

Examples of embedding Sage in L^AT_EX with SageT_EX

Dan Drake and others

March 25, 2010

1 Inline Sage, code blocks

This is an example $2+2=4$. If you raise the current year mod 100 (which equals 10) to the power of the current day (25), you get 1000000000000000000000000. Also, 2010 modulo 42 is 36.

Code block which uses a variable `s` to store the solutions:

```
1+1
var('a,b,c')
eqn = [a+b*c==1, b-a*c==0, a+b==5]
s = solve(eqn, a,b,c)
```

Solutions of $eqn = [bc + a = 1, -ac + b = 0, a + b = 5]$:

$$\left[a = \frac{((25I)\sqrt{79} + 25)}{((6I)\sqrt{79} - 34)}, b = \frac{((5I)\sqrt{79} + 5)}{(I\sqrt{79} + 11)}, c = \left(\frac{1}{10}I\right)\sqrt{79} + \frac{1}{10} \right]$$
$$\left[a = \frac{((25I)\sqrt{79} - 25)}{((6I)\sqrt{79} + 34)}, b = \frac{((5I)\sqrt{79} - 5)}{(I\sqrt{79} - 11)}, c = \left(-\frac{1}{10}I\right)\sqrt{79} + \frac{1}{10} \right]$$

Now we evaluate the following block:

```
E = EllipticCurve("37a")
```

You can't do assignment inside `\sage` macros, since Sage doesn't know how to typeset the output of such a thing. So you have to use a code block. The elliptic curve E given by $y^2 + y = x^3 - x$ has discriminant 37.

You can do anything in a code block that you can do in Sage and/or Python. Here we save an elliptic curve into a file.

```
try:
    E = load('E2')
except IOError:
    E = EllipticCurve([1,2,3,4,5])
    E.anlist(100000)
    E.save('E2')
```

The 9999th Fourier coefficient of $y^2 + xy + 3y = x^3 + 2x^2 + 4x + 5$ is -27 .

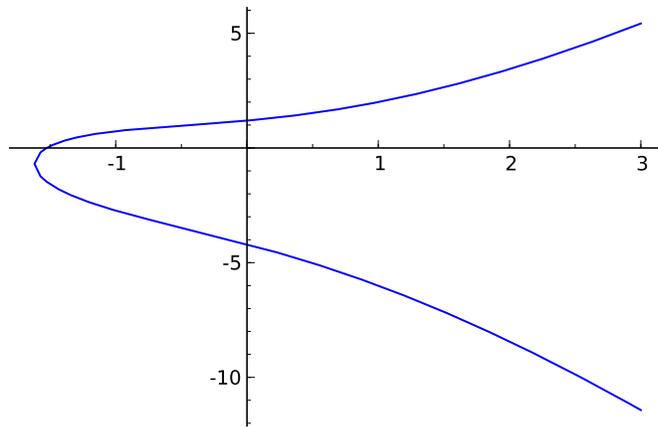
The following code block doesn't appear in the typeset file... but we can refer to whatever we did in that code block: $e = 7$.

```
var('x')
f(x) = log(sin(x)/x)
```

The Taylor Series of f begins: $x \mapsto -\frac{1}{467775}x^{10} - \frac{1}{37800}x^8 - \frac{1}{2835}x^6 - \frac{1}{180}x^4 - \frac{1}{6}x^2$.

2 Plotting

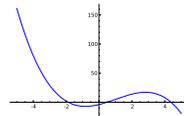
Here's a plot of the elliptic curve E .



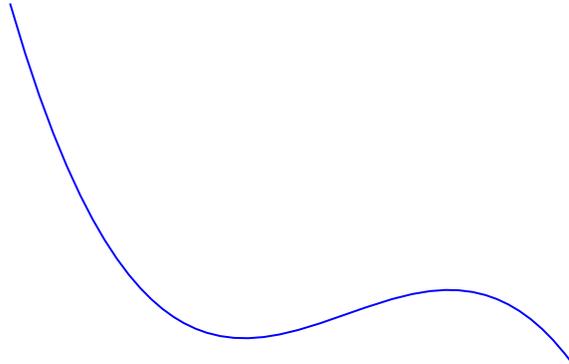
You can use variables to hold plot objects and do stuff with them.

```
p = plot(f, x, -5, 5)
```

Here's a small plot of f from -5 to 5 , which I've centered:



On second thought, use the default size of $3/4$ the `\textwidth` and don't use axes:

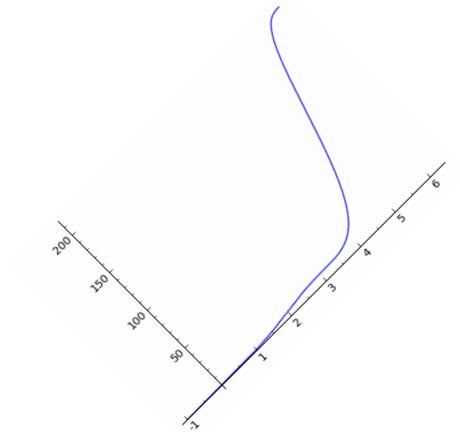


Remember, you're using Sage, and can therefore call upon any of the software packages Sage is built out of.

```
f = maxima('sin(x)^2*exp(x)')
g = f.integrate('x')
```

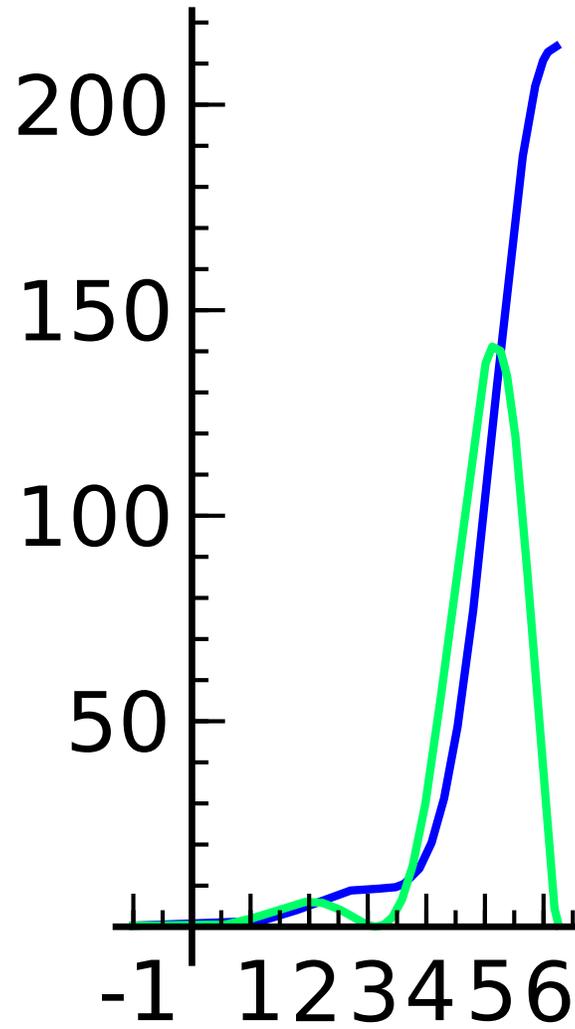
Plot $g(x)$, but don't typeset it.

You can specify a file format and options for `includegraphics`. The default is for EPS and PDF files, which are the best choice in almost all situations. (Although see the section on 3D plotting.)



If you use regular `latex` to make a DVI file, you'll see a box, because DVI files can't include PNG files. If you use `pdflatex` that will work. See the documentation for details.

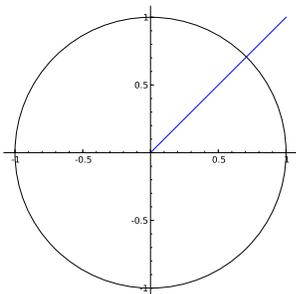
When using `\sageplot`, you can pass in just about anything that Sage can call `.save()` on to produce a graphics file:



To fiddle with aspect ratio, first save the plot object:

```
p = plot(x, 0, 1) + circle((0,0), 1)
p.set_aspect_ratio(1)
```

Now plot it and see the circular circle and nice 45 degree angle:



Indentation and so on works fine.

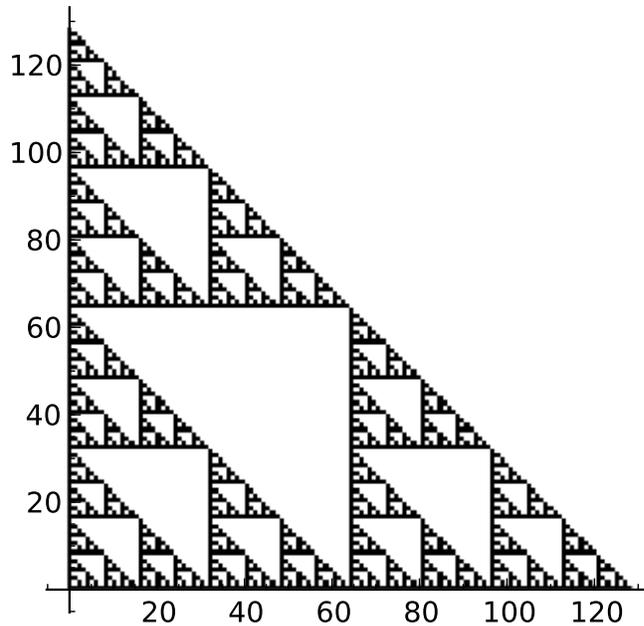
```

s      = 7
s2     = 2^s
P.<x>  = GF(2) []
M      = matrix(parent(x),s2)
for i in range(s2):
    p   = (1+x)^i
    pc  = p.coeffs()
    a   = pc.count(1)
    for j in range(a):
        idx      = pc.index(1)
        M[i,idx+j] = pc.pop(idx)

matrixprogram = matrix_plot(M,cmap='Greys')

```

And here's the picture:



Reset `x` in Sage so that it's not a generator for the polynomial ring: `x`

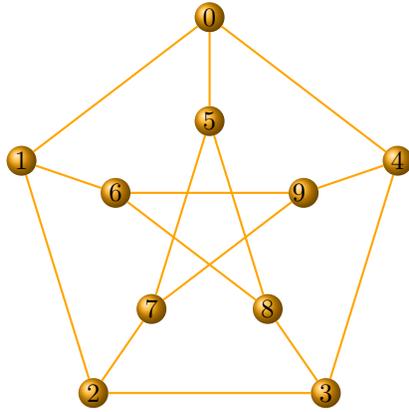
2.1 Plotting (combinatorial) graphs with TikZ

Sage now includes some nice support for plotting graphs using TikZ. Here, we mean things with vertices and edges, not graphs of a function of one or two variables.

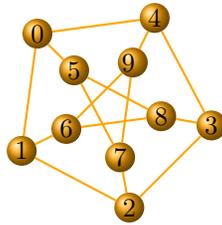
First define our graph:

```
g = graphs.PetersenGraph()
g.set_latex_options(tkz_style='Art')
```

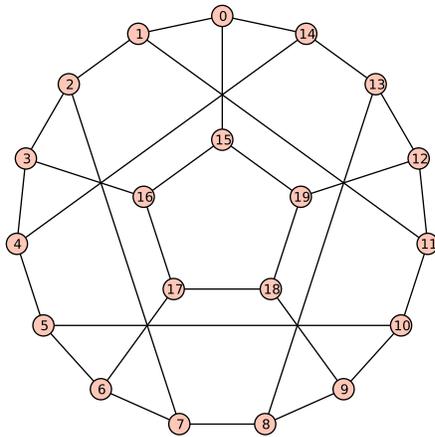
Now just do `\sage{}` on it to plot it. You'll need to use the `tkz-berge` package for this to work; that package in turn depends on `tkz-graph` and TikZ. See "L^AT_EX Options for Graphs" in the Sage reference manual for more details.



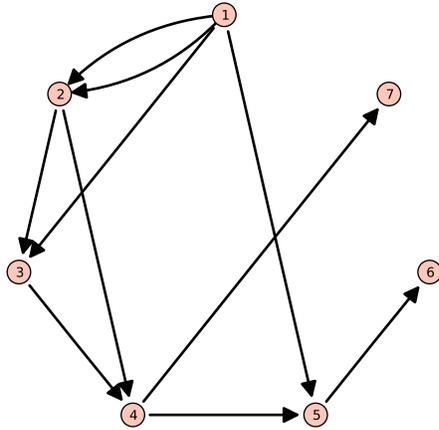
The above command just outputs a `tikzpicture` environment, and you can control that environment using anything supported by TikZ—although the output of `\sage{g}` explicitly hard-codes a lot of things and cannot be flexibly controlled in its current form.



Here's some more graphs, plotted using the usual plot routines.



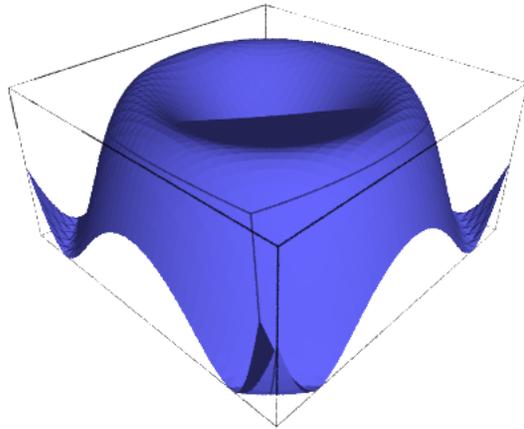
```
G4 = DiGraph({1:[2,2,3,5], 2:[3,4], 3:[4], 4:[5,7], 5:[6]},\
multiedges=True)
G4plot = G4.plot(layout='circular')
```



2.2 3D plotting

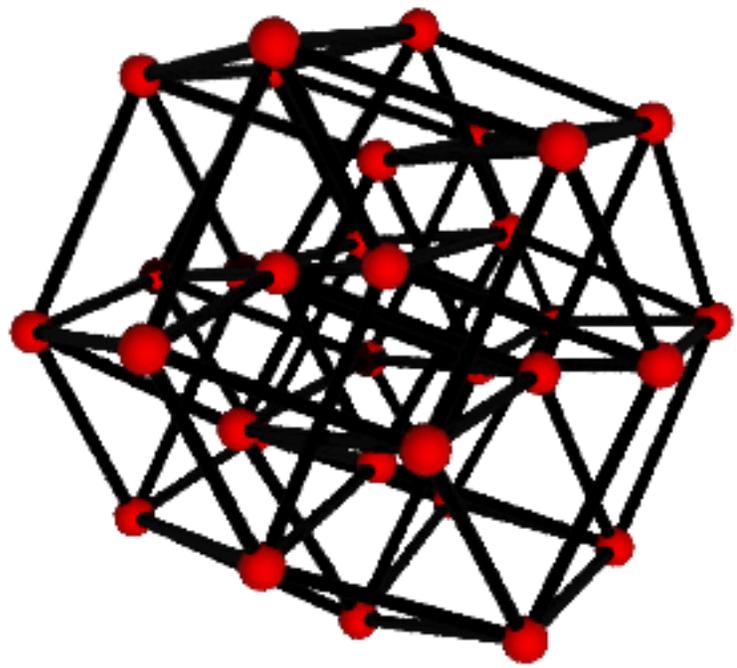
3D plotting right now (Sage version 4.3.4) is problematic because there's no convenient way to produce vector graphics. We can make PNGs, though, so if you pass `sageplot` a graphics object that cannot be saved to EPS or PDF format, we will automatically save to a PNG file, which can be used when typesetting a PDF file, but not when creating a DVI file. However, you can specify the “`imagemagick`” option, which will use the Imagemagick `convert` utility to make EPS files. See the documentation for details.

Here's a 3D plot whose format we do not specify; it will automatically get saved as a PNG file and won't work when using `latex` to make a DVI file.



Here's the (perhaps-not-so-) famous Sage cube graph in 3D.

```
G = graphs.CubeGraph(5)
```



3 Pausing SageTeX

Sometimes you want to “pause” for a bit while writing your document if you have embedded a long calculation or just want to concentrate on the L^AT_EX and ignore any Sage stuff. You can use the `\sagetexpause` and `\sagetexunpause` macros to do that.

A calculation: (SageTeX is paused) and a code environment that simulates a time-consuming calculation. While paused, this will get skipped over.

```
import time
time.sleep(15)
```

Graphics are also skipped: SageTeX is paused; no graphic

4 Make Sage write your L^AT_EX for you

With SageTeX, you can not only have Sage do your math for you, it can write parts of your L^AT_EX document for you! For example, I hate writing `tabular` environments; there’s too many fiddly little bits of punctuation and whatnot... and what if you want to add a column? It’s a pain—or rather, it *was* a pain. Just write a Sage/Python function that outputs a string of L^AT_EX code, and use `\sagestr`. Here’s how to make Pascal’s triangle.

```
def pascals_triangle(n):
    # start of the table
    s = [r"\begin{tabular}{cc|" + "r" * (n+1) + "|}"]
    s.append(r" & & $k$: & \\\")
    # second row, with k values:
    s.append(r" & ")
    for k in [0..n]:
        s.append("& {0} ".format(k))
    s.append(r"\\")
    # the n = 0 row:
    s.append(r"\hline" + "\n" + r"$n$: & 0 & 1 & \\\")
    # now the rest of the rows
    for r in [1..n]:
        s.append(" & {0} ".format(r))
        for k in [0..r]:
            s.append("& {0} ".format(binomial(r, k)))
        s.append(r"\\")
    # add the last line and return
    s.append(r"\end{tabular}")
```

```

return ''.join(s)

# how big should the table be?
n = 8

```

Okay, now here's the table. To change the size, edit `n` above. If you have several tables, you can use this to get them all the same size, while changing only one thing.

		<i>k:</i>								
		0	1	2	3	4	5	6	7	8
<i>n:</i>	0	1								
	1	1	1							
	2	1	2	1						
	3	1	3	3	1					
	4	1	4	6	4	1				
	5	1	5	10	10	5	1			
	6	1	6	15	20	15	6	1		
	7	1	7	21	35	35	21	7	1	
	8	1	8	28	56	70	56	28	8	1

5 Include doctest-like examples in your document

Here are some examples of using the `sageexample` environment:

```
sage: 1+1
```

2

```
sage: factor(x^2 + 2*x + 1)
```

$(x + 1)^2$

If you want to see the plain-text output as well as the typeset output, renew the `sageexampleincludetextoutput` command to `True`:

```
\renewcommand{\sageexampleincludetextoutput}{True}
```

This can be useful to check that the two outputs are consistent.

When this environment is near the bottom of the page, it may look like the page number is the output of a command, when in fact the real output is on the next page. If the output of a command below looks like 12, don't worry, that's just the page number.

```
sage: 1+1
```

2

2

```
sage: factor(x^2 + 2*x + 1)
(x + 1)^2
```

$$(x + 1)^2$$

Multiline statements are supported, as are triple-quoted strings delimited by single quotes:

```
sage: def f(a):
...     '''This function is really quite nice,
...     although perhaps not very useful.'''
...     print "f called with a = ", a
...     y = integrate(SR(cyclotomic_polynomial(10)) + a, x)
...     return y + 1

sage: f(x)
```

$$\frac{1}{5}x^5 - \frac{1}{4}x^4 + \frac{1}{3}x^3 + x + 1$$

Note that the “*f* called with...” stuff doesn’t get typeset, since when running Sage on `example.sage`, that gets printed to the terminal.

When typesetting your document, the validity of the outputs is not checked. In fact, the provided outputs are completely ignored:

```
sage: is_prime(57)
toothpaste
```

False

However, typesetting your document produces a file named `example_doctest.sage` containing all the doctest-like examples, and you can have Sage check them for you with:

```
$ sage -t example_doctest.sage
```

You should get one doctest failure from the “toothpaste” line above.

Please look into this file for the original line numbers.

Beware that `sage -t` does not handle well file names with special characters in them, particularly dashes, dots, and spaces—this ultimately comes from the way Python interprets `import` statements. Also, running doctests on files outside the main Sage library does not always work, so contact `sage-support` if you run into troubles.

Some more examples. This environment is implemented a little bit differently than the other environments, so it’s good to make sure that definitions are preserved across multiple uses. This will correctly define *a*, but not print its output because the statement is made up of a sequence of expressions.

```
sage: 1; 2; a=4; 3; a
```

After that, Sage should remember that $a = 4$ and be able to use that in future `sageexample` blocks:

```
sage: f(a)
```

$$\frac{1}{5}x^5 - \frac{1}{4}x^4 + \frac{1}{3}x^3 - \frac{1}{2}x^2 + 5x + 1$$