SageMath: A (Very) Short Introduction

Mathematical software is part of the standard toolkit of mathematicians. Throughout this course we will use the program called *SageMath* (it used to be know as *Sage*, but the "Math" was added to distinguish it from other "Sages") to do computations. The goal of this handout is to provide some basic orientation on the program, including where to get it. SageMath is free; it can be used online or installed on your own machine; it is very powerful, and it takes some time to learn. *SageMath* incorporates the functionality of many other free mathematics programs, including Maxima, Octave, R, GAP, and GP. *SageMath* runs either in a browser window or a terminal.

Of course, there are other mathematical software tools. Both *Maple* and *Mathematica* are well-known and quite powerful. The people who make *Mathematica* are also behind Wolfram Alpha, which can do lots of mathematics as well. These can all do some of the things we need to do, but I have decided to focus on *SageMath* because it is free and I want to learn more about how to use it.

General Information

SageMath is an ambitious attempt to create powerful mathematical software that is free and open-source. The ultimate ambition, says the Sage home page, is to create "a viable free open source alternative to *Magma*, *Maple*, *Mathematica* and MATLAB." (I'd say that they are very close to that, and in some aspects well beyond.) The development approach emphasizes openness: while William Stein is the leader of the team, contributions have come from across the mathematical community.

SageMath can be used through a web interface, without needing to download and install the program. There are two ways to do that: either the SageMath Cell or the more elaborate interface based on projects and notebooks offered by *CoCalc*. It is also possible to download and install the program on your own computer. When you do, you can run *SageMath* in a terminal window or you can run it in your browser. The latter is much like using *CoCalc*, but the program is running on your local machine.

Of the two web interfaces, SageMath Cell is particularly easy to use for small computations. It presents the user with a big blank rectangle where one can type in Sage commands. Below is a button labeled "Evaluate," which does exactly that. The output appears below. The downside is that SageMath will not remember what you did, so if you define a symbol and then press the "Evaluate" button, you cannot use it again without repeating the definition.

There are two very nice features of SageMath Cell that deserve note. First, it works on a tablet or phone. Second, because SageMath incorporates other open-source mathematical software, SageMath Cell can be put into GP or R mode to do computations in GP or R. I use this feature a lot.

CoCalc requires creating an account. Once you log in to your account, you can create projects, and each project can contain many notebooks. Notebooks allow you to enter lines of Sage code, which are evaluated when you hit Shift-Enter. Definitions and results are remembered within each session.

If you are going to use the program a lot, then of course the right thing to do is to download and install it. It takes quite a bit of space, but having it on your own machine avoids connectivity issues. Installing on a PC is easier than on a Mac.

The central web site for *SageMath* is `www.sagemath.com`. You can go there to download the program, find documentation, and so on.

Examples

Time to show some examples. You can find many more in A Tour of Sage, which is available online. SageMath is built using the Python programming language, so its syntax is similar to Python's.

Everything is done by entering a command and then asking SageMath to evaluate it. In SageMath Cell, you can enter a sequence of commands as well. If you enter

```
3 + 5
```

into the SageMath Cell and hit "Evaluate" you will get

```
8
```

in the results box. SageMath Cell typically only prints your last result. If you want to see two results, it's best to ask it explicitly:

```
print(3+5)
print(57.1^100)
```

Then you get two lines:

```
8
4.60904368661396e175
```

You can get some space between the two lines of output by adding `print(" ")` between the two commands.

In the terminal window, SageMath gives you a prompt and you type in your commands and then hit enter. The same example looks like this:

```
sage: 5+3
8
sage: 57.1^100
4.60904368661396e175
```

In a notebook, the process is similar to what happens in a terminal, except that you hit shift-enter to evaluate a notebook cell. Notebooks are best if you are going to do graphical things (such as a 3D plot).

SageMath likes you declare that a letter is a variable before using it that way. The only variable it knows by default is x. So, to work with functions of two variables x and y you will need to begin with

```
sage: var('x y')
(x, y)
```

Here is one way to define a function (everything in SageMath can be done in lots of different ways).

```
sage: f(x)=x^4
sage: f
x |--> x^4
sage: f(x)
x^4
```

The command `f(x)=x^4` produces no output, but *SageMath* has been told what you mean by `f`. Notice also that *SageMath* distinguishes the function `f` (the rule that sends x to x^4) from `f(x)` (the value of f when you plug in x).

Two ways to compute a derivative:

```
sage: diff(f,x)
x |--> 4*x^3
sage: diff(f(x),x)
4*x^3
```

(On SageMath Cell you need define f each time, remember.) The first one is the derivative *function* f', the second is $f'(x)$. Higher derivatives are easy too. Here's the second derivative:

```
sage: diff(f,x,2)
x |--> 12*x^2
sage: diff(f(x),x,2)
12*x^2
```

The command `diff(f,x,x)` is equivalent to `diff(f,x,2)`. SageMath also uses the object-oriented convention:

```
sage: f.diff(x)
x |--> 4*x^3
sage: f(x).diff(x)
4*x^3
sage: f(x).diff(x,4)
24
```

One advantage of this syntax is that either in the terminal window or in a notebook you can write `f.` and then hit the tab key. SageMath will then open a pop-up window telling you the many things you can do to the function `f`.

Here's how to integrate:

```
sage: f(x).integrate(x)
1/5*x^5
sage:
sage: f.integrate(x)
x |--> 1/5*x^5
sage: integrate(f(x),x)
1/5*x^5
sage: integrate(f(x),x,2,3)
211/5
```

The last one is the definite integral

$$\int_2^3 f(x)\,dx.$$

(Here's a chance to review you single-variable calculus: check that *SageMath* got it right!)

Let's try a hard one:

```
sage: g(x)=integrate(sqrt(x)*sqrt(1+x),x)
sage: print(g)
x |--> 1/4*((x + 1)^(3/2)/x^(3/2) + sqrt(x + 1)/sqrt(x))/((x + 1)^2/x^2
   - 2*(x + 1)/x + 1) - 1/8*log(sqrt(x + 1)/sqrt(x) + 1)
   + 1/8*log(sqrt(x + 1)/sqrt(x) - 1)
```

The first command produces no output, so I used `print` to tell *SageMath* to show me the result. The big mess is printed out as a single line, but I have added line breaks here.

On SageMath Cell or a notebook, you can get better looking output by using `show` instead of `print`. For example, in SageMath Cell if we enter

```
x = var('x')
g(x)=integrate(sqrt(x)*sqrt(1+x), x)
show(g(x))
```

and hit evaluate, we get something like:

$$\frac{\frac{(x+1)^{\frac{3}{2}}}{x^{\frac{3}{2}}}+\frac{\sqrt{x+1}}{\sqrt{x}}}{4\left(\frac{(x+1)^2}{x^2}-\frac{2(x+1)}{x}+1\right)}-\frac{1}{8}\log\left(\frac{\sqrt{x+1}}{\sqrt{x}}+1\right)+\frac{1}{8}\log\left(\frac{\sqrt{x+1}}{\sqrt{x}}-1\right)$$

There is also `latex(f(x))`, which is what I did to get the LaTeX code to typeset the result. While `show` produces output that is nicer to look at, the output of `print`

is easier to cut-and-paste. In general, it's best to ask *SageMath* to either `print` or `show` the outputs you want to see.

Maybe there's a way to simplify that monster? One way is to try

```
sage: g(x).simplify_full()
```

That gives

$$\frac{1}{4}(2x+1)\sqrt{x+1}\sqrt{x} - \frac{1}{8}\log\left(\frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x}}\right) + \frac{1}{8}\log\left(\frac{\sqrt{x+1}-\sqrt{x}}{\sqrt{x}}\right)$$

I guess that is a little better, but notice that it didn't use the rule that $\log(a) - \log(b) = \log(a/b)$, so we could simplify a bit more.

SageMath can also plot functions. For example

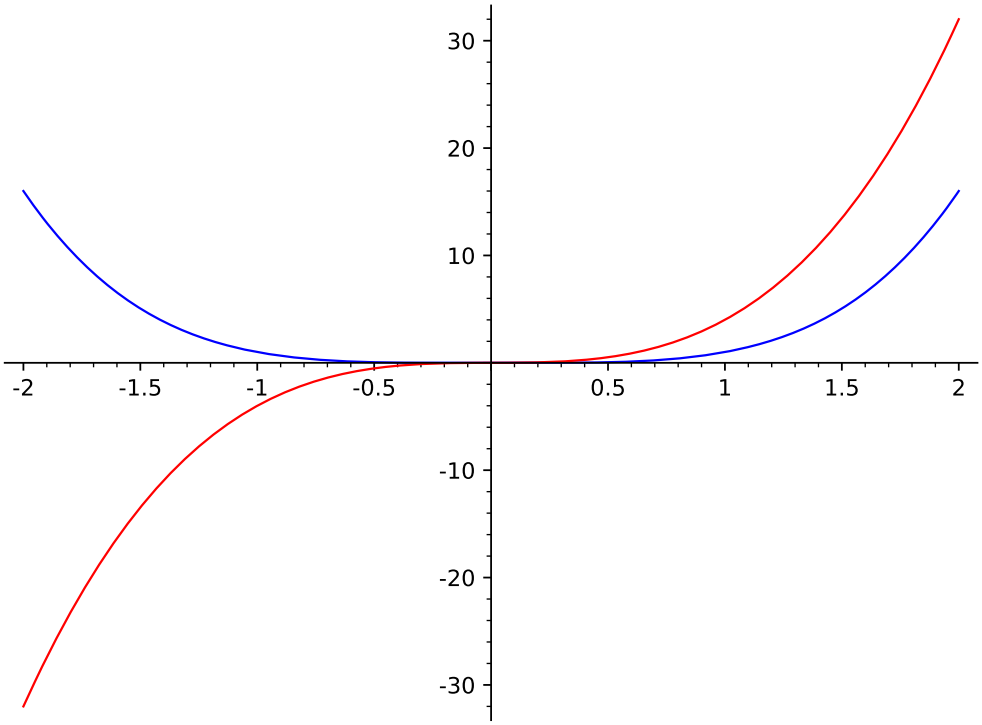
```
sage: plot(f(x),(-2,2))
```

will display a plot of our function $f(x) = x^4$. (On SageMath Cell you need define f each time, remember.) There are many options to the `plot` command; you should type `?plot` for examples if you would like to.

You can also give your plot a name and then do things to it.

```
sage: P1=plot(f(x),(-2,2),color="blue")
sage: P2=plot(diff(f(x),x),(-2,2),color="red")
sage: P=P1+P2
sage: show(P)
```

Produces this:



For a function of two variables, you can get a three-dimensional plot by

```
var('x y')
plot3d(y^2==x^3-x,(x,-2,2),(y,-2,2))
```

Notice that you need to give ranges for both variables.

There are lots of other plotting commands. To plot a curve given by an equation like $y^2 = x^3 - x$, do this in the cell server:

```
var('x y')
implicit_plot(y^2==x^3-x,(x,-2,2),(y,-2,2))
```

Or even in three variables:

```
var('x y z')
implicit_plot3d(y^2*z==x^3-x,(x,-2,2),(y,-2,2),(z,-2,2))
```

This gives a rotatable graph of the surface defined by that equation.

That's probably enough to start. To learn more, start with *Sage for Undergraduates*, which is avaliable online as well as in print. See also the Tour of Sage, then go on to theSage Tutorial. There's a lot more documentation online.