

import graph3;
projection default_projection = orthographic(3,1.5,0.5,up=Z);
currentprojection = default_projection;
currentlight = nolight;

size3(0,5cm,0,keepAspect=true);  // Make y=5cm tall

// Define surfaces
real rotation_degs = 30;
path3 xy_edge = (0,0,0)--(3,0,0)--(3,4,0)--(0,4,0)--cycle;
surface xy = surface(xy_edge);
transform3 p_t = shift((0,2,0))*rotate(rotation_degs, X)*shift((0,-2,0));
path3 p_edge = p_t*xy_edge;
surface p = p_t*xy;
```

```
// Draw surfaces
draw(xy, surfacepen=figure_material);
draw(xy_edge, boundary_pen);
draw(p, surfacepen=figure_material);
draw(p_edge, boundary_pen);

// Draw the line of intersection
draw((0,2,0)--(3,2,0), boundary_pen);

// Normal vectors
path3 n = (0,0,0)--(0,0,1.5);
path3 n_xy = shift((3,2,0))*n;
path3 n_p = p_t*shift((3,2,0))*n;
draw(n_xy, HIGHLIGHT_COLOR, Arrow3(DefaultHead2));
draw(n_p, HIGHLIGHT_COLOR, Arrow3(DefaultHead2));

// Arc showing angle between normals
triple v1 = (0,0,0.4);   // Radius of the arc
triple v2 = rotate(rotation_degs,X)*(v1);
path3 n_angle_arc = arc(O, v1, v2);
draw("$\theta$", shift((3,2,0))*n_angle_arc,
     HIGHLIGHT_COLOR, Arrow3(DefaultHead2));  // Angle between normals
draw("$\theta$", shift((3,2,0))*rotate(-90,X)*n_angle_arc,
     HIGHLIGHT_COLOR, Arrow3(DefaultHead2)); // Angle between planes
```

This code spotlights the power of transforms. We don't have to give the equations of the planes or specify their normals. Instead, in line 17 we define the edge of the horizontal plane region and then in line 18 we create that as a surface. To get the other plane, the one at an angle, we make a transform `p_t` in line 19 that basically rotates by `rotation_degs` about the x axis. The new plane with its edge and its normal vector then comes from applying that transform to the horizontal plane, its edge, and its normal.

In lines 24 and 26 we use `figure_material` to give the surfaces color. We will reuse this later so the definition is in `jh.sty`; see lines 26–32 in the Appendix. This involves `opacity(..)` and note that you can indeed see through the planes.

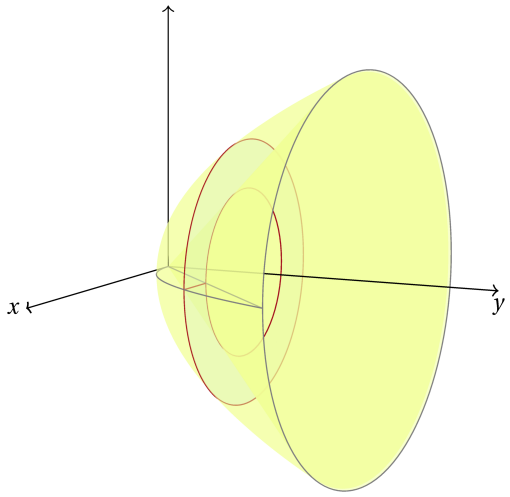
Finally, in lines 40–46 we take advantage of one of *Asymptote*'s many helper functions to find and draw the arc of the angle between the planes and the normals.

Our last graphic is from the Calculus I lecture on the volume of a surface of revolution, specifically using slices that are washers. We start with the xy plane area between $y = x^2$ and $y = x$, here defined as line 33's `pth`.

```
// washer.asy
settings.tex = "pdflatex";
settings.outformat = "pdf";
settings.render = 8;  // Tweaked this until poster pic looked better
settings.prc = true;
import "../../../asy/jh.asy" as jh;

import graph3;
projection default_projection = orthographic(3,1.5,0.5,up=Z);
currentprojection = default_projection;
currentlight = nolight;

size3(0,5cm,0,keepAspect=true);

real square(real x) {return x^2;}
real ident(real x) {return x;}
path f = graph(square, 0, 1);
path fa = graph(ident, 0, 1);

real c = 0.4;  // Pt on y axis where slice is taken
real delta_y = 0.2;  // Thickness of washer

// Make the washer
```

```
transform3 washer_t = shift(0,c,0)*rotate(90, X);
surface washer = washer_t*surface(reverse(scale(sqrt(c))*unitcircle) ^^ scale(c)*unitcircle);

// Draw the washer
draw(washer_t*scale3(c)*unitcircle3, HIGHLIGHT_COLOR);
draw(washer_t*scale3(sqrt(c))*unitcircle3,  HIGHLIGHT_COLOR);
draw(washer, surfacepen=slice_material, light=nolight);

// Draw the xy-plane area
path3 pth = path3(fa&f);  // Two paths joined with `&'
draw(pth, gray(0.5));
draw((c,c,0)--(sqrt(c),c,0),  HIGHLIGHT_COLOR);
draw(circle((0,1,0), 1, normal=Y), gray(0.5));

// Draw surface of revolution formed by xy-plane area
surface region = surface(pth, c=0, axis=Y);
draw(region, surfacepen=figure_material, light=nolight);

// Axes
xaxis3(Label("$x$",position=EndPoint, align=W),
       0,1.5, black, Arrow3(TeXHead2));
yaxis3(Label("$y$"),
       0,1.75, black, Arrow3(TeXHead2));
zaxis3(Label(""),\
       0,1.25, black, Arrow3(TeXHead2));
```

In line 39 *Asymptote* rotates that area about the y axis, giving the 3D figure.



Include the graphic in a LaTeX file To illustrate including 3D output in your document we use `asy/vectors.asy`. Create the LaTeX file `main_3d.tex`.

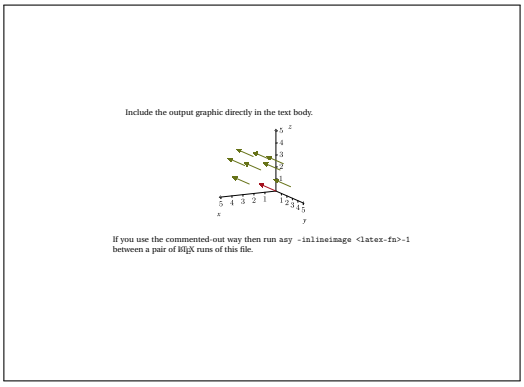
```
% main_3d.tex  LaTeX demonstraton file showing Asymptote 3D, Jim Hefferon
\documentclass{article}
\usepackage{graphicx}
\graphicspath{ {asy/} }  % Path to poster graphic; dirs in curly braces,  trailing slash
\usepackage{asymptote}
\def\asydir{asy}  % No trailing slash
% For non-manipulable 3D figures, just compile with "asy vectors" and
% include the PDF output as with 2D figures.
% For manipulable 3D figures:
% (1) Compile the .asy file with "asy -inlineimage vectors".
% It outputs a number of files, including .tex and .pre files,
% and the .pdf "poster" that you click on to start the manipulation.
```

```
% (2) Include the .pre line here along with one of the lines in center below
\input asy/vectors.pre

\begin{document}
Include the output graphic directly in the text body.
\begin{center}
%   \asyinclude{asy/vectors.asy}
  \input asy/vectors.tex %
\end{center}
If you use the commented-out way then run
\texttt{asy -inlineimage <latex-fn>-1}
between a pair of \LaTeX{} runs of this file.
\end{document}
```

Run LaTeX.

```
jim@millstone:~/Documents/asy_tut/src/chapter4$ pdflatex main_3d
```

(As outlined in the document, another option that many people prefer is to instead use `\asyinclude{..}`. For that you first run LaTeX, then go to the `asy/` subdirectory and run `asy -inlineimage main_3d-1` and then return to the starting to directory to run LaTeX on the above document a second time.) Here is a shot of the output.



In that output PDF, you should be able to click on the poster graphic to bring up the manipulable graphic, if your reader supports that.

Appendix: File with defaults

Rather than copy and paste code common across graphics, we can put them in a separate file and import them. In the earlier *Asymptote* sources the lines

```
import "../../../asy/jh.asy" as jh;
```

will run the commands in the file `../../../asy/jh.asy` and also make available the routines and data defined there.

Here is the file's contents. First, it sets the fonts and changes the default font size. (The smaller size leaves more room for graphic elements and also helps the graphics have a visually cohesive identity distinct from the document's body.) Then it defines a color scheme.[1] Last, it gives the `material` defaults for 3D graphics.

```
// Common definitions for asy_tut

// Set up LaTeX for each label
texpreamble("\usepackage{mathtools}
             \usepackage[utf8]{inputenc}
             \usepackage[osf,scaled=.92,loosest]{heuristica}
             \usepackage[heuristica,vvarbb,bigdelims]{newtxmath}
             \usepackage[T1]{fontenc}
             \renewcommand*\oldstylenums[1]{\textosf{#1}}");
// Change default size of fonts
import fontsize;
defaultpen(fontsize(9pt));  // Like LaTeX \small

// Colors: Morning in Vermont muted by Michelle Delapenha
pen HIGHLIGHT_COLOR = rgb("A6121F"); // Ecstatic Red
pen BACKGROUND_COLOR = rgb("8FB6D9"); // Blue Bell
pen BOLD_COLOR = rgb("677319"); // Fern Frond
pen LIGHT_COLOR =rgb("A1A60F"); // Fistfull of Green
pen NEUTRAL_COLOR = rgb("BDBFAA"); // New Neutral

// Generic 2D function
path GENERIC_FCN_PLOT = (-0.25,0)..(1,0.35)..(2,0)..(3,-0.25)..(4,0)..(5.25,0.25);

// Default materials for 3D surfaces
import three;
material figure_material = material(diffusepen=BOLD_COLOR+opacity(0.5),
                                    emissivepen=BOLD_COLOR+white,
                                    specularpen=BOLD_COLOR+white);
material slice_material = material(diffusepen=BACKGROUND_COLOR+opacity(0.25),
                                   emissivepen=BACKGROUND_COLOR,
                                   specularpen=BACKGROUND_COLOR);
pen boundary_pen = gray(0.2)+opacity(0.5);
```

[1] Credit to `color.adobe.com` user Michelle Delapenha for this scheme.