Chapter 1

Welcome to Sage!

Mathematics is something that we do, not something that we watch. Therefore, the best way to learn any branch of mathematics is to actually *get started* and give things a try. The same is true for learning Sage! Just connect to the Sage "Cell Server" and dive right on in, trying the examples of the next few pages.

`https://sagecell.sagemath.org/`

1.1. Using Sage as a Calculator

First off, you can always use Sage as a simple calculator. For example, if you type `2+3` and click "evaluate" then you will learn the answer is 5.

The expressions can become as complicated as you like. To calculate the amount on a simple interest loan at 6% per year for 90 days, and principal $ 900, we all know the formula to be $A = P(1 + rt)$ so we would just type in

```
900*(1+0.06*(90/365))
```

and click "evaluate". You will learn that the answer is 913.315068493151, or $ 913.32 after rounding to the nearest penny.

Notice that the symbol for addition is the plus sign, and the symbol for subtraction is the hyphen. The symbol for multiplication is the asterisk, and the symbol for division is the forward slash (that's the one found with the question mark, not the backslash.) These are the same rules used by MS-Excel and many computer languages, including C, C++, Perl, Python, and Java.

For compound interest, we'd need to take exponents. The symbol for exponents is the caret, found above the number six on most keyboards. It looks like this "ˆ".

Let's consider 12% compounded annually on a signature loan for 3 years, and a principal of $ 11,000. We all know the formula to be $A = P(1 + i)^n$, so we would just type in

```
11000*(1+0.12)^3
```

and click "evaluate". You will learn that the answer is 15454.2080000000, or \$ 15,454.21 after rounding to the nearest penny. By the way, instead of clicking "evaluate," you can just press shift-enter on the keyboard, and it will do the same thing.

Warning: It is very important not to enter a comma in large numbers when using Sage. You have to type `11000` and not `11,000`

For those who don't like the caret, you can use two asterisks in a row.

```
11000*(1+0.12)**3
```

which is actually a throw-back to the historical programming language `FORTRAN` which was most popular[1] during the 1970s and 1980s.

What if you make a mistake? Let's say that I really meant to say \$ 13,000 and not \$ 11,000. Click on the mistake, and correct it using the keyboard, and change the 11 into 13. Now click "evaluate" again, and all is well.

Grouping Symbols:

When there are multiple sets of parentheses in a formula, sometimes mathematicians use brackets as a type of "super parentheses." As it turns out, Sage needs the brackets for other things, like lists, so you have to always use parentheses for grouping inside of formulas.

For example, let's say you need to evaluate

$$550\frac{\left[1-(1+0.05)^{-30}\right]}{0.05}$$

So you should not type

```
550 [ 1 - (1+0.05)^(-30) ]/0.05
```

but rather

```
550 ( 1 - (1+0.05)^(-30) )/0.05
```

where the brackets have become parentheses.

Some very old math books use braces { and } as a sort of auxiliary also to the parentheses and brackets. These too, if they are for grouping in a formula, must become parentheses. As it turns out, Sage, as well as modern mathematical books, use the braces { and } to denote sets.

By the way, the above formula was not artificial. It is the value of a loan at 5% compounded annually for 30 years, with an annual payment of \$ 550. The formula is called "the present value of an annuity."

[1] An irrelevant historical aside: Because rewriting software is a painful and time-consuming process, many important pieces of scientific software remained written in `FORTRAN` until a few years ago, and likewise the same is true of business software in `COBOL`. Each new programming language incorporates features of the previous generation of languages, to make it easier to learn, and accordingly one sees some truly ancient seeds of dead languages once in a while.

Three Mistakes that are 90+% of My Errors in Sage:

There are three mistakes that I make a lot when using Sage. For example, if you want to say $11,000x + 1200$ then you have to type

```
11000*x+1200
```

The first error that I sometimes make is that I leave out the asterisk between `11000` and `x`. In Sage, you must include that asterisk, as that is the symbol of multiplication. The second error that I'll often add a comma inside the `11000`, but it is not acceptable in Sage to write `11,000` for `11000`. The third error is that I'll have mismatched parentheses. Any Sage expression should have the same number of `(`s as it has of `)`s—no more and no less.

A more complete list is given as Appendix A, "What to Do When Frustrated," on Page 295.

1.2. Using Sage with Common Functions

Now I'll discuss how Sage works with square roots, logarithms, exponentials, and so forth.

Square Roots:

The standard "high school" functions are also built-in. For example, type

```
    sqrt(144)
```

then click "evaluate" and you'll learn that the answer is 12. From now on, I'm not going to say "click evaluate", because repeating that might get tiresome; I'll assume that you know you have to do that. Since Sage likes exact answers, try

```
    sqrt(8)
```

and get `2*sqrt(2)` as your answer. If you really need a decimal, try instead

```
    N( sqrt(8) )
```

and obtain

```
    2.82842712475.
```

which is a decimal approximation.

The `N()` function, which can also be written `n()`, will convert what is inside the parentheses to a real number, if possible. Usually that is a decimal expansion, thus unless what is inside the parentheses is an integer[2] then it will be, necessarily, an approximation. Sage will assume that all decimals are mere approximations, so for example

[2]To be mathematically correct, I should say "an integer or a fraction with denominator writable as a product of a power of 5 and a power of 2." For example, 25ths, 16ths, and 80ths can be written exactly with decimals, where as 3rds, 15ths, and 14th cannot. Observe that $25 = 5^2$ and $16 = 2^4$ as well as $80 = 2^4 \times 5$. Those denominators have only 2s and 5s in their prime factorization. Meanwhile, $3 = 3$ and $15 = 5 \times 3$ while $14 = 7 \times 2$. As you can see, those denominators have primes other than 2 and 5 in their prime factorization. If you find this interesting, you should read "Using Sage to work with Integers" on Page 145.

```
sqrt(3.4)
```

will evaluate to `1.84390889145858`, without need of using `N()`.

Higher Order Roots:

Higher order roots can be calculated like exponents are. For example, to find the sixth root of 64, do

```
64^(1/6)
```

and obtain that the answer is 2.

The Special Constants π and e:

Frequently in math we need the special constants π and e. It turns out that π is built into Sage as "pi" (both letters lower case) and e is built in as "e" (again, lower case).

You can do things like

```
numerical_approx(pi, digits=200)
```

to get many digits of pi, or likewise

```
numerical_approx(sqrt(2), digits=200)
```

to get a high-accuracy expansion of $\sqrt{2}$. In this case, we get 200 digits. You can also abbreviate with

```
N(sqrt(2), prec=200)
```

as we did earlier. That's what the `N()` or `n()` function (they are the same) is an abbreviation of, namely "numerical approx."

Tab Completion:

Now would be a good time to explain a neat feature of Sage. If you are typing a long command, like "`numerical_approx`" and part way through you forget the exact ending (is it approximation? approximations? appr?) then you can hit the tab button. If only one command in the entire library has that prefix, then Sage will fill in the rest for you. If it does not, then you get a list of suggestions. It is a very useful feature when you cannot remember exact commands. We will see other examples of built-in help features in Section 1.10 on Page 47.

There's another keyboard shortcut that I like. While I mentioned it before, if you are tired of clicking "evaluate" all the time, you can just press Shift and Enter at the same time. This does the same thing as clicking "Evaluate."

Is Sage Case-Sensitive?

You've probably had a situation in life where you entered your password correctly, but discovered that it was rejected because you had the wrong capitalization. For example, perhaps you've left the "CAPS-LOCK" key on. In any event, Sage does think of `Pi` as different from `pi` and `Sin` as different from `sin`.

The only four exceptions known to me are i and `n()`, as well as `True` and `False`. We will not make extensive use of `True` and `False` until Chapter 5, but it is harmless to mention that you can also enter them as `true` and `false`, and Sage recognizes these as the same. You may have heard that $\sqrt{-1}$ is often referred to as "the imaginary number" and is denoted i.

Next, you can represent $\sqrt{-1}$ as either `i` or `I` and either way Sage will know what you meant, namely $\sqrt{-1}$. If you've never heard of imaginary numbers or i, then you can safely ignore this fact. Lastly, you can write `n(sqrt(2))` or `N(sqrt(2))` and Sage treats those as identical.

So far as I am aware, these easements only apply to $\sqrt{-1}$ and `n()`, as well as `True` and `False` as it turns out; in all other cases, capitalization matters.

Exponentials:

Just as

```
2^3
```

gives you 2^3 and likewise

```
3^3
```

gives you 3^3, if you want to say e^3 then just say

```
e^3
```

and that's fine. Or for a decimal approximation you can do

```
N(e^3)
```

Also, it is worth mentioning sometimes books will write

$$\exp(5 \cdot 11 + 8) \text{ instead of } e^{5 \cdot 11 + 8}$$

and Sage thinks that's just fine. For example,

```
exp(5*11+8) - e^(5*11+8)
```

evaluates to 0. Don't forget the asterisk between the 5 and the 11.

Logarithms:

Of course, Sage knows about logarithms—but there's a problem. There are several types of logarithm, including the common logarithm, the natural logarithm, the binary logarithm, and the logarithm to any other base you feel like using. In high school "log" refers to the common logarithm, and "ln" to the natural logarithm. However, after calculus "log" refers to the natural logarithm. Since Sage was mainly meant for university-and-higher level work, then it is only natural they chose to use the natural logarithm for "log."

So to find the natural logarithm of 100 type

```
    N( log(100) )
```

for the common logarithm of 100 type

```
    N( log(100,10) )
```

and for the binary logarithm of 100 type

```
N( log(100,2) )
```

which of course generalizes. For example, to find the logarithm of 100 taken base 42, type

```
N( log(100,42) )
```

Note that Sage is quite good at getting exact answers. Try

```
log ( sqrt(100^3), 10)
```

and you will obtain 3, the exact answer.

A Financial Example:

Suppose you deposit \$ 5000 in an account that earns 4.5% compounded monthly. Perhaps you are curious when your total will reach \$ 7000. Using the formula $A = P(1+i)^n$, where P is the principal, A is the amount at the end, i is the interest rate per month, and n is the number of compounding periods (number of months), we have

$$\begin{aligned} A &= P(1+i)^n \\ 7000 &= 5000(1+0.045/12)^n \\ 7000/5000 &= 5000(1+0.045/12)^n \\ 1.4 &= (1+0.045/12)^n \\ \log 1.4 &= \log(1+0.045/12)^n \\ \log 1.4 &= n\log(1+0.045/12) \\ \frac{\log 1.4}{\log(1+0.045/12)} &= n \end{aligned}$$

And so, we type into Sage

```
log(1.4)/log(1+0.045/12)
```

and get the response `89.8940609330801`, so that we know 90 months will be required. Therefore we write the answer 90 months, or even better, 7 years and 6 months.

This is an example of a computer-assisted solution to a problem. In addition to that approach, Sage is also willing to solve the problem from start to finish, with no human intervention. We'll see that on Page 46.

Pure Mathematics and Square Roots:

Here are a couple of notes about square roots that would appeal to math majors, or anyone who wants to work with the complex numbers. Most students will want to skip this subsection, as well as the next one on complex numbers, and continue with either "Using Sage for Trigonometry" on Page 7, or with "Using Sage to Graph 2-Dimensionally" on Page 8.

In the spirit of absolute pomposity, you may know that $(-2)^2 = 4$ as well as $2^2 = 4$. So if you want *all* the square roots of a number you can do

```
sqrt(4,all=True)
```

to obtain

```
    [2, -2]
```
where the square brackets indicate a list. Any sequence of numbers separated by commas and enclosed in square brackets is a list. We'll see other examples of lists throughout the book.

A Taste of the Complex Numbers:

If you know about complex numbers, you can do
```
    sqrt(-4,all=True)
```
to obtain
```
    [2*I, -2*I]
```
where the capital letter I represents $\sqrt{-1}$, the imaginary constant. As I mentioned before, you can use the lowercase `i` instead of the capital one, if you prefer.

While Sage is quite good at complex analysis, and can produce some rather lovely plots of functions of a complex variable, we will not go into those details here. The color plots of complex-value functions will be covered in the electronic-only online appendix to this book "Plotting in Color, in 3D, and Animations," available on my webpage `www.gregorybard.com` for downloading.

1.3. Using Sage for Trigonometry

Some students are unfamiliar with trigonometry, or are in a course that has nothing to do with trigonometry. If so, then they may confidently skip this section and jump to the next one, which begins on Page 8.

The commands for trigonometry in Sage are very intuitive but it is important to remember that Sage works in radians. So if you want to know the sine of $\pi/3$, you should type
```
    sin(pi/3)
```
and you will get the answer `1/2*sqrt(3)`. This is an exact answer, rather than a mere decimal approximation. You will find that Sage is very oriented toward exact rather than approximate answers. Sometimes this is irritating, because if you ask for the cosine of $\pi/12$, then you would type
```
    cos(pi/12)
```
and obtain `1/12*(sqrt(3) + 3)*sqrt(6)` which is especially unsatisfying if you want a decimal. Instead, if you type
```
    N(cos(pi/12))
```
then you will obtain `0.965925826289`, a rather good decimal approximation.

You will discover that Sage is fairly savvy when it comes to knowing when functions will go wrong. In particular, just try evaluating tangent at one of its asymptotes. For example,
```
    tan(pi/2)
```
will produce the helpful answer "Infinity." The "rare" or "reciprocal" trigonometric functions of cotangent, secant, and cosecant, which are important

in calculus but annoying on hand-held calculators, are built into Sage. They are identified as `cot`, `sec`, and `csc`.

The inverse trigonometric functions are also available. They are used just like the trigonometric functions. For example, if you type

```
arcsin(1/2)
```

you will obtain

```
1/6*pi
```

as expected. Likewise

```
arccos(1/2)
```

produces

```
1/3*pi
```

The usual abbreviations are all known by and used by Sage. Here is a complete list:

Math Notation:	Long-form Command:	Short-form Command:
$\sin^{-1} x$	`arcsin(x)`	`asin(x)`
$\cos^{-1} x$	`arccos(x)`	`acos(x)`
$\tan^{-1} x$	`arctan(x)`	`atan(x)`
$\cot^{-1} x$	`arccot(x)`	`acot(x)`
$\sec^{-1} x$	`arcsec(x)`	`asec(x)`
$\csc^{-1} x$	`arccsc(x)`	`acsc(x)`

You can also use Sage to graph the trigonometric functions. We'll do that in the section entitled "Using Sage to Graph 2-Dimensionally," on Page 11.

Converting between Degrees and Radians:

We all remember from trigonometry class that to covert radians to degrees, just multiply by $180/\pi$, and to covert from degrees to radians, just multiply by $\pi/180$. Accordingly, here's what you type, to convert $\pi/3$ to degrees:

```
(pi/3) * (180/pi)
```

produces `60` while

```
60 * (pi/180)
```

produces `1/3*pi`.

The way I like to remember this technique is that a protractor is 180 degrees, and the word "protractor" begins with "p," while π is the Greek "p." For example, 90 degrees is half a protractor, and 60 degrees is a third of a protractor. That's why I can remember that they are $\pi/2$ and $\pi/3$. Likewise, if I see $\pi/6$, then I know that this is one-sixth of a protractor or 30 degrees.

1.4. Using Sage to Graph 2-Dimensionally

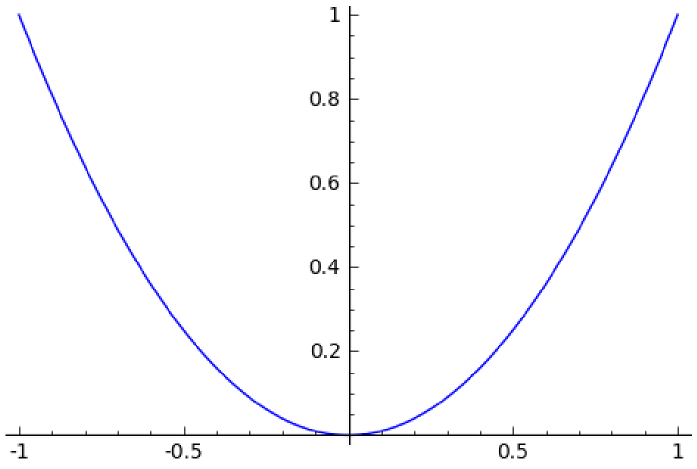
My favorite shape is the parabola, so let's start there. Type

```
plot(x^2)
```

and you get a lovely plot of a parabola going in the range

$$-1 < x < 1$$

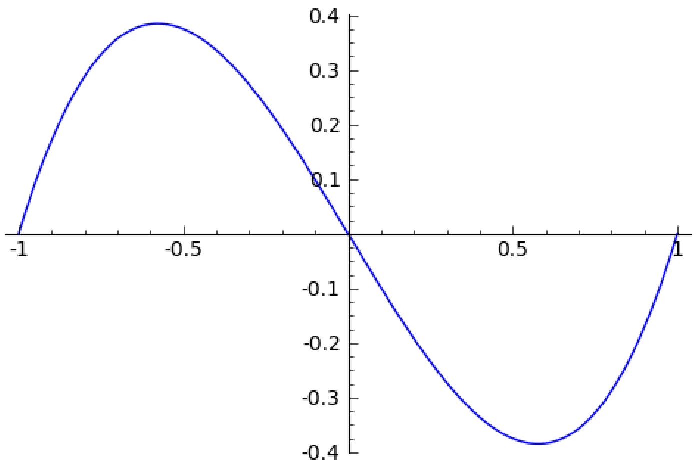
which is the default range for the `plot` command. It will bound the y-values to whatever is needed to show those x-values. Here's a screenshot of what you should see:



Likewise you can do

```
plot(x^3-x)
```

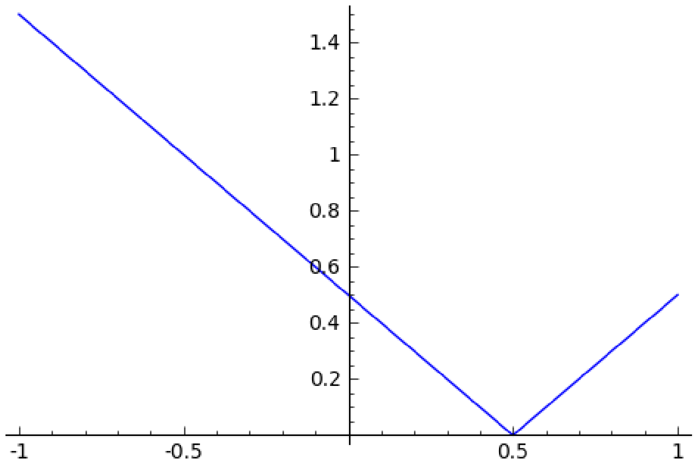
which is nice and visually appealing, as you can see:



For $|x - 1/2|$ you can do

```
plot(abs(x-1/2))
```

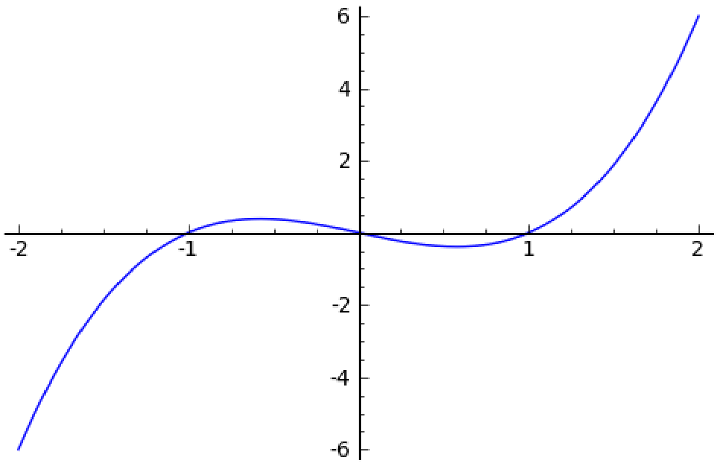
which produces the following screenshot



What if you wanted a different x range? For example, to graph in $-2 < x < 2$ you would type

```
plot(x^3-x, -2, 2)
```

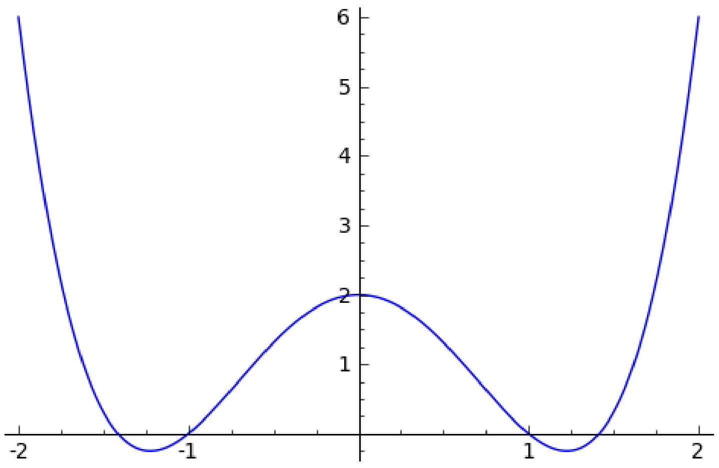
and you get the desired graph, namely:



Now would be a good time to mention that you can save any of these plots to a file after having Sage generate them. Just right-click on the displayed graph in your web browser, and save the file to your computer. You can then import the image into any report or paper that you might be writing. A very cool graph is

```
plot(x^4 - 3*x^2+2,-2,2)
```

but notice the asterisk between 3 and x in " `3*x` ". You will get an error if you leave that out! The plot is



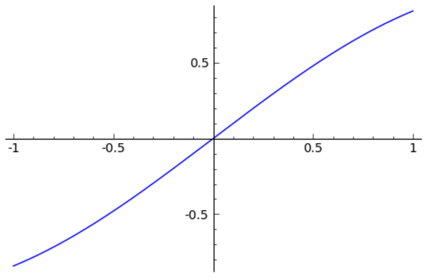
Another minor sticking point is that for $y = \sin(t)$ you have to say

```
plot(sin(x))
```

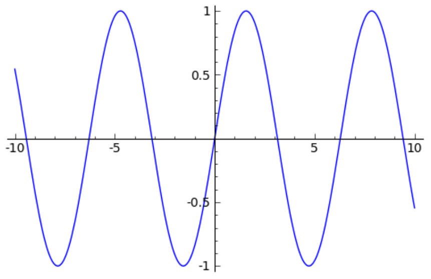
or better yet

```
plot(sin(x),-10,10)
```

because plot is not expecting t, it expects x. The images that you get are



plot(sin(x))



plot(sin(x),-10,10)

In fact if you were to type

```
plot(sin(t))
```

you would see

```
NameError: name 't' is not defined
```

because in this case, Sage does not know what 't' means. An alternative way of handling this is given on Page 104.

1.4.1. Controlling the Viewing Window of a Plot

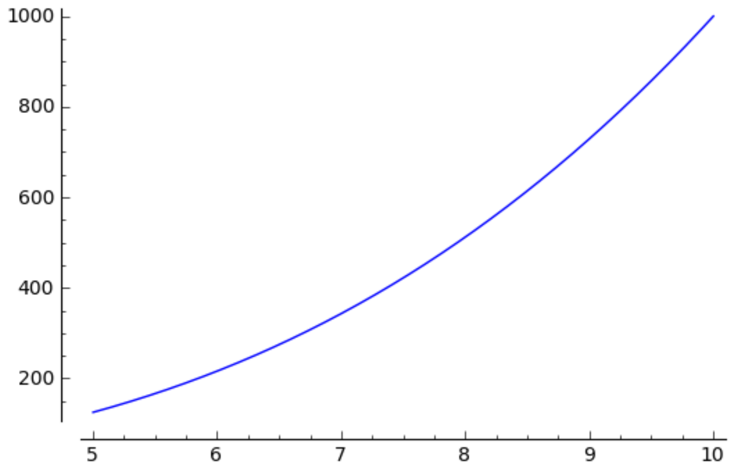
Much of the time, the default viewing window of a graph is going to be exactly what you wanted—but not always. The most common exception is a function with vertical asymptotes. Here we are going to discuss how to adjust the viewing window.

Going "Off the Scale":

Consider the plot of x^3 from $x = 5$ to $x = 10$, which is given by

```
plot(x^3, 5, 10)
```

and as you can imagine, the function goes from $5^3 = 125$ up to $10^3 = 1000$, thus the origin should appear very far below the graph. This is the plot:

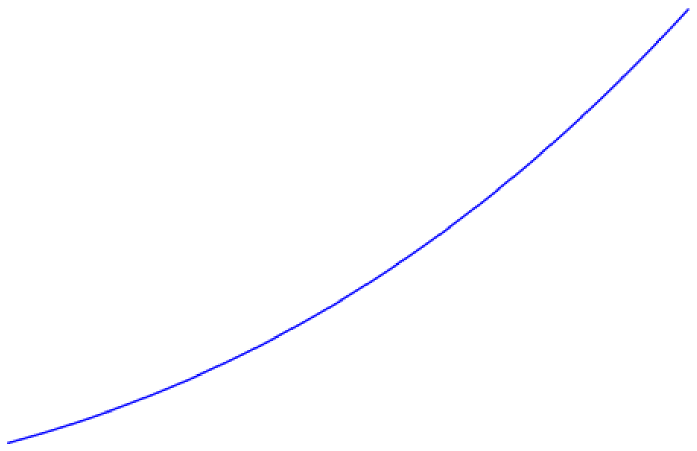


Your hint that the location of the x-axis in the display is not where it would be normally is that the axes do not intersect. This is to tell you that the origin is far away. When the axes do intersect on the screen, then the origin is (both in truth and on the screen) where they intersect.

Also, if you want to, you can hide the axes with

```
plot(x^3, 5, 10, axes=False)
```

and that produces



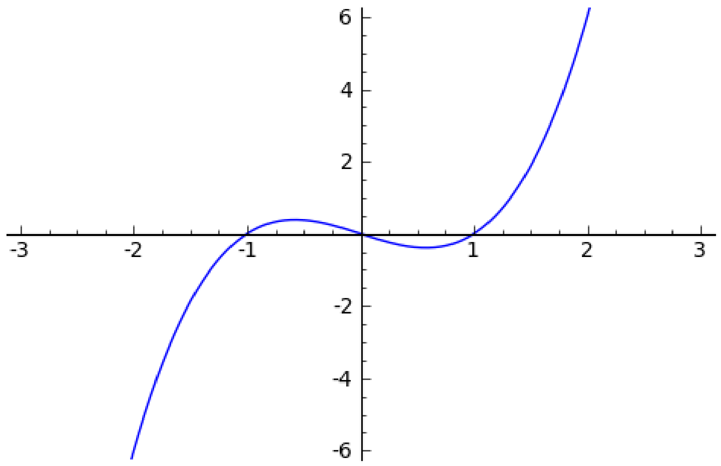
which is considerably less informative.

To Force the Y-Range of a Graph:

If, for some reason, you want to force the y-range of a graph to be constrained between two values, you can do that. For example, to keep $-6 < y < 6$, we can do

```
plot( x^3-x, -3, 3, ymin = -6, ymax = 6)
```

in order to obtain

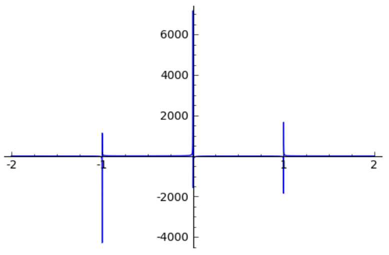


This is important, because normally Sage wants to show you the entire graph. Therefore, it will make sure that the y-axis is tall enough to include every point, from the maximum to the minimum. For some functions, either the maximum or the minimum or both could be huge.

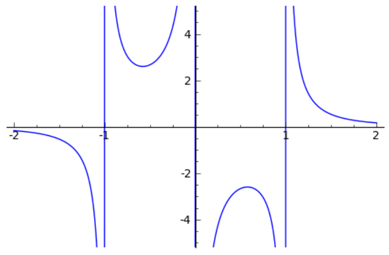
Plots of Functions with Asymptotes:

The way that Sage (and all computer algebra tools) computes the plot of a function is by generating a very large number of points in the interval of the xs, and evaluating the function at each of those points. To be precise, if you want to graph $f(x) = 1/x^2$ between $x = -4$ and $x = 4$, the computer might pick 10,000 random values of x between -4 and 4 find the y values by plugging them into $f(x)$ and then finally drawing the dots in the appropriate spots on the graph.

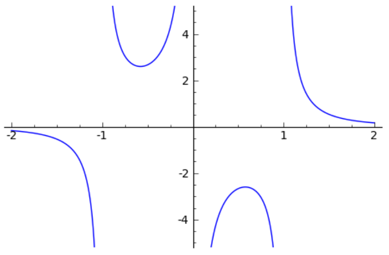
So if the graph has a vertical asymptote, then near that asymptote, the value will be huge. Because of this, when you graph a rational function, be sure to restrict the y-values. For example, compare the following:



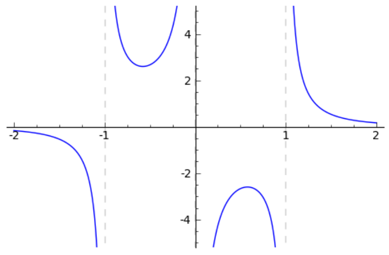
Plot 1



Plot 2



Plot 3



Plot 4

Plot 1:

```
plot(1/(x^3-x), -2, 2)
```

Plot 2:

```
plot(1/(x^3-x), -2, 2,  ymin = -5,  ymax = 5)
```

Plot 3:

```
plot(1/(x^3-x), (x,-2, 2),  detect_poles=True,
          ymin = -5,  ymax = 5)
```

Plot 4:

```
plot(1/(x^3-x), (x,-2, 2),  detect_poles='show',
          ymin = -5,  ymax = 5)
```

As you can see, the first is a disaster. The second one cuts off the very-high and very-low y-values, but it keeps trying to connect the various "limbs" of the graph. When you set "detect poles" to `True`, then it will figure out that the pieces are not connected.

One minor point remains. Did you notice in the four plots above, that `'show'` is in quotes, but `True` is not in quotes? That's because `True` and `False` occur so often in math, that they are built-in keywords in Sage. However, the `'show'` occurs less often, and therefore is not built in, and we must put it in quotes.

1.4.2. Superimposing Multiple Graphs in One Plot

Now we're going to see how to superimpose plots on each other. It turns out that Sage thinks of this as "adding" the plots together, using a plus sign.

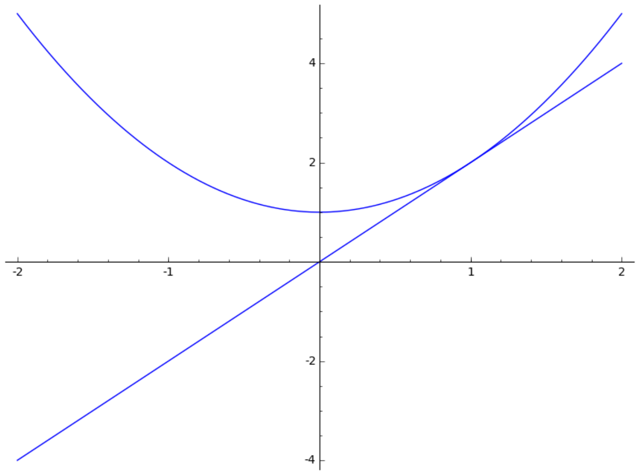
An Example from Calculus: The Tangent Line

Suppose one wants to draw a picture of $f(x) = x^2 + 1$ over the interval $-2 < x < 2$ and the tangent line to that parabola at $x = 1$. Because $f(1) = 2$ and $f'(1) = 2$, we know the line has to go through the point $(1, 2)$ and will have slope 2. It is not hard to compute that the equation of that line is $y = 2x$. The trick is that we want to graph them both at the same time, in the same picture.

The command for this will be

```
plot( 2*x, -2, 2 ) + plot( x^2+1, -2, 2 )
```

and we will get the image:



As you can see, adding the two plots makes a super-imposition. The plus sign tells Sage to draw the two curves on top of each other. Now, we can do better by adding a dot at the point of tangency—the point $(1, 2)$ and perhaps some gridlines. The command for that will be

```
plot( 2*x, -2, 2, gridlines='minor' ) + plot( x^2+1, -2, 2 ) +
point( (1,2), size=30 )
```

and we will get the image: