

A Quick Introduction to Sage

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, you can use Earlham's own *Sage* Notebook server at <https://sage.cs.earlham.edu:8000/>.

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.

So 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
223
140
```

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
1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858127594672917553146825187145285692314\
51051124936122493198378815695858127594672917553146825187145285692314\
04359845775746985748039345677748242309854210746050623711418779541821\
53046474983581941267398767559165543946077062914571196477686542167660\
429831652624386837205668069376
```

```
(355/113)**10
31789487802830871103515625
33945673899222314849
```

```
factorial(4)
24
```

```
4.factorial()
24
```

```
print(125.factorial())
18826771768889260997437677024916008575954036487149242588759823150835\
31563316135988668829328894959231336464054459300577406301619193413805\
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 * 5 * 7
315.factor()
20.factorial() - 12.factorial()
3^2 * 5 * 7
```

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(_)`

$2^{10} \cdot 3^5 \cdot 5^2 \cdot 7 \cdot 11^3 \cdot 41976119$

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)`

0.4355016184976975426780034699980734981419044126751400227013960351069068541814735297648401904204298624

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

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

3.141592653589793238462643383279502884197169399375105820974944592307\
81640628620899862803482534211706798214808651328230664709384460955058\
22317253594081284811174502841027019385211055596446229489549303819644\
28810975665933446128475648233786783165271201909145648566923460348610\
45432664821339360726024914127372458700660631558817488152092096282925\
40917153643678925903600113305305488204665213841469519415116094330572\
7036575951953092186117381932611793105118548074462379962749567351885\
75272489122793818301194912983367336244065664308602139494639522473719\
07021798609437027705392171762931767523846748184676694051320005681271\
45263560827785771342757789609173637178721468440901224953430146549585\
37105079227968925892354201995611212902196086403441815981362977477130\
99605187072113499999983729780499510597317328160963185950244594553469\
0830264252308253344685035261931188171010003137838752886587533208381\
42061717766914730359825349042875546873115956286388235378759375195778\
18577805321712268066130019278766111959092164201989380952572010654858\
63278865936153381827968230301952035301852968995773622599413891249721\
77528347913151557485724245415069595082953311686172785588907509838175\
46374649393192550604009277016711390098488240128583616035637076601047\
1018194295596198946767837449448255379774726847104047534646208046684\
25906949129331367702898915210475216205696602405803815019351125338243\
00355876402474964732639141992726042699227967823547816360093417216412\
19924586315030286182974555706749838505494588586926995690927210779509\
302953211653449872027559602364806654991198818347977535663980742654\
2527862551818417574672890977727938000816470600161452491921732172147\
72350141441973568548161361157352552133475741849468438523323907394143\
3345477624168625189835694855620992192218427255025425688767179049460\
1653466804988627232791786085784388279679766814541009538837863609506\
8006422512520511739298489608412848862694560424196528502210661186306\
74427862203919494504712371378696095636437191728746776465757396241389\
08658326459958133904780275900994657640789512694683983525957098258226\
2052248940772671947826848260147699002640136394437455305608203496252\
45174939965143142980919065925093722169646151570985838741059788595977\
29754989301617539284681382686838689427741559918559252459539594310499\
72524680845987273644695848653836736222626099124608051243884390451244\
136549762780777156914359977001296160894416948685584840635342207222\
58284886481584560285060168427394522674676788952521385225499546667278\
23986456596116354886230577456498035593634568174324112515076069479451\
09659609402522887971089314566913686722874894056010150330861792868092\
0874760917824938589097149096759852613655497818931297848216829989487\
2265880485756401427047755513237964145152374623464542858444795265867\
82105114135473573952311342716610213596953623144295248493718711014576\
54035902799344037420073105785390621983874478084784896833214457138687\
51943506430218453191048481005370614680674919278191197939952061419663\
42875444064374512371819217999839101591956181467514269123974894090718\
64942319615679452080951465502252316038819301420937621378559566389377\
87083039069792077346722182562599661501421503068038447734549202605414\
6659252014974285073251866600213243408819071048633173464965145390579\
62685610055081066587969981635747363840525714591028970641401109712062\
80439039759515677157700420337869936007230558763176359421873125147120\
53292819182618612586732157919841484882916447060957527069572209175671\
1672910981690915280173506712748583222871835209353965725121083579151\
36988209144421006751033467110314126711136990865851639831501970165151\
1685171437657618351565088490998959823873455283316350764791853589\
32261854896321329330898570642046752590709154814165498594616371802709\
81994309924488957571282890592323326097299712084433573265489382391193\
25974636673058360414281388303203824903758985243744170291327656180937\
73444030707469211201913020330380197621101100449293215160842444859637\
66983895228684783123552658213144957685726243344189303968642624341077\
322697802807318915441101044682325716201052652272111660396665730925\
47110557853763466820653109896526918620564769312570586356620185581007\
29360659876486117910453348850346113657686753249441668039626579787718\
5560845296541266540853061434443185867697514566140680070023787765913\
44017127494704205622305389945613140711270004078547332699390814546646

```

45880797270826683063432858785698305235808933065757406795457163775254\
20211495576158140025012622859413021647155097925923099079654737612551\
76567513575178296664547791745011299614890304639947132962107340437518\
95735961458901938971311179042978285647503203198691514028708085990480\
1094121472213179476477262241425485454033215718530614228813758504306\
33217518297986622371721591607716692547487389866549494501146540628433\
66393790039769265672146385306736096571209180763832716641627488880078\
69256029022847210403172118608204190004229661711963779213375751149595\
01566049631862947265473642523081770367515906735023507283540567040386\
7435136222477158915049530984448933309634087807693259939780541934144\
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(_)
```

```
2 x^8 + 27 x^7 + 156 x^6 + 521 x^5 + 1170 x^4 + 1944 x^3 + 2384 x^2 + 1872 x + 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 if **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 anew 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)
```

$$[x == -1/2 * \sqrt{17} - 5/2, x == 1/2 * \sqrt{17} - 5/2]$$

Obviously the quadratic equation has 2 roots, just as you would expect. *Sage* writes this result in a rather unfriendly way. If we want to show each of the solutions in a nicer form, we can give the solutions a name and then show each solution. This means writing a little loop in *Sage* like this:

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

```
solns
```

$$[x == -1/2 * \sqrt{17} - 5/2, x == 1/2 * \sqrt{17} - 5/2]$$

```
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)
```

$$[x == -1/2 * (b + \sqrt{-4 * a * c + b^2})/a, x == -1/2 * (b - \sqrt{-4 * a * c + b^2})/a]$$

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

$$-\frac{1}{2} \frac{(b + \sqrt{-4ac + b^2})}{a}$$

$$-\frac{1}{2} \frac{(b - \sqrt{-4ac + b^2})}{a}$$

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()

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

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

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

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)

```
Traceback (click to the left for traceback)
...
#0: to_poly_solve(e=a*x^5+b*x^4+c*x^3+d*x^2+e*x+f =
0,vars=x)(topoly_solver.mac line 15)
```

Sage produces an error. 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:

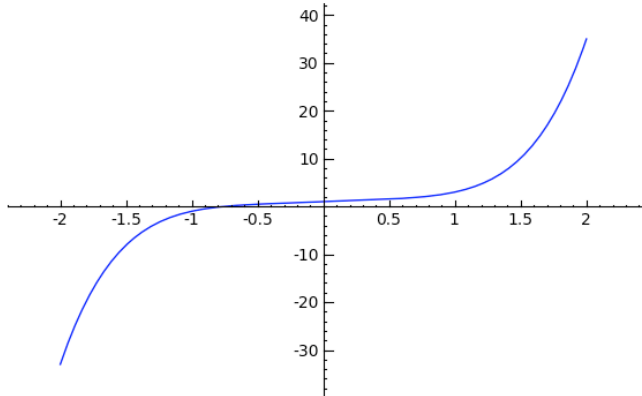
```
find_root(x^5 + x + 1 == 0, -10, 10)
```

```
-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.

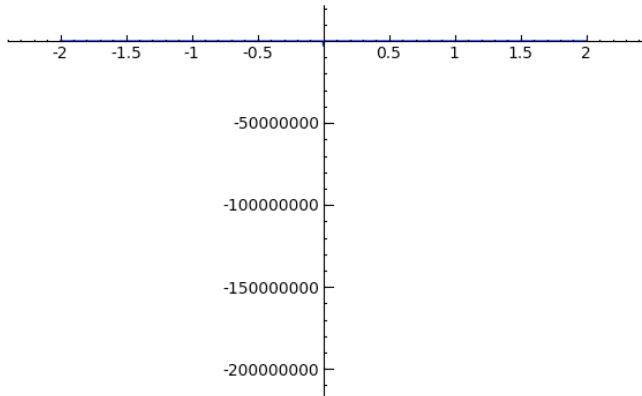
```
plot(x^5 + x + 1, (x, -2, 2))
```



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

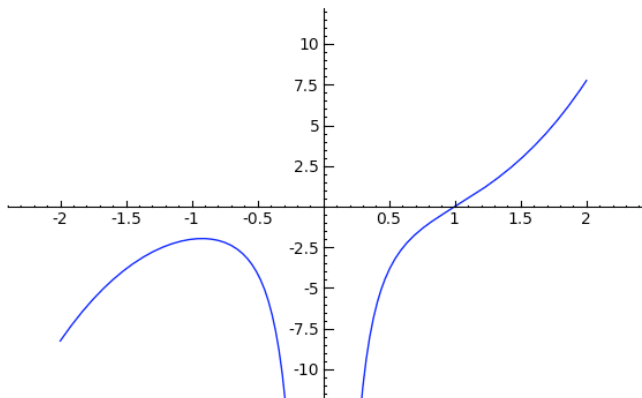
Sage sometimes makes an unhelpful choice of axes.

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



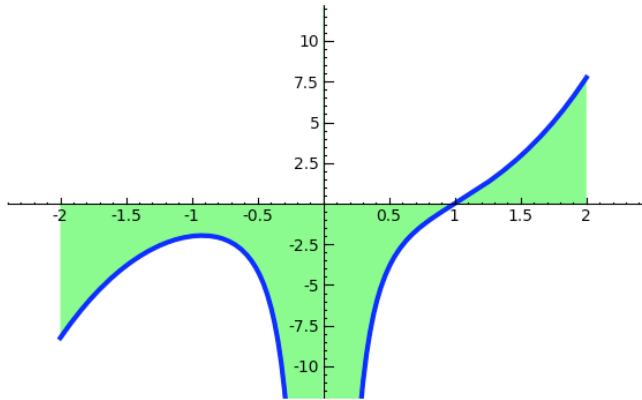
The problem here was the vertical asymptote at $x = 0$. *Sage* picked a scale that tried to show that asymptote, but ended up showing nothing - the whole graph ended up inside the axes at that scale. In this case, we can show the plot with a more limited range of y values.

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



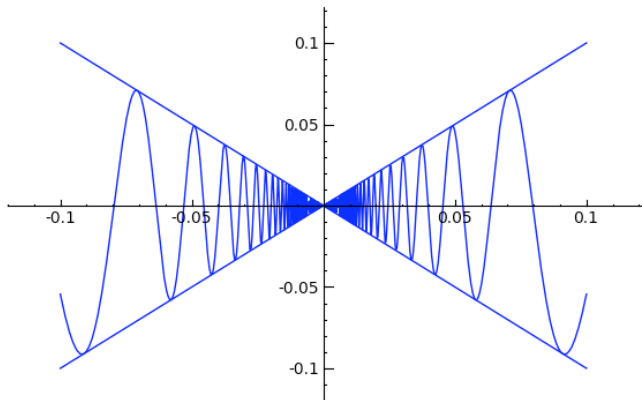
plot() and **show()** have an enormous number of options. Here are a few of them:

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



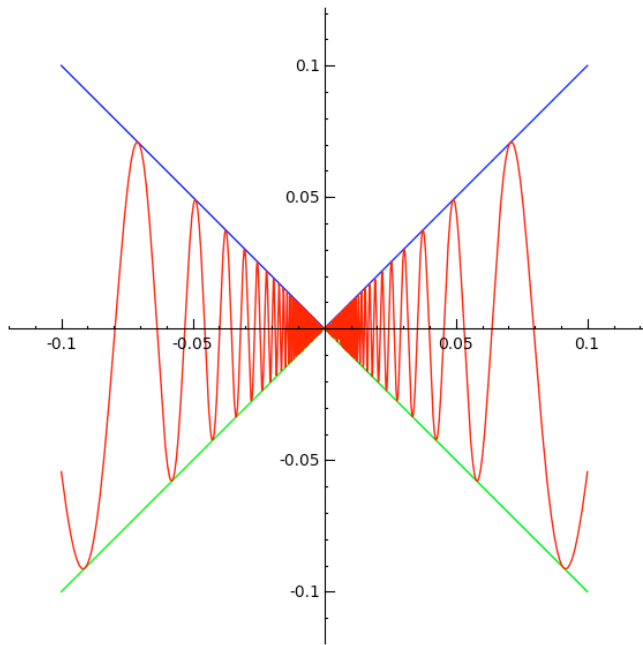
Sage can plot multiple functions together.

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



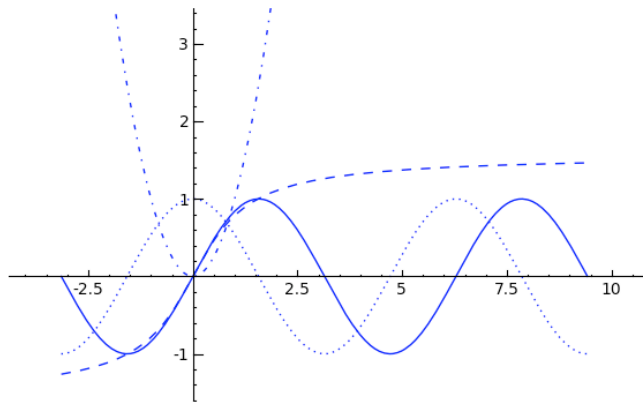
One difficulty with this is that you may not know which curve is which. In that case, you can give the curves distinct colors in their own **plot** statements, and then **show** all the plots together by joining them with + signs:

```
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))
```

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?

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
      (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 ↦ x^2 + x
f(3)
      __main__:7: DeprecationWarning: Substitution using function-call
      syntax and unnamed arguments is deprecated and will be removed from
      a future release of Sage; you can use named arguments instead, like
      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(pi,4)
      4

```

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.

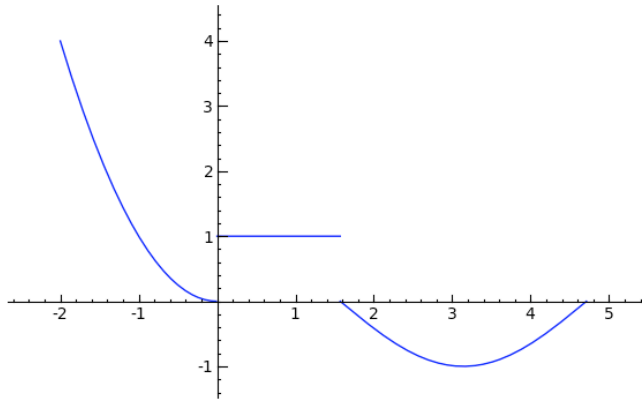
```

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))
```

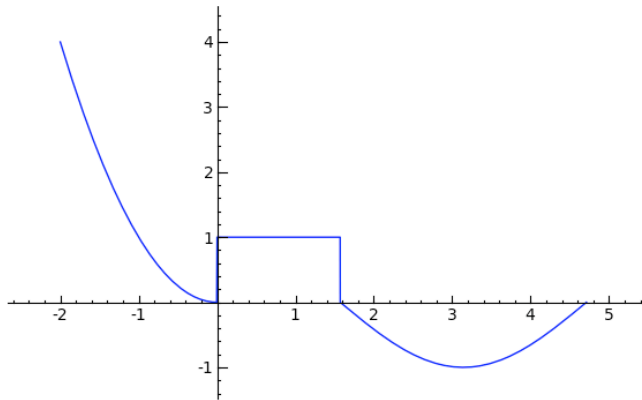
beans(0.5)

1

beans(3*pi/4)

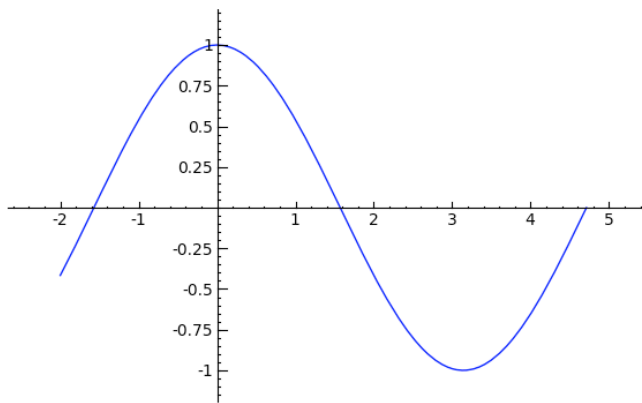
$-\frac{1}{2}\sqrt{2}$

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)

$$2\frac{1}{x^3} + 2$$

derivative(f(x),x,3)

$$-6\frac{1}{x^4}$$

derivative(f(x),x,20)

$$2432902008176640000 \frac{1}{x^{21}}$$

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)

$$-1$$

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

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

Sage doesn't yet support infinite sums, but some infinite sums can be computed by using *Sage* to call *Maxima*, another open source computer algebra system that ships with *Sage*. Here's an example that is just too cool to leave out, showing that

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

maxima('sum(1/n^2,n,1,inf), simpsum')

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

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 $Sax^2+bx+\theta=0$ 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.