# Paul's very short introduction to SageMath for Calculus 3 students

Paul T. Allen

Fall 2020

## Contents

# 1 What is Sagemath?

Sage is open source mathematical software. It can be used to generate plots of mathematical objects and to do all sorts of mathematical computations. Sagemath has the advantages (and drawbacks) of being open source.

# 2 How do I use Sagemath?

It is easiest to use Sage though an online Sage Cell Server. This is a website that will process short bits of code for you. Since you cannot save code on the Sage Cell Server, **I suggest that you keep a text file somewhere with bits of code that you use frequently.** A google doc in your google drive is probably best, since you can access it from any machine.

- You can access a public Sage Cell Server here:
  https://sagecell.sagemath.org

All of the code in this document is designed for easy use within the Sage Cell environment.

If you want, you can also use Sage in the following ways:

- download Sage to your personal machine:
  https://www.sagemath.org/index.html
- use the campus Jupyter Notebook Server:
  https://jupyter.datasci.watzek.cloud

# 3 Preliminary comments

The syntax of Sage is rooted in the Python programming language. Like most programming languages, Sage is very fussy about syntax. It is important to pay attention to the details of the code you enter. For example, if you want to tell Sage to multiply 2 times $x$ in order to form the quantity $2x$ you need to explicitly indicate the multiplication and type `2*x`.

While it might be tempting to copy/paste pieces of code from this document to the SageCellServer, I do not recommend this. Using copy/paste tends to add in weird bits of formatting that sometimes causes issues with code. **I strongly encourage you to type in all code "by hand."** This has the added advantage of helping you to learn the commands!

# 4 Resources

Sagemath has extensive documentation, available here:

- https://doc.sagemath.org/html/en/reference/index.html.

The documentation is written in a very "programming-oriented" style that not everyone (including Paul!) always finds the most intuitive. There is one documentation page, however, that I do find very helpful - the 2D plotting page is great, with lots of helpful examples:

- https://doc.sagemath.org/html/en/reference/plotting/sage/plot/plot.html.

I've found the book *Computational Mathematics with SageMath* by Zimmerman *et al.* to be exceptionally helpful. The book is available here:

- https://aimath.org/textbooks/approved-textbooks/zimmermann/

Finally, search online! There is lots of helpful code out there.

# 5 First steps in getting to know Sage

Here is some Sage code that generates a plot of the function $f(t) = 2t^2$ on the interval $-1 \leq t \leq 2$.

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

Let's unpack this code line-by-line:

1. We tell Sage that we would like to use the letter $t$ as a variable.
2. We define the function $f$ by $f(t) = 2t^2$
3. We tell Sage to plot $f$ on the domain $-1 \leq t \leq 2$.

Instead of plotting the function $f$, we could evaluate it at $t = 10$. Do this with the following code

```
var('t')
f(t) = 2*t^2
f(10)
```

Let's now return to plotting. It is possible to add all sorts of options to the plotting function. For example:

```
var('t')
f(t) = 2*t^2
plot(f(t),(t,-1,2),
    ymin=-.5, ymax = 3,
    thickness=2, color='purple', linestyle='dashed',
    axes_labels=['time (s)','volume (mL)'])
```

For manipulating plots, it is better to give the plot a name and then to use that name to display the plot. This code should yield the same plot as the previous code.

```
var('t')
f(t) = 2*t^2
fplot=plot(f(t),(t,-1,2),
    ymin=-.5, ymax = 3,
    thickness=2, color='purple', linestyle='dashed')

fplot.show(axes_labels=['time (s)','volume (mL)'])
```

But the following code allows us to adjust the size of the graphic.

```
var('t')
f(t) = 2*t^2
fplot=plot(f(t),(t,-1,2),
    ymin=-.5, ymax = 3,
    thickness=2, color='purple', linestyle='dashed')

fplot.show(axes_labels=['time (s)','volume (mL)'],figsize=[4,3])
```

We can easily modify the code to save the graphic as a pdf file. (Alternatively, you can just right-click on the graphic and save it.)

```
var('t')
f(t) = 2*t^2
fplot=plot(f(t),(t,-1,2),
    ymin=-.5, ymax = 3,
    thickness=2, color='purple', linestyle='dashed')

fplot.save(filename='your-file-name.pdf',
     axes_labels=['time (s)','volume (mL)'],
     figsize=[4,3])
```

Go slowly through each line of the following code. What does each line do?

```
var('t')
f(t) = 2*t^2
g(t) = 1-t^2

fplot=plot(f(t),(t,-1,2),thickness=2, color='purple')
gplot=plot(g(t),(t,-1,2),thickness=2, color='blue')

mainplot=fplot+gplot
mainplot.show(ymin=-.5, ymax = 3,
              xmin=-.3, xmax= 1.5,
              axes_labels=['time (s)','distance (m)'],
              figsize=[7,5])
```

# 6 Plotting many functions

If we want to plot many different functions, we can first define an "empty graphic" and then add different plots to the graphic using the `+=` command.

```
var('t')
```

```
mainplot = Graphics()

mainplot += plot( cos(t), (t, 0, 2*pi))
mainplot += plot( cos(t)+1, (t, 0, 2*pi))
mainplot += plot( cos(t)+2, (t, 0, 2*pi))

mainplot.show()
```

This code can be improved by using a loop. To create a loop, we use the command `[0..2]` which creates the list $0, 1, 2$. Here is the same example as above, but with a loop.

```
var('t')

mainplot = Graphics()

for k in [0..2]:
    mainplot += plot( cos(t)+k, (t, 0, 2*pi))

mainplot.show()
```

There are a variety of ways that we can distinguish between the lines. For example, we vary the thickness of each line.

```
var('t')
mainplot = Graphics()
for k in [0..2]:
    mainplot += plot( cos(t)+k, (t, 0, 2*pi), thickness = 1+k)
mainplot.show()
```

Another thing we can do is change the color.

```
var('t')
mainplot = Graphics()
for k in [0..2]:
    mainplot += plot( cos(t)+k, (t, 0, 2*pi), rgbcolor=(.5*k,0,1))
mainplot.show()
```

A fancier thing we can do is to first make a list of rainbow colors, and then use that list for the color of each plot. In this code, the variable `ct` is the number of functions we plot.

```
var('t')

ct = 20

from sage.plot.colors import rainbow
colorz = rainbow(ct)

mainplot = Graphics()
for k in [0..ct-1]:
    mainplot += plot( cos(t)+k, (t, 0, 2*pi), color = colorz[k])
mainplot.show()
```

# 7 Region and implicit plots

To draw the hyperbola given by

$$x^2 - y^2 = 1$$

we use this code:

```
var('x','y')
implicit_plot(x^2 - y^2 ==1,
            (x,-3,3), (y,-3,3),
            axes_labels=["$x$","$y$"])
```

Shade the region of the plane determined by

$$x^2 + y^2 \leq 1.$$

```
var('x','y')
g(x,y) = x^2 + y^2
region_plot(g(x,y) <= 1, (x,-3,3), (y,-3,3))
```

Plot the surface in 3D space determined by

$$x^2 - y^2 = 1$$

```
var('x','y','z')
g(x,y) = x^2 - y^2
implicit_plot3d(g(x,y) ==1, (x, -3, 3), (y, -3, 3), (z, -3, 3))
```

If you want to save the image, click on the little "i" symbol in the corner of the graphic.

If we want to plot the region

$$1 < x^2 + 3y^2 < 3$$

then we need to split the two conditions as follows:

```
var('x','y')
region_plot([1 < x^2+3*y^2 , x^2+3*y^2 < 3],
            (x,-3,3), (y,-3,3),
            axes_labels=["$x$","$y$"])
```

# 8 Plotting parametric curves

Plot the curve

$$x(t) = t\cos(t), \quad y(t) = t\sin(t), \qquad 0 \leq t \leq 4\pi.$$

```
var('t')
x(t) = t*cos(t)
y(t) = t*sin(t)
parametric_plot([x(t),y(t)],(t,0,4*pi))
```

Plot the curve

$$x(t) = t\cos(t), \quad y(t) = t\sin(t), \quad z(t) = t, \qquad 0 \le t \le 4\pi.$$

```
var('t')
x(t) = t*cos(t)
y(t) = t*sin(t)
z(t) = t
parametric_plot3d([x(t),y(t),z(t)],(t,0,4*pi),thickness=5)
```

# 9 Plotting transformations of the plane

Let's plot the polar coordinate transformation

$$T(u,v) = (u\cos(v), u\sin(v)) \qquad 0 \le u \le 3, \quad -pi \le v \le \pi.$$

To do this, we will draw a bunch of parametric curves where $v$ is constant, and then when $u$ is constant.

```
var('u', 'v', 't')

T(u,v) = ( u*cos(v), u*sin(v) )

u_min=0
u_max = 3
u_steps=5

v_min = -pi
v_max = pi
v_steps = 10

mainplot = Graphics()
du = (u_max-u_min)/u_steps
dv = (v_max-v_min)/v_steps

for k in [0..v_steps]:
    mainplot += parametric_plot( T(t, v_min + k*dv),
                                 (t, u_min, u_max),
                                 color = "red" )

for k in [0..u_steps]:
    mainplot += parametric_plot( T(u_min + k*du, t),
                                 (t, v_min, v_max),
                                 color = "blue" )

mainplot.show()
```

Of course, we can get fancier about changing the thicknesses of the lines or varying the colors...

## 10 Plotting functions $\mathbb{R}^2 \to \mathbb{R}$

Let's plot the function

$$f(x,y) = e^{-(x^2+y^2)}$$

on the domain $-1 \leq x, y \leq 1$.

```
var('x','y')
f(x,y) = exp(-(x^2+y^2))
fplot = plot3d(f(x,y), (x,-1,1),(y,-1,1))
fplot.show()
```

How can you add the plot of the function $g(x,y) = 1 - x^2 - y^2$ to the image?

Now let's make a contour plot of the function $f$.

```
var('x','y')
f(x,y) = exp(-(x^2+y^2))
fplot = plot3d(f(x,y), (x,-1,1),(y,-1,1))
fcontour = contour_plot(f(x,y), (x,-1,1),(y,-1,1))
fcontour.show()
```

Now let's make some adjustments

```
var('x','y')
f(x,y) = exp(-(x^2+y^2))
fplot = plot3d(f(x,y), (x,-1,1),(y,-1,1))
fcontour = contour_plot(f(x,y),
                        (x,-1,1),(y,-1,1),
                        fill=false,
                        contours=20,
                        cmap=u"BrBG")
fcontour.show()
```

See the Sage documentation for more options.

## 11 Plotting vector fields

Plot the vector field

$$\vec{V} = \langle x, -y \rangle.$$

```
var('x','y')
V = vector([x,-y])
plot_vector_field(V, (x,-3,3),(y,-3,3))
```

Let's add the plot of this vector field.

$$\vec{W} = \langle y, x \rangle.$$

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

V = vector([x,-y])
```

```
Vplot = plot_vector_field(V, (x,-3,3),(y,-3,3), color="blue")

W = vector([y,x])
Wplot = plot_vector_field(W, (x,-3,3),(y,-3,3), color="red")

mainplot = Vplot+Wplot
mainplot.show()
```

We can also plot the unit vector of a given vector field

```
var('x','y')
V = vector([x,-y])
unit_V = V/V.norm()
plot_vector_field(unit_V, (x,-3,3),(y,-3,3))
```

We can also plot vector fields in 3D:

```
var('x','y','z')
V = vector([x*y, y*z, z*x])
plot_vector_field3d( V, (x,-2,2), (y,-2,2), (z,-2,2) )
```

Here is the same vector field with more options used:

```
var('x','y','z')
V = vector([x*y, y*z, z*x])
plot_vector_field3d( V,
                     (x,-2,2), (y,-2,2), (z,-2,2),
                    center_arrows=true,
                    plot_points=10)
```

# 12 Plotting parametrically defined surfaces

Plot the surface in 3D space defined by

$$x(u,v) = u\cos(v)$$
$$y(u,v) = u\sin(v)$$
$$z(u,v) = v/4$$

where

$$0 \le u \le 3 \quad \text{and} \quad 0 \le v \le 8\pi.$$

```
var('u','v')
x(t) = u*cos(v)
y(t) = u*sin(v)
z(t) = v/4
parametric_plot3d([x(u,v),y(u,v),z(u,v)],(u,0,3),(v,0,8*pi),opacity=0.7)
```

## 13 Some algebra

Suppose we want to solve the system

$$x^2 + y^2 = 1, \quad y = 2\lambda x, \quad x = 2\lambda y$$

```
var('x','y','L')
sol=solve([x^2 + y^2 == 1, y==2*L*x, x== 2*L*y],x,y,L)
show(sol)
```

## 14 Differentiation

Compute

$$\frac{\partial}{\partial x}\left(e^{-(x^2+y^2)}\right)$$

```
var('x','y')
f(x,y) = exp(-(x^2+y^2))
result = diff(f(x,y),x)
show(result)
```

Here is some code to compute the gradient of

$$f(x, y, z) = xyz.$$

```
var('x','y','z')
f(x,y,z) = x*y*z
gradient = f.diff()
gradient(x,y,z)
```

In order to compute the Hessian, we specify that we want the second derivative.

```
var('x','y','z')
f(x,y,z) = x*y*z
hessian = f.diff(2)
hessian(x,y,z)
```

## 15 Integration

To compute

$$\int_{x=0}^{1}\int_{y=0}^{2} xy\,dy\,dx$$

we use the code

```
var('x','y')
f = x*y
f.integrate(y,0,2).integrate(x,0,1)
```

To compute

$$\int_{x=0}^{1}\int_{y=0}^{x^2} xy\,dy\,dx$$

we use the code

```
var('x','y')
f = x*y
f.integrate(y,0,x^2).integrate(x,0,1)
```

Thus there are several ways to compute

$$\iint_{\text{unit disk}} (x^2+y^2)dA$$

See if you can figure out what each of these pieces of code do.

```
var('x','y')
f(x,y) = x^2 + y^2
result = f.integrate( x, -sqrt(1-y^2),sqrt(1-y^2) ).integrate(y,-1,1)
show(result)
```

```
var('r','th')
f(x,y) = x^2 + y^2
jac = r
result = ( f(r*cos(th),r*sin(th))*jac ).integrate(r,0,1).integrate(th,-pi,pi)
show(result)
```