

Earlham Tim's Sage 4.3 Intro

A Quick Introduction to Sage 4.3

Tim McLarnan
Earlham College

Getting Started

To use *Sage* from the Notebook interface, you need to connect to a *Sage* server. There are several ways to do this. If you are just playing with *Sage* for the first time, one easy thing is to go to <http://sagenb.org/> (sagenb stands for *Sage* Notebook) and sign up for a free account. Alternatively (and better if you are an Earlham person), you can use Earlham's own *Sage* Notebook server at <https://sage.cs.earlham.edu:8000/>. Notice that that link begins with https, not with http.

If you have installed *Sage* on your machine and want to use your own *Sage* installation, start *Sage* (on the Mac, by opening `/Applications/sage/sage` with the Terminal) and type **notebook()** at the **sage:** prompt. If the Notebook server is already running on your machine, just open a browser window and connect to `http://localhost:8000` .

Once you have opened a new Worksheet, you are ready to go. Type *Sage* commands in the rectangular input cells. Send those commands to *Sage* by hitting shift-Enter. Multiple *Sage* commands can be placed in a single input cell, either on separate lines (gotten by just hitting Return) or separated by semicolons.

New input cells can be created by clicking on the blue line that appears when you mouse above an existing cell. New cells for html input are created by shift-clicking on these bars. There is more detail about text editing in *Sage* at the end of this document.

When you're done working on a Worksheet, it's important to quit it. Do this by clicking the "Save & quit" button or the "Discard & quit" button at the top of the Worksheet. Failure to do this will keep a *Sage* process for this Worksheet running on the server. Once enough of these processes build up, the system will slow to an unusable crawl for everyone, ruining the lives of your friends. This has happened on Earlham's server. **Don't be a cause of added suffering in the world!**

Now let's do some mathematics.

Basic Commands

`+`, `-`, `*` and `/` do what you expect:

```
2+5
```

7

Unlike most calculators, though, *Sage* does operations with fractions exactly.

```
1/2 + 1/3 + 1/4 + 1/5 + 1/6 + 1/7
```

Other arithmetic operations are a^b or $a^{**}b$, which means a^b , and $n.factorial()$ or $factorial(n)$, which means $n! = 1 * 2 * 3 * \dots * n$

```
2^1000
```

```
1071508607186267320948425049060001810561404811705533607443750388370351051
```

```
print(2^1000)
```

```
107150860718626732094842504906000181056140481170553360744375038837  
510511249361224931983788156958581275946729175531468251871452856923  
043598457757469857480393456777482423098542107460506237114187795418  
530464749835819412673987675591655439460770629145711964776865421676  
429831652624386837205668069376
```

```
(355/113)**10
```

```
31789487802830871103515625  
339456738992222314849
```

```
factorial(4)
```

```
24
```

```
4.factorial()
```

```
24
```

```
print(125.factorial())
```

```
188267717688892609974376770249160085759540364871492425887598231508  
315633161359886688293288949592313364640544593005774063016191934138  
978188834575585470555243263755650071317708800000000000000000000000  
000000
```

Is it a surprise that $125!$ ends in so many zeros? Can you find a way to compute how many zeros are at the end of $n!$ for any n ?

The past couple of lines, and the rest of this Worksheet, show an important thing about *Sage*. Some common *Sage* commands are implemented as functions applied to arguments, with syntax like $f(x)$. Many other commands are implemented as methods applied to objects, with the less familiar syntax $x.f()$. Some commands, like **factorial**, can be used either way. There are good software engineering reasons for doing things this way, and those of you who are CS students either know or will learn what they are. For the rest of us, all we need to know is that some commands have one form, and some have the other, and that we need to remember which commands work which way. Of course, if one forgets, one can try it both ways and see what works.

factor factors an integer into primes. Like many *Sage* functions, it can either be called as a function sent the argument n : **factor(n)**, or as a message sent to the object n : **n.factor()**.

```
factor(315)
```

$$3^2 \cdot 5 \cdot 7$$

```
315.factor()
```

$$3^2 \cdot 5 \cdot 7$$

```
20.factorial() - 12.factorial()
```

```
2432902007697638400
```

The result of the previous computation in *Sage* is denoted `_`. Thus, if we want to factor the number we just computed, we can just say

```
factor(_)
```

```
210 · 35 · 52 · 7 · 113 · 41976119
```

Notice that this result is pretty remarkable: *Sage* has asserted (and very quickly) that 41976119 is a prime.

Decimal Approximations

Sage's usual mode of operation is to do exact arithmetic with no roundoff or decimal approximation.

```
2^30/3^20*sqrt(2)
```

$$\frac{1073741824}{3486784401} \sqrt{2}$$

The function (or method) `n` gives a numerical approximation to this number.

```
_.n()
```

```
0.435501618497698
```

If this isn't yet enough precision, `n` can be given a second optional argument specifying how many digits precision you want:

```
(2^30/3^20*sqrt(2)).n(digits=100)
```

```
0.435501618497697542678003469998073498141904412675140022701396035106906854
```

Would you like to know the first few digits of π ?

```
print(pi.n(digits=5000))
```

```
3.1415926535897932384626433832795028841971693993751058209749445923
816406286208998628034825342117067982148086513282306647093844609550
223172535940812848111745028410270193852110555964462294895493038196
288109756659334461284756482337867831652712019091456485669234603486
454326648213393607260249141273724587006606315588174881520920962829
409171536436789259036001133053054882046652138414695194151160943305
703657595919530921861173819326117931051185480744623799627495673518
752724891227938183011949129833673362440656643086021394946395224737
```

070217986094370277053921717629317675238467481846766940513200056812
452635608277857713427577896091736371787214684409012249534301465495
371050792279689258923542019956112129021960864034418159813629774771
996051870721134999999837297804995105973173281609631859502445945534
083026425223082533446850352619311881710100031378387528865875332083
420617177669147303598253490428755468731159562863882353787593751957
185778053217122680661300192787661119590921642019893809525720106548
632788659361533818279682303019520353018529689957736225994138912497
775283479131515574857242454150695950829533116861727855889075098381
463746493931925506040092770167113900984882401285836160356370766010
101819429555961989467678374494482553797747268471040475346462080466
259069491293313677028989152104752162056966024058038150193511253382
003558764024749647326391419927260426992279678235478163600934172164
199245863150302861829745557067498385054945885869269956909272107975
302955321165344987202755960236480665499119881834797753566369807426
25278625518184175746728909777279380008164706001614524919217321721
723501414419735685481613611573525521334757418494684385233239073941
334547762416862518983569485562099219222184272550254256887671790494
165346680498862723279178608578438382796797668145410095388378636095
800642251252051173929848960841284886269456042419652850222106611863
744278622039194945047123713786960956364371917287467764657573962413
086583264599581339047802759009946576407895126946839835259570982582
205224894077267194782684826014769909026401363944374553050682034962
451749399651431429809190659250937221696461515709858387410597885959
297549893016175392846813826868386894277415599185592524595395943104
725246808459872736446958486538367362226260991246080512438843904512
136549762780797715691435997700129616089441694868555848406353422072
582848864815845602850601684273945226746767889525213852254995466672
239864565961163548862305774564980355936345681743241125150760694794
096596094025228879710893145669136867228748940560101503308617928680
087476091782493858900971490967598526136554978189312978482168299894
226588048575640142704775551323796414515237462343645428584447952658
821051141354735739523113427166102135969536231442952484937187110145
540359027993440374200731057853906219838744780847848968332144571386
519435064302184531910484810053706146806749192781911979399520614196
428754440643745123718192179998391015919561814675142691239748940907
649423196156794520809514655022523160388193014209376213785595663893
870830390697920773467221825625996615014215030680384477345492026054
665925201497442850732518666002132434088190710486331734649651453905
626856100550810665879699816357473638405257145910289706414011097120
804390397595156771577004203378699360072305587631763594218731251471
532928191826186125867321579198414848829164470609575270695722091756
167229109816909152801735067127485832228718352093539657251210835791
369882091444210067510334671103141267111369908658516398315019701651
168517143765761835155650884909989859982387345528331635507647918535
322618548963213293308985706420467525907091548141654985946163718027
819943099244889575712828905923233260972997120844335732654893823911
259746366730583604142813883032038249037589852437441702913276561809
734440307074692112019130203303801976211011004492932151608424448596
669838952286847831235526582131449576857262433441893039686426243410

```
322697802807318915441101044682325271620105265227211166039666557309
471105578537634668206531098965269186205647693125705863566201855810
293606598764861179104533488503461136576867532494416680396265797877
556084552965412665408530614344431858676975145661406800700237877659
440171274947042056223053899456131407112700040785473326993908145466
458807972708266830634328587856983052358089330657574067954571637752
202114955761581400250126228594130216471550979259230990796547376125
765675135751782966645477917450112996148903046399471329621073404375
957359614589019389713111790429782856475032031986915140287080859904
109412147221317947647772622414254854540332157185306142288137585043
332175182979866223717215916077166925474873898665494945011465406284
663937900397692656721463853067360965712091807638327166416274888800
692560290228472104031721186082041900042296617119637792133757511495
015660496318629472654736425230817703675159067350235072835405670403
743513622224771589150495309844489333096340878076932599397805419341
7377441842631298608099888687413260472
```

Variable Expressions

Sage also works with variable expressions. To tell it that x and y are variables, we have to use the syntax below.

```
var('x y')
```

$$(x, y)$$

```
(2*x+7) * (x^2+3) * (x+2)^5
```

$$(x + 2)^5(2x + 7)(x^2 + 3)$$

```
expand(_)
```

$$2x^8 + 27x^7 + 156x^6 + 521x^5 + 1170x^4 + 1944x^3 + 2384x^2 + 1872x + 672$$

That saved a bit of work, but *Sage* can do far more than this. At this point, *Sage* no longer remembers where this polynomial came from. It's just a random polynomial. But *Sage* can still factor it:

```
_.factor()
```

$$(x + 2)^5(2x + 7)(x^2 + 3)$$

Could you have factored this polynomial as fast as *Sage* did? Could you have factored it at all? (If you think about it a bit, your answers to these questions should be "no" and either "yes" or "maybe".)

Assignment and Simplification

Often we need to assign a name to the result of a computation. *Sage* does this using the syntax **variable = value**. In making an assignment, *Sage* does some obvious simplifications first.

```
e1 = (x+y)^3*(x+y)^2
```

Notice that an assignment statement doesn't print any return value. If we want the value of **e1**, we need to ask for it explicitly. We can do this either in a separate *Sage* command or by doing two *Sage* commands together, separated either by a newline or by a semicolon.

```
e1
```

$$(x + y)^5$$

```
e2 = (x+y)^4*(x+y)^2
```

```
e2
```

$$(x + y)^6$$

```
e3 = (x+y)^4*(x+y)^3; e3
```

$$(x + y)^7$$

Often we need to simplify mathematical expressions. This is a complicated task, since there are so many manipulations we could do to most expressions, and since deciding what form is simplest is sometimes a judgement call. *Sage* has a number of methods that can be used to try to simplify expressions. Sometimes one has to try several in order to get the right result. Here we assign a new value to **e2** and then try calling first **simplify()** and then **simplify_rational()**. There are also methods called **simplify_log()**, **simplify_trig()**, **simplify_exp()**, **simplify_radical()**, and **simplify_full()**.

```
e2 = (x^3-y^3)/(x^2+x-y-y^2)
```

```
e2
```

$$\frac{x^3 - y^3}{x^2 - y^2 + x - y}$$

```
e2.simplify()
```

$$\frac{x^3 - y^3}{x^2 - y^2 + x - y}$$

```
e2.simplify_rational()
```

$$\frac{x^2 + xy + y^2}{x + y + 1}$$

Can we understand where this simplification came from? Well, what if we take the numerator and denominator of **e2** and factor them separately:

```
e2.numerator()
```

$$x^3 - y^3$$

```
_.factor()
```

$$(x - y)(x^2 + xy + y^2)$$

```
e2.denominator().factor()
```

$$(x - y)(x + y + 1)$$

Ah, so the simplification just came from factoring both the top and bottom and then cancelling the common factor. We could get the same result by just factoring **e2**.

```
e2.factor()
```

$$\frac{x^2+xy+y^2}{x+y+1}$$

Solving Equations

Sage can also solve equations, even symbolic ones. Notice the syntax of the commands below. You have to specify first the equation, then the variables you want to solve for. Also, since `=` is already taken as the assignment operator, equations are written using `==`. This agrees with the notation used in the C programming language, or in Python, in which *Sage* is written.

```
solve(x^2 + 5*x + 2 == 0, x)
```

$$\left[x = -\frac{1}{2} \sqrt{17} - \frac{5}{2}, x = \frac{1}{2} \sqrt{17} - \frac{5}{2} \right]$$

Obviously the quadratic equation has 2 roots, just as you would expect. If we want to show the two solutions separately, we can write a little loop in *Sage* like this:

```
solns = solve(x^2 + 5*x + 2 == 0, x)
for i in solns:
    show(i)
```

$$x = -\frac{1}{2} \sqrt{17} - \frac{5}{2}$$

$$x = \frac{1}{2} \sqrt{17} - \frac{5}{2}$$

Want numerical solutions? There are a couple of options. You can get the exact solutions and then approximate them. Here we take the right hand side of each of the equations in `solns` and then approximate it.

```
for i in solns:
    i.rhs().n()
```

```
-4.56155281280883
-0.438447187191170
```

Alternatively, we can use `find_root()` to look for numerical solutions between two values of x . This works even for equations *Sage* cannot solve symbolically. This probably works best if we have already plotted the curve so we know where to look for roots, but here's an example.

```
find_root(x^2 + 5*x + 2 == 0, -1, 0)
```

```
-0.438447187191
```

```
find_root(x^2 + 5*x + 2 == 0, -5, -1)
```

-4.56155281281

`solve()` can also handle more than one equation at a time. Here's an example.

```
solve([x+y==5, x-y==2], x, y)
[[x = (7/2), y = (3/2)]]
```

What if we get more adventuresome and try equations with symbolic coefficients?

```
var('a b c d e f')
(a, b, c, d, e, f)
```

```
quadratic = a*x^2 + b*x + c == 0
quadratic
```

$$ax^2 + bx + c = 0$$

```
solve(quadratic, x)
```

$$\left[x = \frac{-b + \sqrt{-4ac + b^2}}{2a}, x = \frac{-b - \sqrt{-4ac + b^2}}{2a} \right]$$

```
for i in _:
    i.rhs().show()
```

$$\frac{-b + \sqrt{-4ac + b^2}}{2a}$$

$$\frac{-b - \sqrt{-4ac + b^2}}{2a}$$

OK, so *Sage* knows the quadratic formula. Big deal. So do I. But how about equations of higher degree?

```
cubic = a*x^3 + b*x^2 + c*x + d == 0
cubic
```

$$ax^3 + bx^2 + cx + d = 0$$

```
print(solve(cubic, x))
```

```
[
x == -1/2*(I*sqrt(3) + 1)*(1/18*sqrt(27*a^2*d^2 + 4*a*c^3 - b^2*c^2
- 2*(9*a*b*c - 2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d - 9*a*b*c +
2*b^3)/a^3)^(1/3) - 1/3*b/a + 1/18*(-I*sqrt(3) + 1)*(3*a*c -
b^2)/((1/18*sqrt(27*a^2*d^2 + 4*a*c^3 - b^2*c^2 - 2*(9*a*b*c -
2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d - 9*a*b*c +
2*b^3)/a^3)^(1/3)*a^2),
x == -1/2*(-I*sqrt(3) + 1)*(1/18*sqrt(27*a^2*d^2 + 4*a*c^3 - b^2*c^2
- 2*(9*a*b*c - 2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d - 9*a*b*c +
```

```

2*b^3)/a^3)^(1/3) - 1/3*b/a + 1/18*(I*sqrt(3) + 1)*(3*a*c -
b^2)/((1/18*sqrt(27*a^2*d^2 + 4*a*c^3 - b^2*c^2 - 2*(9*a*b*c -
2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d - 9*a*b*c +
2*b^3)/a^3)^(1/3)*a^2),
x == (1/18*sqrt(27*a^2*d^2 + 4*a*c^3 - b^2*c^2 - 2*(9*a*b*c -
2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d - 9*a*b*c + 2*b^3)/a^3)^(1/3)
- 1/3*b/a - 1/9*(3*a*c - b^2)/((1/18*sqrt(27*a^2*d^2 + 4*a*c^3 -
b^2*c^2 - 2*(9*a*b*c - 2*b^3)*d)*sqrt(3)/a^2 - 1/54*(27*a^2*d -
9*a*b*c + 2*b^3)/a^3)^(1/3)*a^2)
]

```

```

for i in solve(cubic,x):
    i.rhs().show()

```

$$-\frac{1}{2} \left(i \sqrt{3} + 1 \right) \left(\frac{\sqrt{27 a^2 d^2 + 4 a c^3 - b^2 c^2 - 2 (9 a b c - 2 b^3) d} \sqrt{3}}{18 a^2} + \frac{-27 a^2 d - 9 a b c}{54 a^3} \right)$$

$$-\frac{1}{2} \left(-i \sqrt{3} + 1 \right) \left(\frac{\sqrt{27 a^2 d^2 + 4 a c^3 - b^2 c^2 - 2 (9 a b c - 2 b^3) d} \sqrt{3}}{18 a^2} + \frac{-27 a^2 d - 9 a b c}{54 a^3} \right)$$

$$\left(\frac{\sqrt{27 a^2 d^2 + 4 a c^3 - b^2 c^2 - 2 (9 a b c - 2 b^3) d} \sqrt{3}}{18 a^2} + \frac{-27 a^2 d - 9 a b c + 2 b^3}{54 a^3} \right)$$

Wow! So there's a cubic formula like the quadratic formula, one giving 3 roots instead of 2 and involving the complex numbers. (The formulas actually continue past the edge of the page, and $I = \sqrt{-1}$ is the imaginary square root of -1 .) There's also a quartic formula that *Sage* knows, but the output for it fills several pages.

How about quintics?

```

quintic = a*x^5 + b*x^4 + c*x^3 + d*x^2 + e*x + f == 0
quintic

```

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f = 0$$

```

solve(quintic,x)

```

$$[0 = ax^5 + bx^4 + cx^3 + dx^2 + ex + f]$$

Sage just gives us back the original equation. Why? Because Évariste Galois proved shortly before his death at the age of 20 that there is no general formula like the quadratic formula (one using $+$, $-$, \cdot , $/$ and radicals) that solves polynomial equations of degree 5 and higher. (That is, there is no formula solving all such equations. For particular values of the coefficients, there are often simple solutions.)

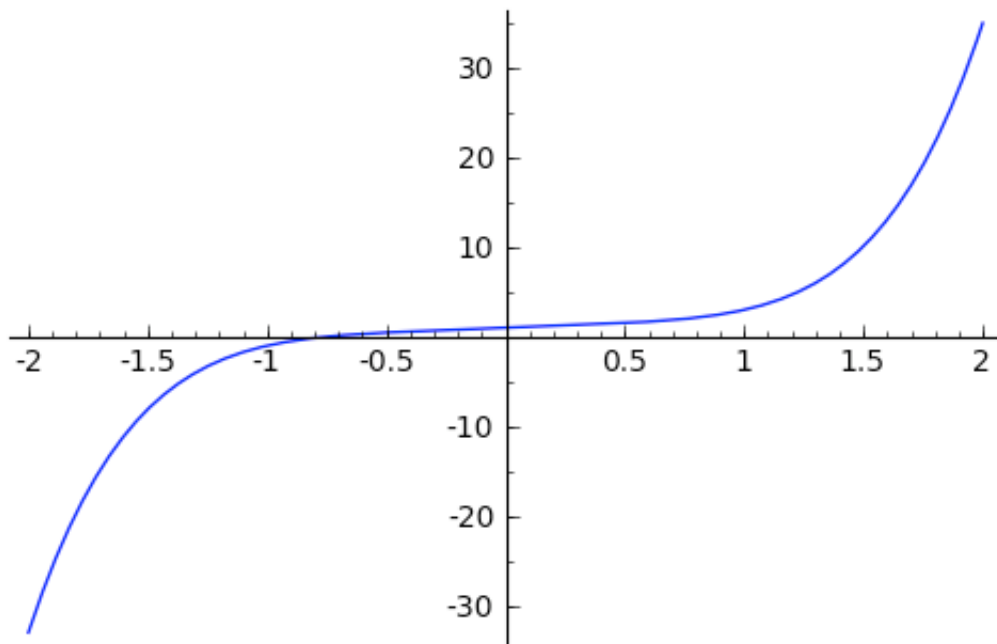
`find_root()` still works to find approximate solutions to equations of high degree:

```
find_root(x^5 + x + 1 == 0, -10, 10)
-0.754877666247
```

Plotting Graphs

Sage only found one solution to the last equation. How can we convince ourselves there is only one? One way to begin to build evidence might be to graph the function.

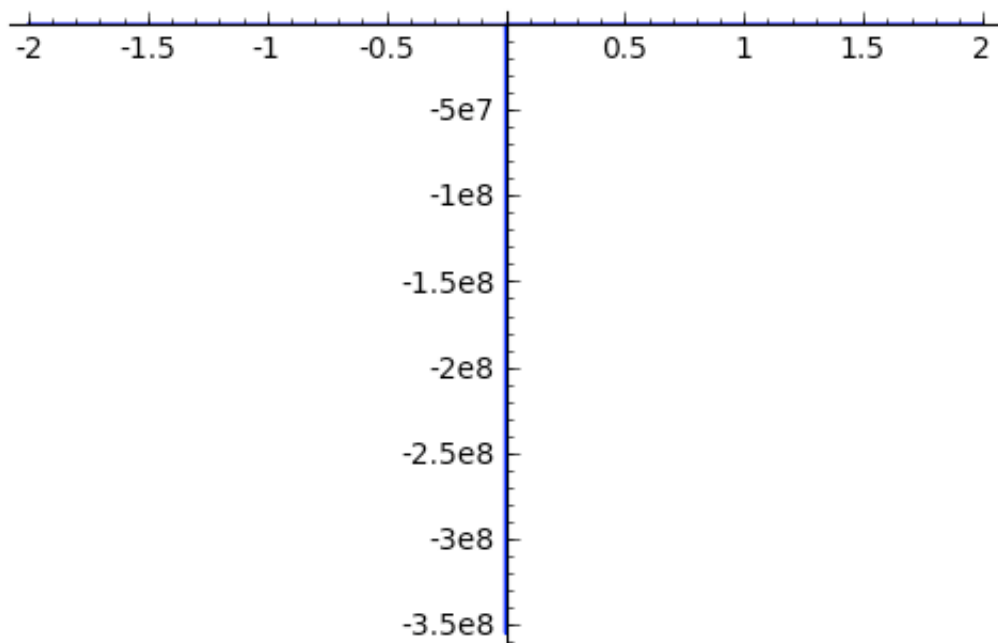
```
plot(x^5 + x + 1, (x, -2, 2))
```



This at least suggests that there is only one root between -2 and 2 .

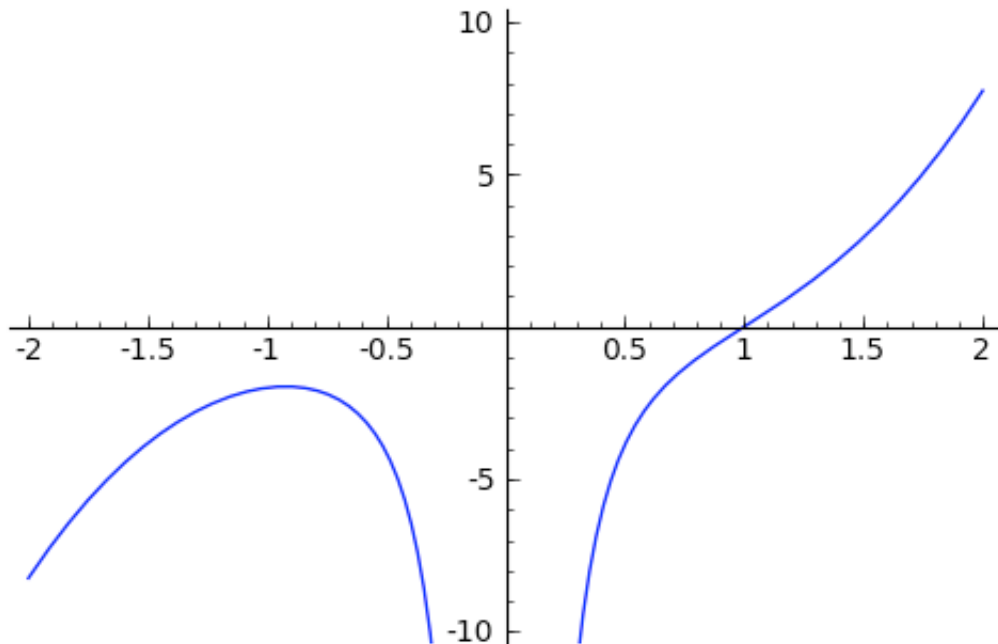
Sage sometimes makes an unhelpful choice of axes.

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



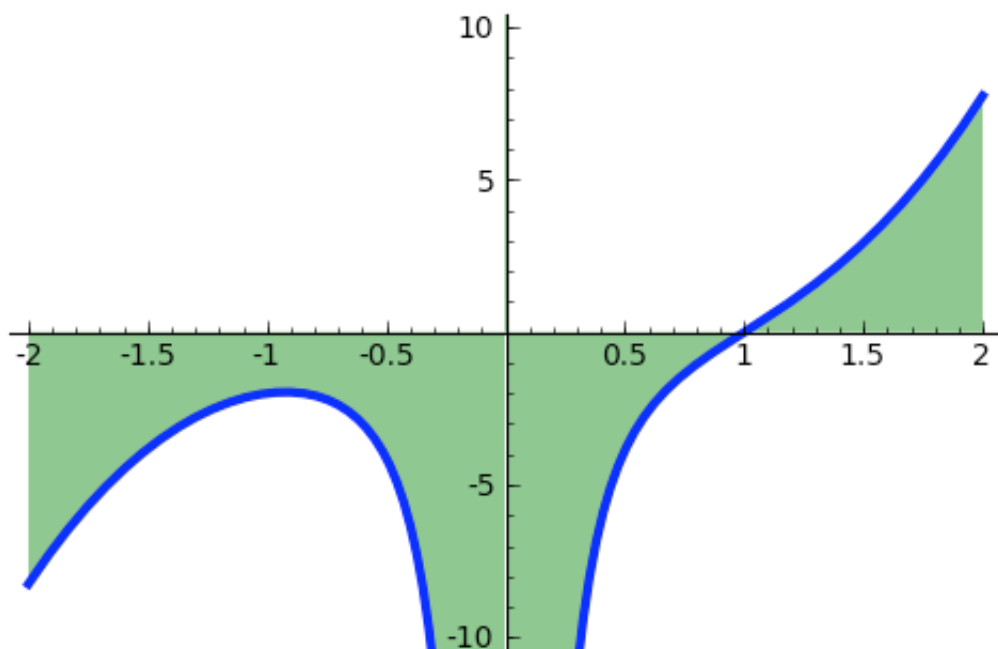
The problem here was the vertical asymptote at $x = 0$. *Sage* picked a scale that tried to show that asymptote, but ended up showing nothing - the whole graph ended up inside the axes at that scale. (For the record, the axis label $-8e6$ means -8×10^6 .) In this case, we can show the plot with a more limited range of y values.

```
plot(x^3 - 1/x^2, (x, -2, 2)).show(ymin = -10, ymax = 10)
```



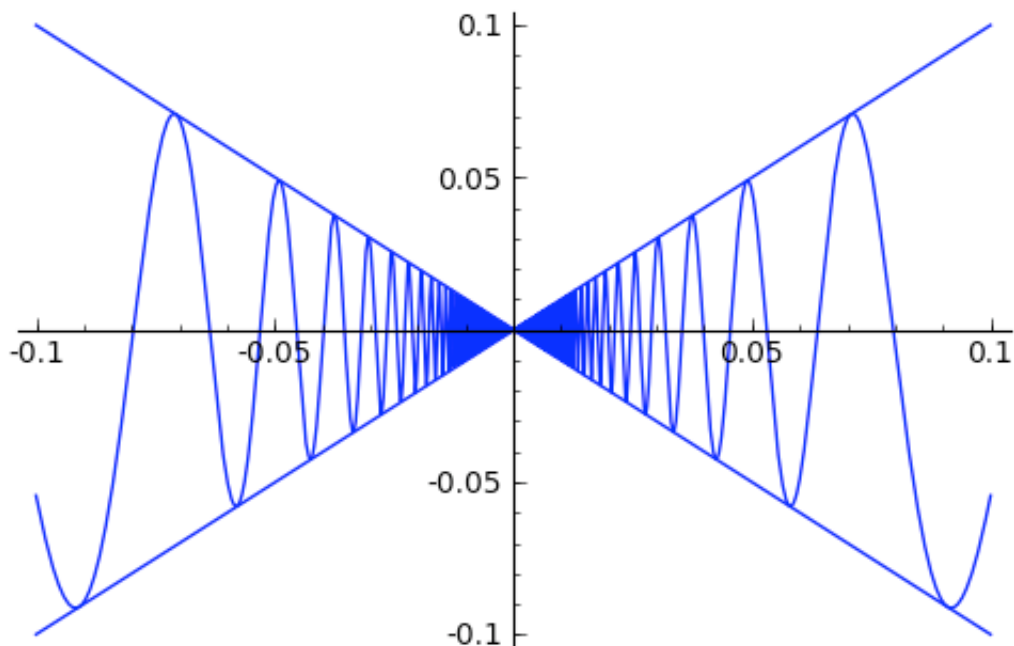
plot() and **show()** have an enormous number of options. Here are a few of them:

```
p = plot(x^3 - 1/x^2, (x, -2, 2), color='blue', thickness=3,
fill='axis', fillcolor='green')
show(p, ymin=-10, ymax=10)
```



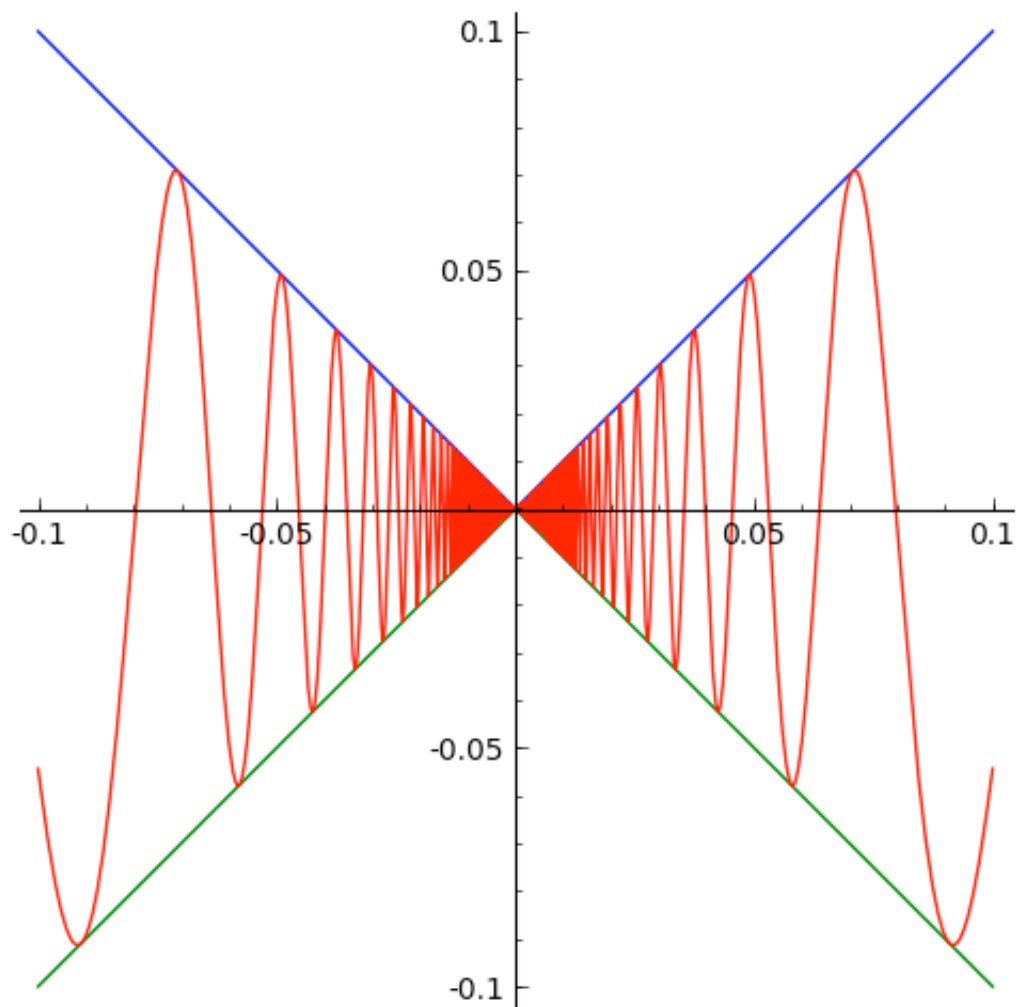
Sage can plot multiple functions together.

```
plot([abs(x), -abs(x), x*sin(1/x)], (x, -0.1, 0.1))
```



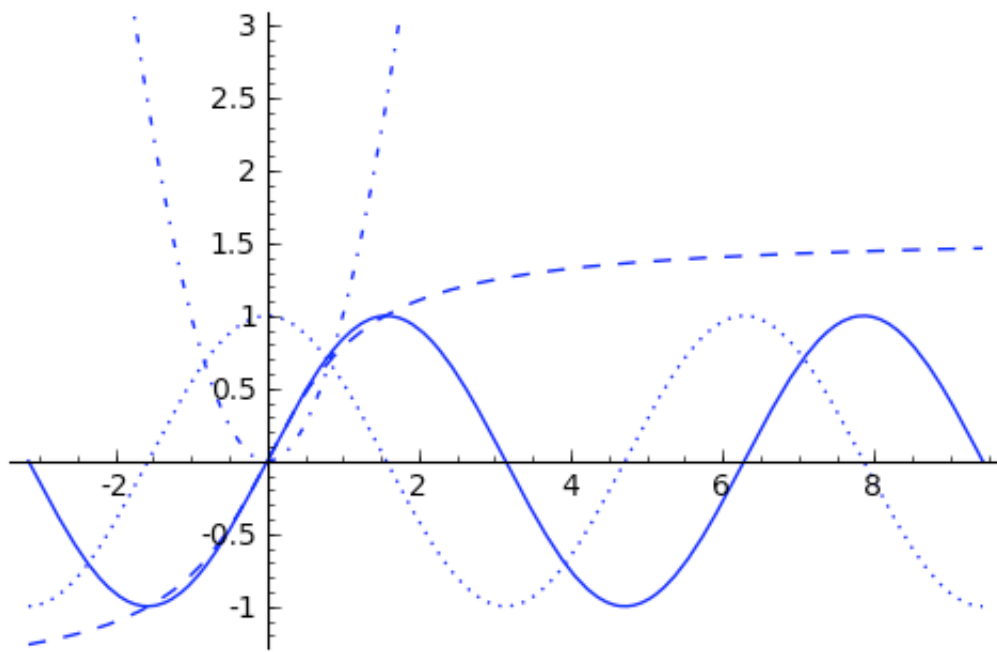
One difficulty with this is that you may not know which curve is which. In that case, you can give the curves distinct colors in their own **plot** statements, and then **show** all the plots together by joining them with + signs:

```
p1 = plot(abs(x), (x, -0.1, 0.1), color='blue')
p2 = plot(-abs(x), (x, -0.1, 0.1), color='green')
p3 = plot(x*sin(1/x), (x, -0.1, 0.1), color='red')
show(p1+p2+p3, aspect_ratio=1)
```



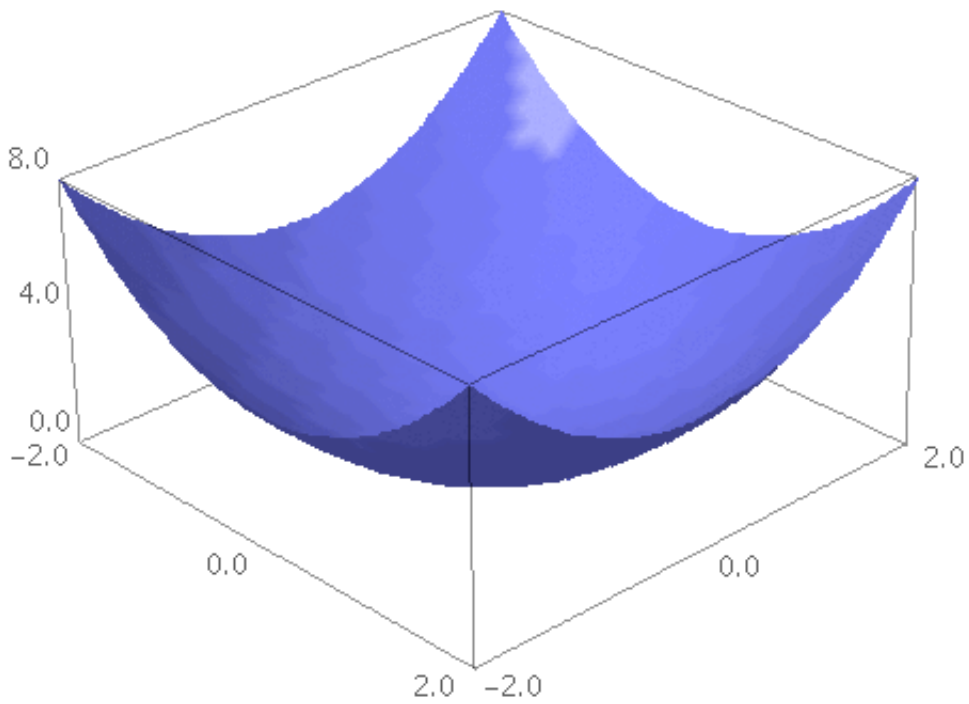
Color blind or printing the result on a monochrome printer? The **linestyle** option can be your friend.

```
p1 = plot(sin(x), (x, -pi, 3*pi), linestyle='-')
p2 = plot(cos(x), (x, -pi, 3*pi), linestyle=':')
p3 = plot(arctan(x), (x, -pi, 3*pi), linestyle='--')
p4 = plot(x^2, (x, -pi, 3*pi), linestyle='-.')
show(p1+p2+p3+p4, ymin=-1.2, ymax=3)
```



3D plots are also possible.

```
plot3d(x^2+y^2, (x,-2,2), (y,-2,2))
```



[Get Image](#)

Help!

There are numerous sources of help within *Sage*. If you know the name of the function on which you want help, you can enter its name followed by a question mark:

```
numerator?
```

```
File: /Applications/sage/local/lib/python2.6/site-packages/sage/misc/functional.py
```

```
Type: <type 'function'>
```

```
Definition: numerator(x)
```

```
Docstring:
```

```
Returns the numerator of x.
```

```
EXAMPLES:
```

```
sage: R.<x> = PolynomialRing(QQ)
sage: F = FractionField(R)
sage: r = (x+1)/(x-1)
sage: numerator(r)
x + 1
sage: numerator(17/11111)
17
```

If you're not certain of the name of a command, you can type the first few letters and then hit the <TAB> key to get a pop-up with possible completions. Try typing **sol**<TAB> to see all command names that start with the letters sol. (Here <TAB> denotes a single keystroke. Just hit the key labeled TAB.) Even more extensively, try **quintic**.<TAB> to see a large pop-up with every single message that could be sent to the object **quintic**.

For even more help, click the [Help](#) link at the top of any *Sage* Notebook.

Substitutions, Etc.

Here are a few more commands which are useful in calculus. We begin by defining an expression, making some substitutions, and using limits to compute its derivative directly from the definition:

```
f = x^2+x
f
```

```
 $x^2 + x$ 
```

```
f.substitute(x=3)
```

```
12
```

```
var('h')
f.substitute(x=x+h)
```

```
 $(h + x)^2 + h + x$ 
```

```
expand(_)
```

$$h^2 + 2hx + x^2 + h + x$$

```
_ - f
```

$$h^2 + 2hx + h$$

```
_/h
```

$$\frac{h^2 + 2hx + h}{h}$$

```
_.simplify_rational()
```

$$h + 2x + 1$$

```
limit(_, h=0)
```

$$2x + 1$$

Defining Functions

An irritating feature of this calculation was using **f.substitute(x=3)** to compute **f** when $x = 3$. We'd like to be able just to say **f(3)**, but we can't, because **f** is just an expression, not a function. So how would we define a function in *Sage*? There are several ways. For simple one-line functions, we can do this:

```
g(x) = x^2 + x
```

```
f
```

$$x^2 + x$$

```
g
```

$$x \mapsto x^2 + x$$

```
f(3)
```

```
__main__:3: DeprecationWarning: Substitution using function-call  
syntax and unnamed arguments is deprecated and will be removed fro  
a future release of Sage; you can use named arguments instead, lik  
EXPR(x=..., y=...)  
12
```

```
g(3)
```

```
12
```

More complicated functions can be written in Python:

```
def biggest(x, y):  
    if x >= y:  
        return(x)  
    else:  
        return(y)
```

```
biggest(pi, 3)
```

π

```
biggest(-3, -pi)
```

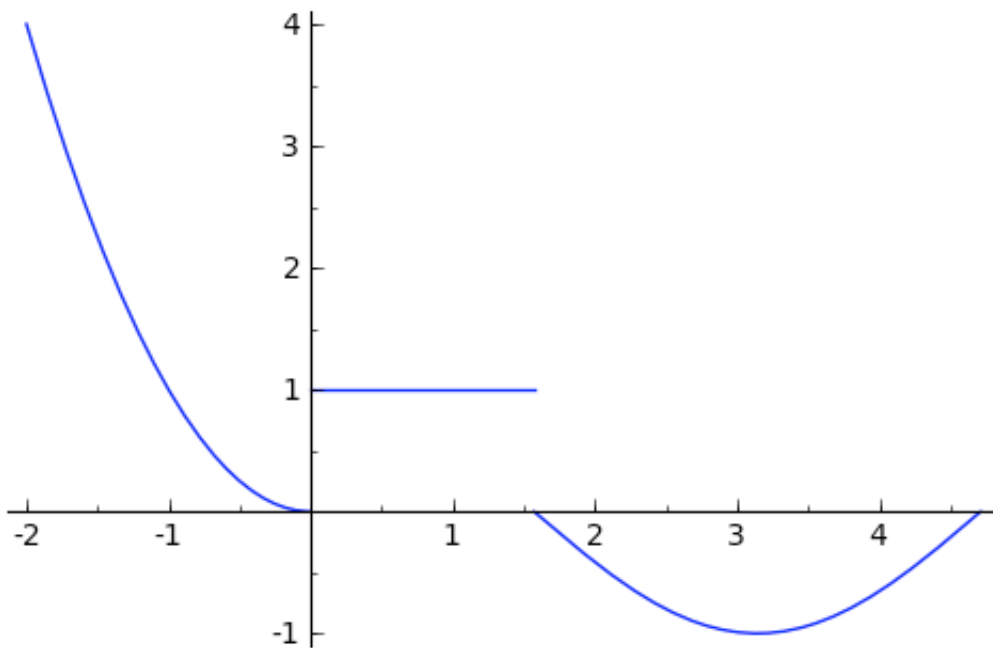
-3

Similarly, piecewise-defined functions can be handled in a couple of ways. To get a plotable expression that is not a function, you can use Sage's **Piecewise** function.

```
peas = Piecewise([[(-2, 0), x^2], [(0, pi/2), 1], [(pi/2, 3*pi/2),  
cos(x) ]])  
peas
```

$$\begin{cases} x^2 & \text{on } (-2, 0) \\ 1 & \text{on } (0, 1/2 * pi) \\ \cos(x) & \text{on } (1/2 * pi, 3/2 * pi) \end{cases}$$

```
plot(peas)
```



To get a proper function, you have to write a Python function using the different cases:

```
def beans(x):  
    if x <= 0:  
        return(x^2)  
    else:  
        if x < pi/2:  
            return(1)  
        else:  
            return(cos(x))
```

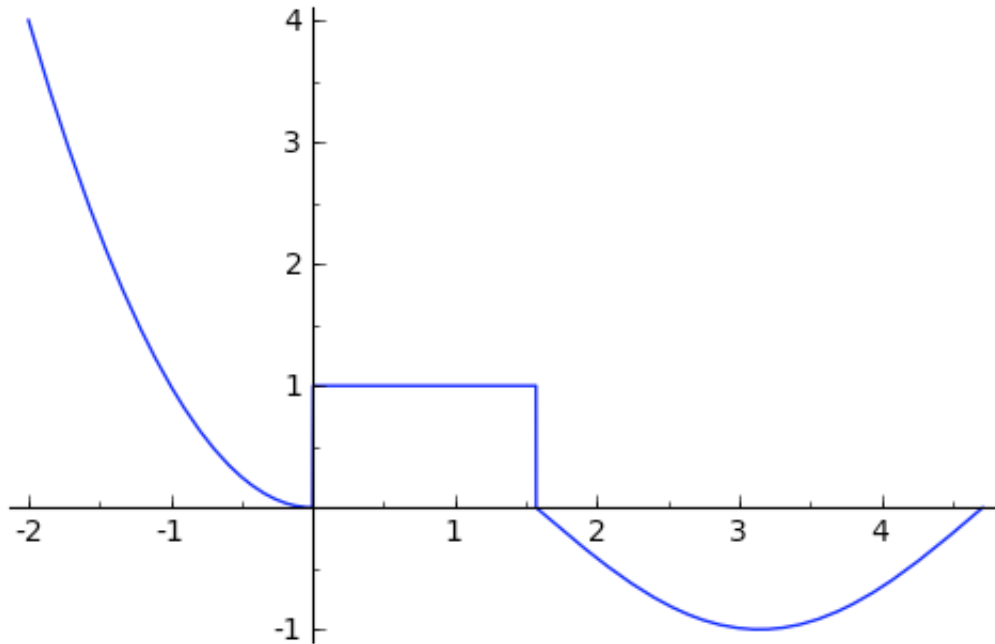
```
beans(0.5)
```

1

```
beans(3*pi/4)
```

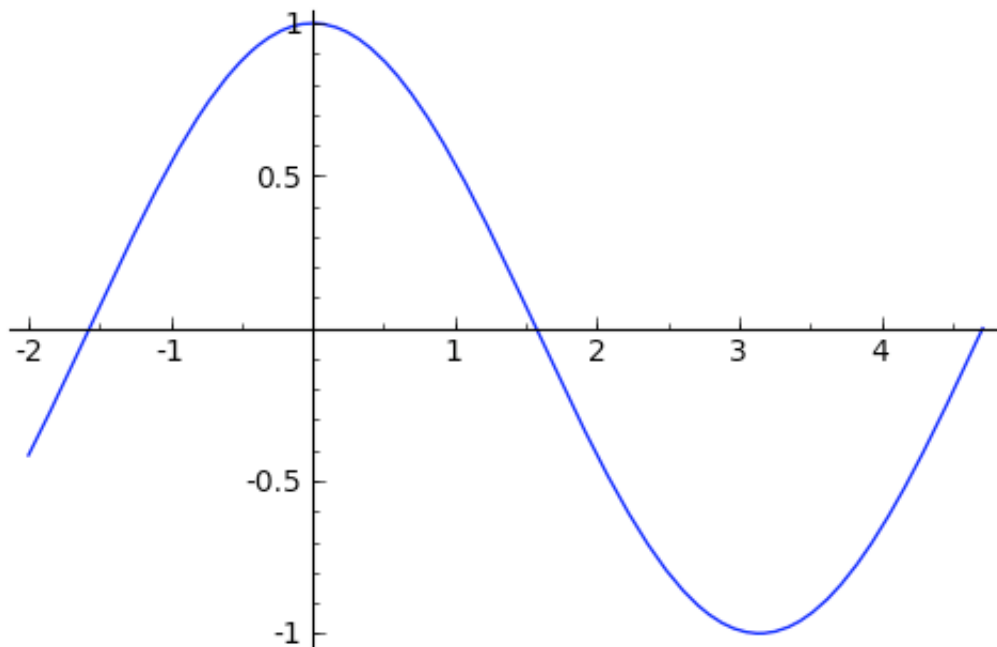
$-\frac{1}{2}\sqrt{2}$

```
plot(beans, (x,-2,3*pi/2))
```



There is one small technicality here. We need to plot **beans**, not **beans(x)**. Otherwise the function **beans(x)** gets evaluated at the start of the **plot** command, returning one of the three functions that make up **beans**, and then this function only gets plotted.

```
plot(beans(x), (x,-2,3*pi/2))
```



More Calculus Functions

Sage can directly compute derivatives (including second and third derivatives and so on), sums, integrals and limits. Here are some examples:

```
derivative(sin(x^2), x)
```

$$2x \cos(x^2)$$

Here are the first, second, third, and twentieth derivatives of the function

$$f(x) = x^2 + \frac{1}{x}.$$

```
f(x) = x^2 + 1/x
f(x)
```

$$x^2 + \frac{1}{x}$$

```
derivative(f(x), x)
```

$$2x + \frac{-1}{x^2}$$

```
derivative(f(x), x, 2)
```

$$\frac{2}{x^3} + 2$$

```
derivative(f(x), x, 3)
```

$$\frac{-6}{x^4}$$

```
derivative(f(x), x, 20)
```

$$\frac{2432902008176640000}{x^{21}}$$

`derivative()` can also be written as `diff()`.

```
diff(ln(x), x)
```

$$\frac{1}{x}$$

```
integral(x^2, x)
```

$$\frac{1}{3} x^3$$

```
integral(x^2, x, 1, 4)
```

$$21$$

```
limit((x^2-3*x+2)/(x-1), x=1)
```

$$-1$$

```
[1/n^2 for n in [1..10]]
```

$$\left[1, \frac{1}{4}, \frac{1}{9}, \frac{1}{16}, \frac{1}{25}, \frac{1}{36}, \frac{1}{49}, \frac{1}{64}, \frac{1}{81}, \frac{1}{100}\right]$$

```
sum([1/n^2 for n in [1..10]])
```

$$\frac{1968329}{1270080}$$

The current version of *Sage* allows a nicer syntax for `sum()`, which even supports infinite sums. Here's an example that is just too cool to leave out, showing that

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi}{6}.$$

```
var('n')
```

```
sum(1/n^2, n, 1, 10)
```

$$\frac{1968329}{1270080}$$

```
sum(1/n^2, n, 1, infinity)
```

$$\frac{1}{6} \pi^2$$

Formatting Text and Manipulating Cells

This document is a *Sage* Worksheet, so obviously *Sage* can be used to typeset text as well as to do mathematics.

Sage can either produce output that is pretty-printed in ASCII characters or else output that is properly typeset. Which you get is controlled by a checkbox at the top of each Worksheet, which can have **Typeset** either ticked on or off. The setting in this checkbox can be changed as you work, if there are some lines you want typeset and some lines you just want printed.

Now a few words about the *Sage* user interface. A new cell for computing mathematics is opened by clicking on the blue line that appears when one hovers directly above a cell. An execution cell is deleted by deleting all the text inside it and then hitting *delete* one more time with the cursor in the empty cell. Removing the cell removes the output for the command in that cell as well.

Shift-clicking on the blue line opens a new html cell for typing in text. The use of one of these cells should be obvious to anybody who has used a word processor of any sort. The only thing that's not obvious is how to input mathematics like

$$\int_0^{\infty} \frac{\sin x}{x} dx = \frac{\pi}{2}.$$

Math like this is written using LaTeX, a math markup language. Equations are written between \$ signs (for inline equations) or between pairs of \$\$ signs (for centered displayed equations). I won't try to give a full introduction to LaTeX, but the equation above was written

\$\$

`\int_0^{\infty} \frac{\sin x}{x} \, dx = \frac{\pi}{2}`.

\$\$

Similarly, if I want to write that the solution to the equation $ax^2 + bx + \theta = 0$ is

$$x = \frac{-b \pm \sqrt{b^2 - 4a\theta}}{2a},$$

I would write that the solution to the equation $ax^2+bx+\theta=0$ is

\$\$

`x=\frac{-b\pm\sqrt{b^2-4a\theta}}{2a}`.

\$\$

Hitting the *Save changes* button to close and display the html window results in the equations showing up in typeset form.