Sage demo: Introduction (Fields Institute, Toronto 2018)

(by Vincent Delecroix, Bordeaux, France)

All worksheets for the workshop are available on the wiki https://wiki.sagemath.org/days96

Sage (or SageMath) is an open source software (GPL-licensed) for mathematics which interfaces many softwares and libraries, e.g.:

- PARI/GP (number theory),
- GAP (group theory),
- Maxima (symbolic calculus),
- The SciPy suite (numpy, scipy, matplotlib)
- GMP (C library for arbitrary precision integers)
- MPFR (C library arbitrary precision floating point numbers)
- NTL (C++ library for number theory)
- and many more

Python, Ipython and Jupyter

Sage is based on the Python language, which is very popular (web programming, graphical interaces, scripts, ...) and easy to learn.

As of version 8.3, Sage uses Python 2 and is in a phase transition towards Python 3.

The way you will mostly interact with Sage is through IPython which is an enhanced Python interpreter. Sage and IPython can be used in two modes: in a console or through Jupyter. This document is an example of a Jupyter worksheet.

Python is an expressive langage

$\left\{17n \;\middle|\; n \in \{0,1,\ldots,9\} \text{ and } n \text{ is odd}\right\}$

```
sage: S = {17*n for n in range(10) if n%2 == 1}
sage: S

sage: 124 in S

sage: sum(S)

sage: {3*i for i in S}
```

To execute the content of one of the code cells above, you need to press `<SHIFT> + <ENTER>` or `<CTRL> + <ENTER>`. If you only press `<ENTER>` it will either bring you in *edit mode* or insert a linebreak.

To access the Jupyter help, pass in *command mode* with `<ESC>` and then press `<h>`.

Sage add some mathematical objects and functions

```
sage: 8324074213.factor()
```

```
sage: m = matrix(ZZ, 3, 3, [0,3,-2,1,4,3,0,0,1])
```

```
sage: m.eigenvalues()
```

```
sage: m.inverse()
```

As in mathematics, the base ring on which an object is defined matters:

```
sage: R.<x> = PolynomialRing(ZZ, 'x')  # ZZ = ring of integers
```

```
sage: R
```

```
sage: P = 6*x^4 + 6*x^3 - 6*x^2 - 12*x - 12
sage: P.factor()
```

```
sage: P2 = P.change_ring(QQ)     # QQ = field of rational numbers
```

```
sage: P2.factor()
```

```
sage: P3 = P.change_ring(AA)     # AA = field of real algebraic numbers
```

```
sage: P3.factor()
```

```
sage: P4 = P.change_ring(QQbar)  # QQbar = field of complex algebraic numbers
```

```
sage: P4.factor()
```

Autocompletion and documentation

The Sage Jupyter notebook (actually IPython) relies on two things to browse the software and its documentation.

First, you can autocomplete names using the `<TAB>` key. Pressing `<TAB>` in the cell below will show you all the objects in Sage whose names start with `in`

```
sage: in
```

(the list should start with `in`, `incomplete_gamma`, `infinity`, ...). When there is only one possible completion, the begining of the word will be automatically completed. Pressing <TAB> below will gives you immediately the sole completion `incomplete_gamma`

```
sage: inc
```

while pressing `<TAB>` in the following will extend the string by one letter and will propose you two possible endings.

```
sage: inf
```

The second useful feature of the Sage Jupyter notebook (which is again an IPython feature) is accessing to the documentation of a single function or object which is achieved via the question mark `?`. Namely, pressing `<SHIFT> + <ENTER>` in the cell below will show you the documentation of the gamma function

```
sage: gamma?
```

As you can notice, the documentation often comes with explanations *and* examples.

Object oriented

Python is an object-oriented language. That means that actions that can be performed on objects (ie, a *function* in computer programming) are attached to the object rather than being globally defined names. We already saw this with `P.change_ring(QQ)` above. The name `change_ring` is a function attached to the object `P`. We say that it is a *class function* or a *method*. A class function is always written in snake case (if this happens to not be the case for some example you encounter, you can report it as a bug on the pad: https://mensuel.framapad.org/p/sagedays96).

Tab completion also works with a class function. The first cell below defines a symbolic function `f`. And using tab completion in the second cell you can figure out how to compute its integral.

```
sage: f(x) = sin(x)^2 -sin(x)
sage: f
```

```
sage: f.in
```

Exercise: Draw the Petersen graph. Which algorithm is used to compute the vertex cover of this graph ?

```
sage: G = grap
```

```
sage: # edit here
```

```
sage: G.vertex
```

```
sage: # edit here
```

Lost?

If you are lost, stuck with something and you can not find any answer in the documentation just ask your question on the ask forum.

Calculator

Integration (symbolic):

```
sage: integral(e^(-x^2), x, -Infinity, Infinity)
```

```
sage: integral(1/sqrt(1+x^3), x, 0, 1)
```

Integration (floating point numeric)

```
sage: numerical_integral(1/sqrt(1+x^3), 0, 1)
```

Integration (certified numeric with arbitrary precision). This example would only work if you have a Sage version >= 8.2

```
sage: R = ComplexBallField(128)
sage: R.integral(lambda x,_: 1/(1+x^3).sqrt(), 0, 1)
```

```
sage: R = ComplexBallField(1024)
sage: R.integral(lambda x,_: 1/(1+x^3).sqrt(), 0, 1)
```

Computing roots

```
sage: f(x) = x^5 - 1/3*x^2 - 7*sin(2*x) + 1
```

```
sage: plot(f, xmin=-2, xmax=2)
```

```
sage: r1 = find_root(f,-2,-1)
sage: r1

sage: r2 = find_root(f,0,1)
sage: r2

sage: r3 = find_root(f,1,2)
sage: r3

sage: plot(f, xmin=-2, xmax=2) + point2d([(r1,0),(r2,0),(r3,0)], pointsize=50, colo
```

Latex:

```
sage: M = Matrix(QQ, [[1,2,3],[4,5,6],[7,8,9]]); M

sage: latex(M)

sage: M.parent()

sage: latex(M.parent())
```

Some 3d Graphics:

```
sage: x, y = SR.var('x,y')
sage: plot3d(sin(x-y)*y*cos(x), (x,-3,3), (y,-3,3))
```

Interaction:

```
sage: var('x')
sage: @interact
sage: def g(f=sin(x)-cos(x)^2, c=0.0, n=(1..30),
....:         xinterval=range_slider(-10, 10, 1, default=(-8,8), label="x-interval"
....:         yinterval=range_slider(-50, 50, 1, default=(-3,3), label="y-interval"
....:     x0 = c
....:     degree = n
....:     xmin,xmax = xinterval
....:     ymin,ymax = yinterval
....:     p   = plot(f, xmin, xmax, thickness=4)
....:     dot = point((x0,f(x=x0)),pointsize=80,rgbcolor=(1,0,0))
....:     ft = f.taylor(x,x0,degree)
....:     pt = plot(ft, xmin, xmax, color='red', thickness=2, fill=f)
....:     show(dot + p + pt, ymin=ymin, ymax=ymax, xmin=xmin, xmax=xmax)
....:     pretty_print(html('$f(x)\;=\;%s$'%latex(f)))
....:     pretty_print(html('$P_{%s}(x)\;=\;%s+R_{%s}(x)$'%(degree,latex(ft),degree
```

Licence and development

SageMath is distributed under a GPL licence which means that you can freely download the software, have access to its source code and you can redistribute it in any form you like as long as you use a GPL-compatible licence.

The source code access can be done with two question marks `??` directly in a cell

```
sage: gamma??
```

The Sage project began in 2005 under the inpetus of William Stein and is now being developed by hundreds of developers around the world. Most of the development is happening on the trac server and the sage-devel mailing list.

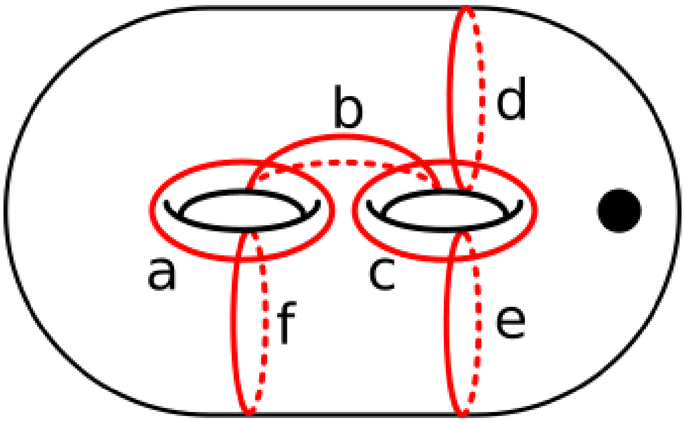
Extra packages for geometry and dynamics

There are several packages built on top of Sage dedicated to geometry and dynamics. We will study them in more depth during this Sage Days 96. Let us mention

- **flipper: mapping classes (homeomorphisms** of surfaces)
- snappy: 3-d hyperbolic geometry
- surface_dynamics: interval exchange transformations, origamis and more
- flatsurf: translation surfaces (affine transformation, linear flow, etc)

These packages are not installed by default in Sage. The instructions to install them are available on the wiki https://wiki.sagemath.org/days96

Let us use flipper and check the braid relation on a surface of genus 2 with 1 puncture. The surface $S_{2,1}$ (that is builtin in flipper) is depicted on the picture below



Here is how to play with the Dehn-twist around the curves *a* and *b*

```
sage: import flipper

sage: S = flipper.load('S_2_1')
sage: a = S.mapping_class('a')
sage: b = S.mapping_class('b')
sage: print(a*b*a == b*a*b)    # braid relation
sage: print(a*b == b*a)        # these do not commute
```

With snappy installed you can investigate 3-dimensional hyperbolic manifolds. It comes with an extensive database of them. Here we compute some invariantes of the manifold "m015" from the database.

```
sage: import snappy

sage: M = snappy.Manifold("m015")
sage: M

sage: M.cusp_info()

sage: M.alexander_polynomial()

sage: M.volume()

sage: M.complex_volume()
```

Flipper can be used to construct mapping tori of pseudo-Anosov homeomorphism and send them to snappy for further analysis

```
sage: import flipper
sage: import snappy
```

```
sage: S = flipper.load('S_2_1')
sage: a = S.mapping_class('a')
sage: b = S.mapping_class('b')
sage: C = S.mapping_class('C')
sage: d = S.mapping_class('d')
sage: f = a * b * C * d
sage: f.nielsen_thurston_type()

sage: M = snappy.Manifold(f.bundle())
sage: M.volume()
```

With surface_dynamics installed you can play with origamis (an origami is a finite cover of a square torus ramified at most over the origin)

```
sage: import surface_dynamics

sage: o = surface_dynamics.Origami('(1,2)', '(1,3)')

sage: o.stratum()

sage: o.plot()

sage: V = o.veech_group()
sage: print V
sage: print V.nu2(), V.nu3(), V.ncusps()

sage: V.farey_symbol().fundamental_domain()
```

Finally, flatsurf allows you to construct translation surface from polygons and play with translation flow. Below we construct saddle connections on the double pentagon

```
sage: import flatsurf

sage: S = flatsurf.translation_surfaces.veech_double_n_gon(5)
sage: S.plot()

sage: sc = S.saddle_connections(20)
sage: S.plot() + sum(s.plot(color='red') for s in sc)
```

Flipper pseudo-Anosov can also be sent to flatsurf as follows

```
sage: import flipper
sage: import flatsurf

sage: S = flipper.load('S_2_1')
sage: a = S.mapping_class('a')
sage: b = S.mapping_class('b')
sage: C = S.mapping_class('C')
sage: d = S.mapping_class('d')
sage: f = a * b * C * d
sage: S = flatsurf.translation_surfaces.from_flipper(f)
```

The essentials

- Use Tab completion to browse and access documentation with `?`
- The main website: http://www.sagemath.org/ (including some HTML documentation)
- A forum to ask your questions about Sage: http://ask.sagemath.org
- A book "Calcul mathématique avec Sage"/"Computational Mathematics with SageMath"/"Rechnen mit Sage", a book about Sage (in french, english and german): http://sagebook.gforge.inria.fr/

What’s next?

system-message

WARNING/2 in `intro.en.rst`, line 455

Title underline too short.

```
What's next? ----------
```

backrefs:

Go to the wiki https://wiki.sagemath.org/days96 and choose a worksheet. If you just start with Sage or does not know much about Python programming, it would be a good idea to work on the 6 Programming worksheets ("First steps with Sage", "Learn about for loops", etc).

You can also have a look at the Sage documentation. It can be accessed from a Jupyter notebook by clicking on "Help". Or from the main website http://www.sagemath.org/.

Authors:

- Thierry Monteil
- Vincent Delecroix