Examples of embedding Sage in LaTeX with SageTeX

Dan Drake and others

October 27, 2009

1 Inline Sage, code blocks

This is an example $2 + 2 = $ **??**. If you raise the current year mod 100 (which equals **??**) to the power of the current day (27), you get **??**. Also, 2009 modulo 42 is **??**.

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 = **??**:

$$??$$

$$??$$

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 **??** has discriminant **??**.

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')
```
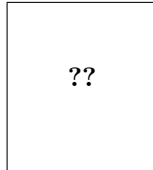
The 9999th Fourier coefficient of **??** is **??**.

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

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

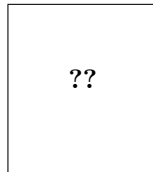The Taylor Series of f begins: **??**.

2 Plotting

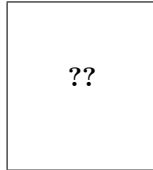Here's a plot of the elliptic curve E.

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:
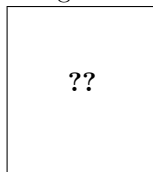

Remember, you're using Sage, and can therefore call upon any of the software packages Sage is built out of.

```
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:

```
??
```

```
??
```

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

```
??
```

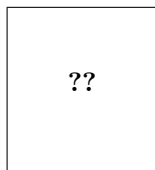Indentation and so on works fine.

```
        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:

<div style="border:1px solid black; width:150px; height:150px; text-align:center; line-height:150px;">

??

</div>

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

2.1   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 3.4.2) 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:

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

<div style="border:1px solid black; width:150px; height:150px; text-align:center; line-height:150px;">

??

</div>
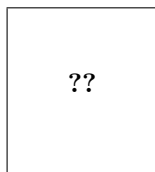
3   Pausing SageTEX

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
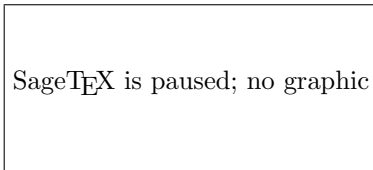
4 Make Sage write your LaTeX for you

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.

??