Examples of embedding Sage in LaTeX with SageTeX

Dan Drake and others

February 20, 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 (20), you get 100000000000000000000. 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 $= [bc + a = 1, -ac + b = 0, a + b = 5]$:

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

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

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.

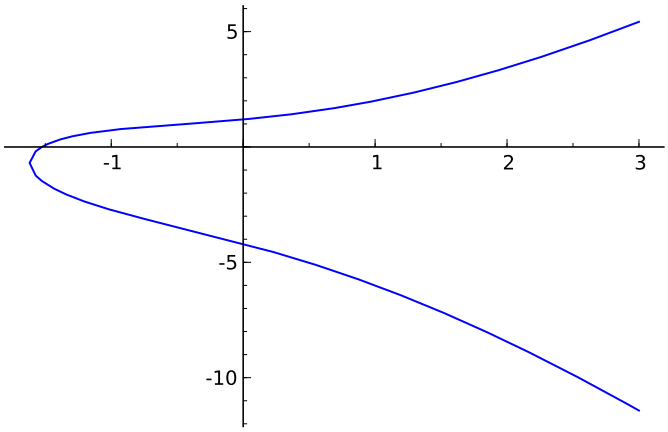
The following code block doesn't appear in the typeset file... but we can refer to whatever we did in that code block: $e = 7$.

```
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

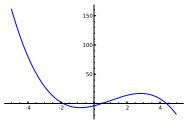
Here's a plot of the elliptic curve E.



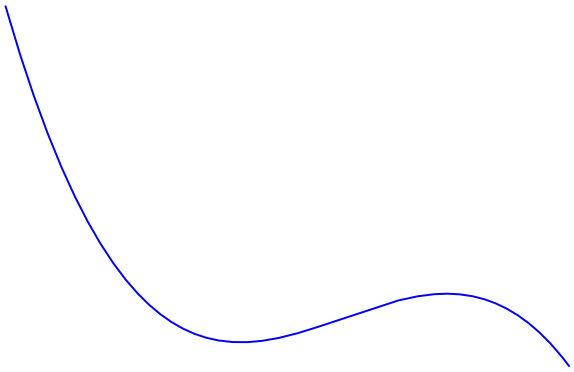
You can use variables to hold plot objects and do stuff with them.

```
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:

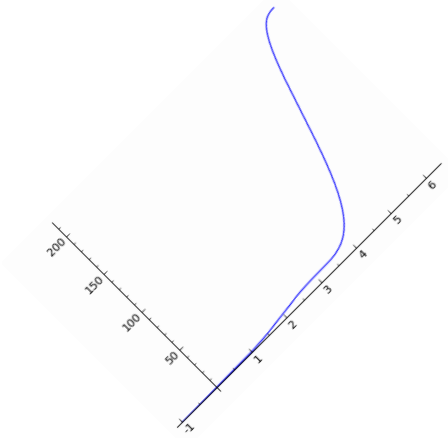


Remember, you're using Sage, and can therefore call upon any of the software packages Sage is built out of.

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

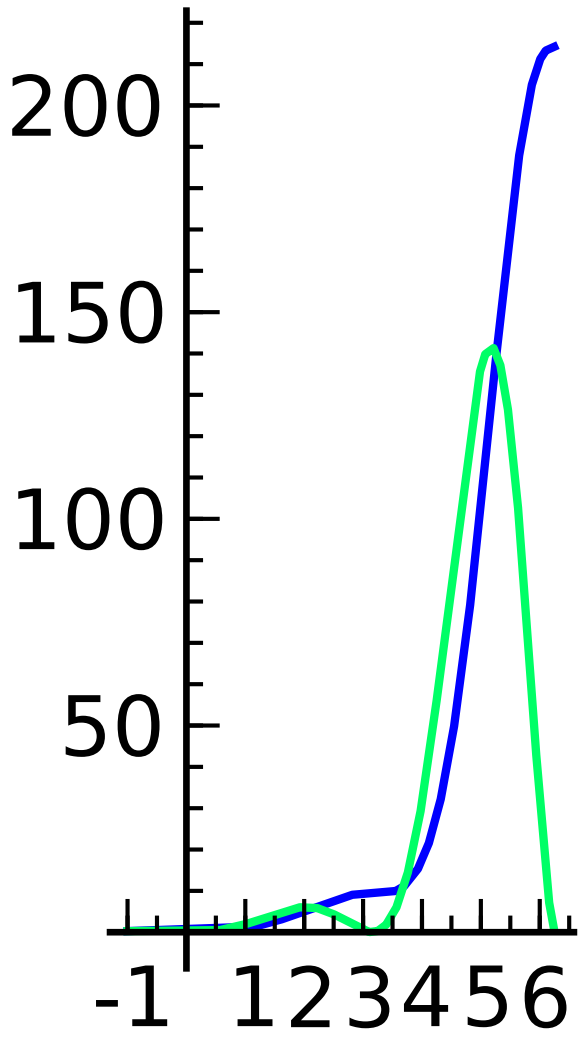
Plot $g(x)$, but don't typeset it.

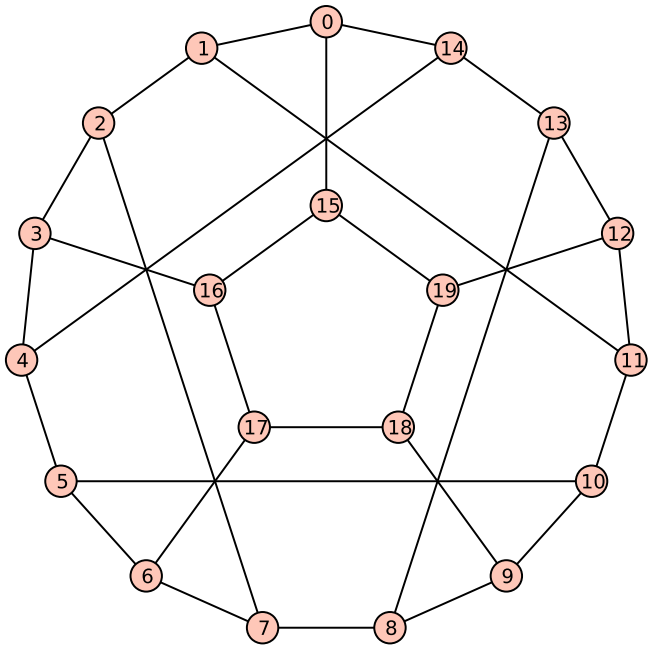
You can specify a file format and options for `includegraphics`. The default is for EPS and PDF files, which are the best choice in almost all situations. (Although see the section on 3D plotting.)



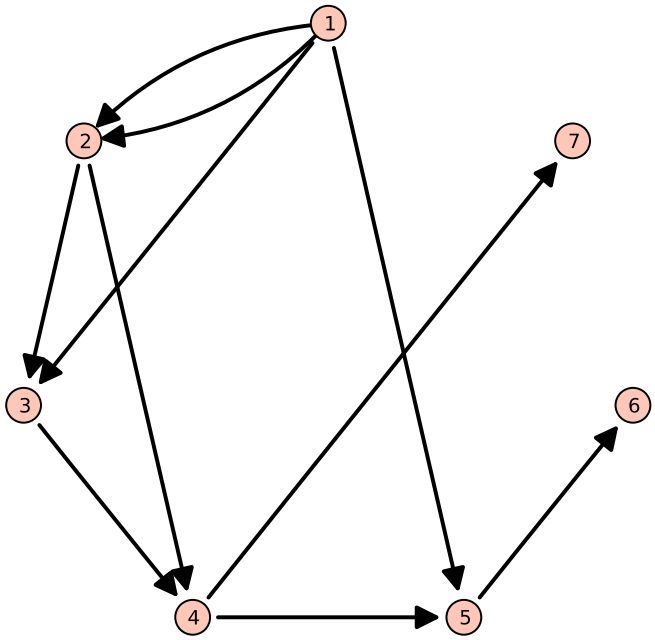
If you use regular `latex` to make a DVI file, you'll see a box, because DVI files can't include PNG files. If you use `pdflatex` that will work. See the documentation for details.

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


200
150
100
50
-1 1 2 3 4 5 6



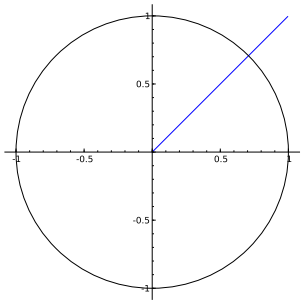

```
G4 = DiGraph({1:[2,2,3,5], 2:[3,4], 3:[4], 4:[5,7], 5:[6]},\
             multiedges=True)
G4plot = G4.plot(layout='circular')
```



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:

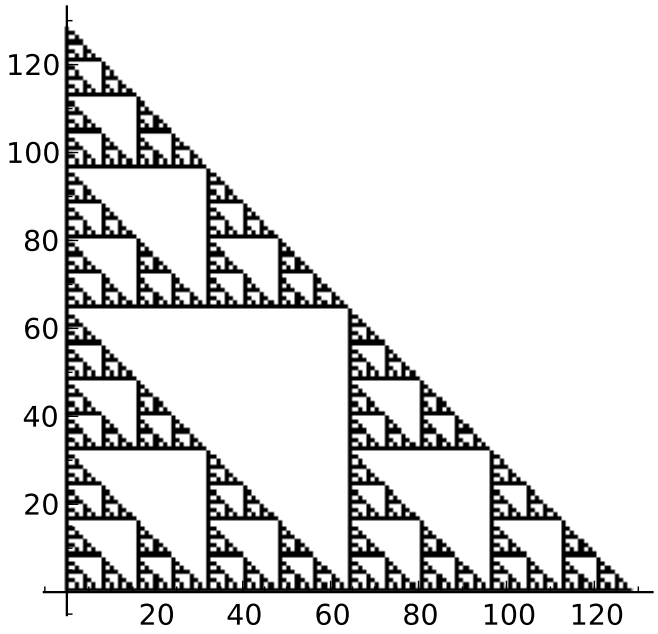


Indentation and so on works fine.

```
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

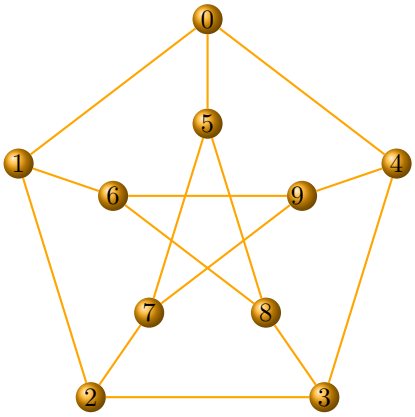
2.1 Plotting (combinatorial) graphs with TikZ

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.

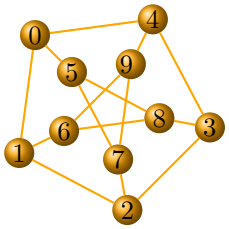
First define our graph:

```
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 "LaTeX 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.

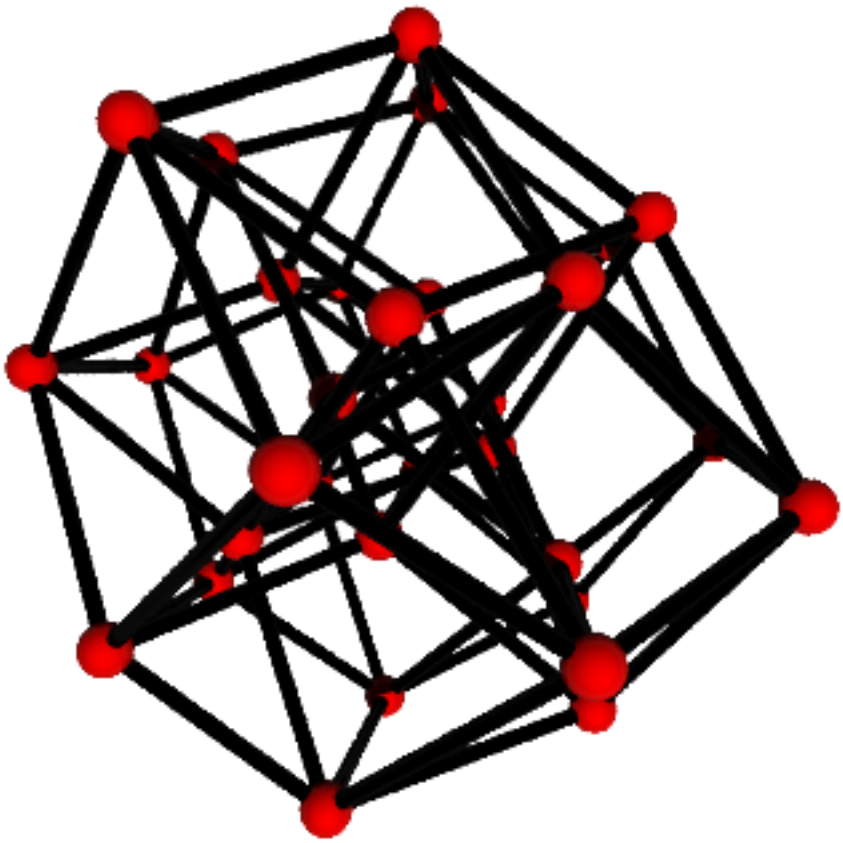


2.2 3D plotting

3D plotting right now is problematic because there's no convenient way to produce vector graphics. We can make PNGs, though, and since the `sageplot` command defaults to EPS and PDF, *you must specify a valid format for 3D plotting.* Sage right now (version 4.2.1) can't produce EPS or PDF files from `plot3d` objects, so if you don't specify a valid format, things will go badly. You can specify the "`imagemagick`" option, which will use the Imagemagick `convert` utility to make EPS files. See the documentation for details.

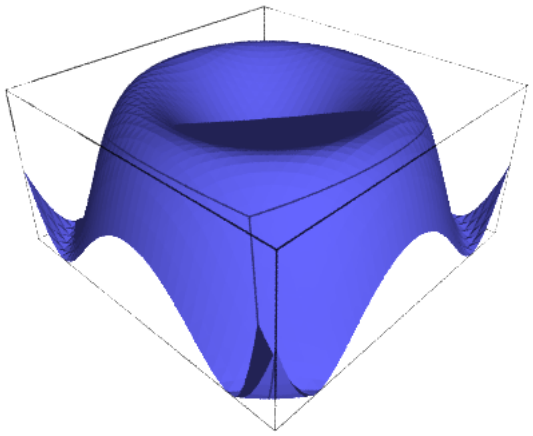
Here's the famous Sage cube graph in 3D.

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



And here's a regular sort of 3D plot. Since `plot3d` objects don't properly support the kind of `.save()` method that we need, so we have to work around it a bit and do things manually. Note that we can't use `\jobname` below. The `sage.misc.viewer.BROWSER` bit tells Sage to not pop up a viewer program; otherwise, when you run the `.sage` script, it will try to start a viewer program on the resulting image, which we don't want.

```
sage.misc.viewer.BROWSER=''
x, y = var('x y')
g = plot3d(sin(pi*(x^2+y^2))/2,(x,-1,1),(y,-1,1))
g.show(filename='sage-plots-for-example.tex/my-3d-plot', viewer='tachyon')
```



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 LATEX 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 LATEX for you

With SageTEX, you can not only have Sage do your math for you, it can write parts of your LATEX 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. Here's how to make Pascal's triangle. It requires the `amsmath` package because of what Sage does when producing a LaTeX representation of a string. (It puts it inside a `\text` macro.)

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

		k:								
		0	1	2	3	4	5	6	7	8
n:	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