Getting started with Sage

Paul

Fall 2016

1 Obtaining Sage

A. You can download Sage (free) to your computer at: http://www.sagemath.org/.

B. You can also use the cloud-based SageMathCloud https://cloud.sagemath.com. There is a free version of this service. However, I suggest you spend the money for one of the personal subscriptions, as the performance is much better.

C. Small pieces of Sage code can be executed at the SageCell: http://sagecell.sagemath.org/

When the instructions for using Sage differ depending on these three options, then they are labeled according to these letters.

2 Resources

I have found the following resources helpful:

- The PREP tutorials: http://doc.sagemath.org/html/en/prep/index.html
- The main Sage help page: http://www.sagemath.org/help.html
- The Sage reference manual: http://doc.sagemath.org/html/en/reference/
- The Python module index: http://doc.sagemath.org/html/en/reference/py-modindex.html
- The *Sage for Undergraduates* book: http://www.gregorybard.com/Sage.html

3 Running Sage from the command line

Suppose we want to compute $2 + 3$. We can do the following:

A. On your computer, run the Sage executable, which brings up a terminal window that looks like

```
sage:
```

Type in

```
sage: 2+3
```

and then hit enter. You should see

```
sage: 2+3
5
sage:
```

B. In SageMathCloud: If you are starting for the first time, you need to create a Project. For now, I suggest creating a single project called something like "LC". Within this project, create a new Terminal file by clicking on the New button, entering a file name in to the name field, and then clicking on the button ">_Terminal". You should get some big message welcoming you to the terminal environment. You want to type in the command

```
sage
```

and then hit enter. You then see

```
sage:
```

and can proceed as above.

C. In the SageCell, simply type

```
2+3
```

in to the window and then click on the Evaluate button.

4 Worksheets

Both Sage on your machine and the SageMathCloud have things called Worksheets. This allows you to save code you have typed, which is extremely handy. Unfortunately, the items called Worksheets are not exactly the same thing in the two different environments. But still, it's handy.

A. If you are using Sage on your computer, worksheets are accessed via the Notebook environment. To start the Notebook environment, type in the command

```
sage: notebook()
```

and hit enter. The notebook environment will launch in your default web browser.

Once you are in the notebook environment, you can create a "New Worksheet". You can put a bunch of code in one of the cells and then hit SHIFT+RETURN in order to evaluate the code.

Click the "Save & Quit" button. You will be sent to the "Home" location, where you can see a list of your worksheets. Click on a worksheet to continue working.

Note: If you want to copy commands from a Sage Worksheet, it can sometimes be helpful to look at the "Text" version rather than the "Worksheet" version.

B. On the SageMathCloud, click on the New button. Type in the name of the worksheet you are constructing and then click on the "SageMath Worksheet" button in order to create the worksheet.

   Type in some code... and then type in SHIFT+RETURN in order to evaluate the code.

C. When using SageMathCell you can't save code directly. You'll need to copy and paste it to some text file if you want to save your commands.

5 SageMath as a Calculator

At its most basic, Sage functions somewhat like a calculator. You enter mathematical commands; they're evaluate; and the results are displayed. Here's an example:

```
3+4*5+8/4
```

When you hit Shift+Return you should see the following output

```
25
```

Most mathematical expressions can be entered into Sage the way you would expect. For example `+` for addtion, `-` for subtraction, `*` for multiplication, `/` for division, etc. Sage understands the standard order of operations, so `3+4/2` is interpreted as `3+(4/2)` instead of `(3+4)/2`. If you want to input the latter expression, use parenthesis. For example, the input

```
3+4/2
```

yields

```
5
```

but (with parenthesis) the input

```
(3+4)/2
```

yields

```
7/2
```

Exercise Evaluate the expression $\frac{2^4+1}{3}$.

It is important to explicitly indicate all operations. We typically write $2(3+4)$ to mean $2*(3+4)$. In Sage, you are required to type in the multiplication symbol.

Exercise Try entering $2(3+4)$ in to Sage. What sort of error message do you get? Then enter in $2*(3+4)$.

Sage can also evaluate familiar mathematical functions like sin, cos, tan, and take square roots, as the following example demonstrates.

```
sin(pi/3) + cos(pi) + sqrt(2)
```

which yields the output

```
1/2*sqrt(3) + sqrt(2) - 1
```

That output is not the easiest to read. Fortunately, we can "typeset" the output as follows:

A. Sage Notebook: Click the "Typeset" box at the top of the Worksheet before executing the code. This will render the resulting computation in a nice-looking format.

B. SageMath: Under the "Modes" menu, choose "Typeset output" which gives you the following command

   ```
   %typeset_mode True
   ```

   Hit Shift+Return to process this code. From now on, outputs will be in the nicer-looking format. To switch back, simply enter the code

   ```
   %typeset_mode False
   ```

Exercise Evaluate the expression $\sin\left(\frac{\pi}{4}+\frac{\pi}{3}\right)$. (Note that *Mathematica* gives the result in exact form, without decimals.)

Note that the exponential function e^x is written `exp(x)` in Sage.

Exercise Have Sage compute e^0.

One useful tool when using Sage for calculations is the ability to store the results of steps in variables. This way, you can break up a long calculation into steps, and combine the individual results. Here's an example:

```
width = 3*sin(pi/3)
height = 4*cos(pi/4)
area=width*height
area
```

As this example shows, you can give a name to the result of any calculation by using an "=" sign. The names of variables must not have spaces.

Exercise Have Sage execute the following code.

```
width = 3*sin(pi/3)
height = 4*cos(pi/4)
area=width*height
```

What happens? Why?

Exercise Compute the hypotenuse of a right triangle whose side lengths are $a = \sqrt{3 * 11 - 1}$ and $b = 6\sqrt{\cos\left(\frac{\pi}{3}\right)}$. Use variables store the side lengths in order to simplify the computation.

Basic plotting

Let?s now have Sage plot the function $f(x) = x^2$ for us. Because we have earlier used the letter x, the first thing we need to do is tell Sage that we want to treat x as a new (unassigned) variable. (If we had not used x earlier, then this would not be necessary.)

```
var('x')
plot(x^2,-1,2)
```

Exercise Plot the function x^3 from $x = -1$ to $x = 1$.

In order to manipulate plots, it is convenient to give them names. Here is an example of how to first give the plot of x^2 the name `Plot1` and then display that plot:

```
Plot1 = plot(x^2,-5,5)
Plot1.show()
```

This may seem more complicated... until we realize that we can put various options inside the function `Plot1.show()`. For example, we can restrict the x and y axes:

```
Plot1 = plot(x^2,-5,5)
Plot1.show(xmin=-2,xmax=3,ymin=-1,ymax=10)
```

Exercise Run the following code

```
Plot1 = plot(x^2,-5,5)
Plot1.show(xmin=-2,xmax=10,ymin=-1,ymax=50)
```

What's going on here? Why does the plot end where it does?

The following code shows how to label the axes and change the aspect ratio.

```
Plot1 = plot(x^2,-5,5)
Plot1.show(xmin=-2,xmax=2,
           ymin=-1,ymax=5,
           axes_labels=['$x$','$y$'],
           aspect_ratio=1)
```

There are many more ways to manipulate plots. Before we encounter them, however, we take a look at functions.

6 Functions

Another way to plot the function $f(x) = x^2$ is to first define the function, then have Sage plot it:

```
f(x)=x^2
plot(f,(x,-1,2))
```

If we want to use a variable other than x, we need to tell this to Sage.

```
var('t')
f(t)=t^2
plot(f,(t,-1,2))
```

Let's combine this with our knowledge of how to give names to plots:

```
var('t')
f(t)=t^2
fPlot = plot(f,(t,-10,10))
fPlot.show(xmin=-2, xmax=2,ymin=-1,ymax=5)
```

Of course, we can do things with functions besides plot them:

```
f(t)=t^2 -1
f(2)
f(f(2))
```

(Notice that I didn't "declare" the `t` variable... this is because once we've told Sage that we're using `t` as a variable, then it knows for the rest of that session. However, if you tried using this code without first the code `var('t')` earlier in the day, then Sage would be confused.)

We can even do loops:

```
f(t) = t^2
for k in [1..10]:
    f(k)
```

Exercise Have Sage give you the list of the first 20 numbers of the form

$$0, 3, 8, \ldots, k^2 - 1, \ldots$$

7 Intermediate plotting

Now that we have the ability to define functions, there are a lot of fun things we can do with plots. Here we show how to plot two functions at once.

```
f(x) = exp(-x)
g(x) = 2-x^2
fPlot = plot(f,(x,-1,2))
gPlot = plot(g,(x,-1,2))
fPlot + gPlot
```

We can have the two functions plotted in different colors, label them, etc.

```
f(x) = exp(-x)
g(x) = 2-x^2
fPlot = plot(f,(x,-1,2),color='purple',
              axes_labels=['$x$',''],
              legend_label='$f(x) = e^x$',
             show_legend='true')
gPlot = plot(g,(x,-1,2),legend_label='$g(x) = 2-x^2$')
fPlot + gPlot
```

We can even give the combined plot a name and manipulate that

```
f(x) = exp(-x)
g(x) = 2-x^2
fPlot = plot(f,(x,-1,2),color='purple',
              axes_labels=['$x$',''],
              legend_label='$f(x) = e^x$',
             show_legend='true')
gPlot = plot(g,(x,-1,2),legend_label='$g(x) = 2-x^2$')
mainPlot=fPlot + gPlot
mainPlot.show(xmin=-2, xmax=1.7,ymin=-1,ymax=2.5)
```

Exercise Plot the function $e^{-x}\sin(x)$ on the domain $0 \leq x \leq 4\pi$ in red. Then add to the graphic the plot of the function e^{-x} in purple.

Finally, we discuss plotting functions with vertical asymptotes. Try out the following code:

```
f(x) = 1/((x-2)*(x-4))
fPlot = plot(f,(x,-10,10))
fPlot.show(xmin=-1, xmax=6,ymin=-8,ymax=8)
```

The resulting graphic does not handle the asymptotes well. We can fix this by using the `detect_poles='show'` command:

```
f(x) = 1/((x-2)*(x-4))
fPlot = plot(f,(x,-10,10),detect_poles='show')
fPlot.show(xmin=-1, xmax=6,ymin=-8,ymax=8)
```

8 Other things Sage can do

The handout has focused a lot on plotting. However, Sage is much more than a fancy plotting device. Here is a non-exhaustive list of things that Sage can also do:

- Compute limits, derivatives, and integrals (symbolically and numerically)
- Make vector and contour plots like in Calculus 3. It can also do some 3D plotting.
- Multiply matrices and vectors; do computations for linear algebra (eigenstuff, etc.)
- Solve (numerically and/or symbolically) differential equations
- Various abstract algebra stuff
- Various discrete things (combo, graph theory, etc.)
- All sorts of statistics stuff (though you might also consider learning R, which the SageMathCloud can understand!)
- Simple programming (with the option to graduate to Python if you want...)

I encourage you to explore!