Monografías Matemáticas *García de Galdeano* **39**, 167–176 (2014)

A FRAMEWORK FOR FREE, FINITELY PRESENTED AND BRAID GROUPS IN SAGE

Miguel Angel Marco-Buzunariz

Abstract. We present a framework for computing with free, finitely presented and braid groups in Sage developed by the author. This framework has been included in Sage since version 5.7beta3. The capabilities of the framework are shown through several examples.

§1. Introduction

Sage[7] is free/open source computer algebra system, distributed under the GNU general public license. The project was started by William Stein in 2005. Its mission statement was (and still is) "*create a viable Free/Open Source alternative to Magma, Maple, Mathematica and Matlab*". Stein had developed previously a good amount of code for Magma, and decided to start Sage because of the restrictions of the Magma license, that, for instance, prevented him from sharing his code with other researchers.

In order to develop Sage, Stein decided to reuse as much freely available code as possible, following the citation "build the car instead of reinventing the wheel". This means that internally Sage makes a big part of the computations by using specialized libraries or programs. A (not exhaustive at all) list of those software pieces could be:

- **Python** for general programming language and interfacing with the rest of the components.
- **Maxima**[4] and **sympy** for symbolic manipulation.
- **Atlas**, **Numpy** and **Scipy**[3] for numerical computations.
- **Singular**[1] for computations on polynomial rings.
- **Gap**[2] for group theory.
- **Pari**[5] for computations related with number theory.
- **R**[6] for statistics.
- **Matplotlib** for graphical visualization.

This list should be taken as just some of the most representative elements. The total amount of software packages that are bundled in sage is around 100.

Sage can be used from a command line interface or through a web-based graphical user interface. This interface allows the user to connect to a machine running Sage (that can be a remote computer or his own personal computer) through a web browser. The project webpage contains plenty of documentation, downloads of the program, and links to servers where Sage can be tested online. The program is available in the form of source code, binary packages

for OSX, several linux flavours. Windows users can install Sage by downloading a virtual machine with Sage already installed in it.

All the code of Sage is not only freely available, but also can be easily seen by the user in the same way that he consults the help of a command. The development process is peer reviewed and completely open. Everybody can participate either by writing new code or reviewing the code written by others. This makes the transition from user to developer very smooth compared to other software projects.

1.1. Some examples

Here we will see some examples of computations for which Sage uses internally other systems.

First we show how a Groebner basis is computed with Singular:

```
R.<x1,x2,x3>=QQ[]
R
```

```
Multivariate Polynomial Ring in x1, x2, x3 over Rational Field
```

```
I=R.ideal(x1^2+x2^3-x1*x3,x1+x2+x3+1)
I
```

```
Ideal (x2^3 + x1^2 - x1*x3, x1 + x2 + x3 + 1) of Multivariate Polynomial
Ring in x1, x2, x3 over Rational Field
```

```
I.groebner_basis()
```

```
[x2^3 + x2^2 + 3*x2*x3 + 2*x3^2 + 2*x2 + 3*x3 + 1, x1 + x2 + x3 + 1]
```

A symbolic integral is done with Maxima:

```
var('x')
integrate(cos(2*x)/sin(x),x)
```

```
1/2*log(cos(x) - 1) - 1/2*log(cos(x) + 1) + 2*cos(x)
```

And finally the lower central series of a permutation group is computed with Gap:

```
a=Permutation("(1,2,3)(4,5,6)")
b=Permutation("(1,4)(5,2)")
P=PermutationGroup([a,b])
P
```

```
Permutation Group with generators [(1,2,3)(4,5,6), (1,4)(2,5)]
```

```
P.lower_central_series()
```

```
[Subgroup of (Permutation Group with generators [(1,2,3)(4,5,6),
(1,4)(2,5)]) generated by [(1,2,3)(4,5,6), (1,4)(2,5)], Subgroup
of (Permutation Group with generators [(1,2,3)(4,5,6), (1,4)(2,5)])
generated by [(1,4)(3,6), (1,4)(2,5)]]
```

§2. Parents and elements

Algebraic structures, such as groups, rings or vector spaces, are represented in Sage by objects called parents. Lets see some examples of parents and how to construct new ones.

```
QQ
```

```
Rational Field
```

```
ZZ
```

```
Integer Ring
```

```
SR
```

```
Symbolic Ring
```

```
Zmod(6)
```

```
Ring of integers modulo 6
```

```
QQbar
```

```
Algebraic Field
```

```
MatrixGroup([matrix([[1,0],[-1,1]]),matrix([[1,1],[0,1]])])
```

```
Matrix group over Integer Ring with 2 generators:
 [[[1, 0], [-1, 1]], [[1, 1], [0, 1]]]
```

```
PolynomialRing(ZZ,'x,y,z')
```

```
Multivariate Polynomial Ring in x, y, z over Integer Ring
```

There are also objects that represent the elements of the parents. Each element belongs to a single parent, and they can be operated with.

```
2+5
```

```
7
```

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

```
x^y
```

```
R.<x,y,z>=PolynomialRing(ZZ)
x*y*(z^2-z+x*y)
```

```
x^2*y^2 + x*y*z^2 - x*y*z
```

The elements can be converted from one parent to another (if that makes sense):

```
R=Zmod(8)
```

```
R(16)
```

```
0
```

```
16.parent()
```

```
Integer Ring
```

```
R(15).parent()
```

```
Ring of integers modulo 8
```

```
ZZ(R(15))
```

```
7
```

When we try to operate with elements of different parents, sage tries to find a common parent where the result makes sense.

```
R.<x>=ZZ[]
```

```
(x+1).parent()
```

```
Univariate Polynomial Ring in x over Integer Ring
```

```
(3/5).parent()
```

```
Rational Field
```

```
(x+1+3/5).parent()
```

```
Univariate Polynomial Ring in x over Rational Field
```

§3. Free Groups

Now we present three new types of parents and their corresponding elements, that have been introduced in Sage recently by the author. The first of them is free groups, which internally use Gap for the computations. We will illustrate their creation with some examples:

```
F=FreeGroup(3)
F
```

```
Free Group on generators {x0, x1, x2}
```

If we want to have the generators available as newly created variables, we can use the procedure "inject_variables":

```
F.inject_variables()
```

```
Defining x0, x1, x2
```

Another way to obtain the same result in only one step is:

```
F.<x0,x1,x2>=FreeGroup()
```

The name of the variables can also be decided in the moment of the construction:

```
FreeGroup('a,b,c,d')
```

```
Free Group on generators {a, b, c, d}
```

We can operate normally with the elements of the group.

```
x0*x1/x2
```

```
x0*x1*x2^-1
```

```
x0.parent()
```

```
Free Group on generators {x0, x1, x2}
```

We can also compute fox derivatives of an element with respect to a generator. The result lives in the group algebra.

```
(x2*x1/x0/x2).fox_derivative(x2)
```

```
B[1] - B[x2*x1*x0^-1*x2^-1]
```

```
(x2*x1/x0/x2).fox_derivative(x2).parent()
```

```
Group algebra of Free Group on generators {x0, x1, x2} over Integer Ring
```

Each element can be represented by its Tietze list. This is a list of integers that represent the generators whose product gives place to the element, with a minus sign when there is the inverse of a generator.

```
(x0*x1*x2).Tietze()
```

```
(1, 2, 3)
```

```
(x2^(-2)*x1^2).Tietze()
```

```
(-3, -3, 2, 2)
```

The Tietze list can also be used to generate elements:

```
F([1,1,1,1,2,-3,-3,-3])
```

```
x0^4*x1*x2^-3
```

§4. Finitely presented groups

The second type of parents that we present are finitely presented groups. They are created as quotients of free groups.

```
G=F.quotient([(x0*x1)^3,x1*x2/x1/x2,x2*x0/x2*x0,x1*x2*x0])
G
```

```
Finitely presented group < x0, x1, x2 | x0*x1*x0*x1*x0*x1,
x1*x2*x1^-1*x2^-1, x2*x0*x2^-1*x0, x1*x2*x0 >
```

Note that, in this case, the elements x_0, x_1, x_2 belong to the free group. If we want to refer to the elements of the quotient, we have to inject the variables again.

```
x0.parent()
```

```
Free Group on generators {x0, x1, x2}
```

```
G.inject_variables()
x0.parent()
```

```
Defining x0, x1, x2
Finitely presented group < x0, x1, x2 | x0*x1*x0*x1*x0*x1,
x1*x2*x1^-1*x2^-1, x2*x0*x2^-1*x0, x1*x2*x0 >
```

But conversion between the free group and the quotient works fine.

```
F(x0), G(F(x0))
```

```
(x0, x0)
```

Here are some examples of what can be done with finitely presented groups.

The abelian invariants is a list $(n_0, \dots n_i)$ such that the abelianized of the group is isomorphic to

$$\frac{\mathbb{Z}}{n_0\mathbb{Z}} \oplus \cdots \oplus \frac{\mathbb{Z}}{n_i\mathbb{Z}}$$

```
G.abelian_invariants()
```

```
(2, 3)
```

We can ask sage to try to simplify the presentation.

```
G.simplified()
```

```
Finitely presented group < x0, x2 | x0^2, x2^3, x2^-1*x0*x2*x0 >
```

In general, the problem of computing the size of a finitely presented group is undecidable. However, Gap includes some methods that can give an answer in some cases. They must be used with caution, since it may happen that the method does not stop, running until it consumes all the available memory and finally get killed by the system.

```
G.cardinality()
```

```
6
```

We can compute the Alexander matrix (that is, the matrix of the fox derivatives of the relations). Note that it is not an invariant of the group, but an invariant of the presentation. Here we show the matrix of the original presentation and the matrix of the simplified presentation.

```
G.alexander_matrix()
```

$$\begin{pmatrix} B_1 + B_{x_0\cdot x_1} + B_{x_0\cdot x_1\cdot x_0\cdot x_1} & B_{x_0} + B_{x_0\cdot x_1\cdot x_0} + B_{x_0\cdot x_1\cdot x_0\cdot x_1\cdot x_0} & 0 \\ 0 & B_1 + (-1)B_{x_1\cdot x_2\cdot x_1^{-1}} & B_{x_1} + (-1)B_{x_1\cdot x_2\cdot x_1^{-1}\cdot x_2^{-1}} \\ B_{x_2} + B_{x_2\cdot x_0\cdot x_2^{-1}} & 0 & B_1 + (-1)B_{x_2\cdot x_0\cdot x_2^{-1}} \\ B_{x_1\cdot x_2} & B_1 & B_{x_1} \end{pmatrix}$$

```
G.simplified().alexander_matrix()
```

```
[              B[1] + B[x0]                                0]
[                         0    B[1] + B[x2] + B[x2^2]]
[B[x2^-1] + B[x2^-1*x0*x2]   -B[x2^-1] + B[x2^-1*x0]]
```

If the group is finite, Sage can try to express it as a permutation group. The same warning as before applies here. Note that this presentation is not minimal, and that the generators correspond to the generators of the initial group-

```
G.as_permutation_group()
```

```
Permutation Group with generators [(1,2)(3,6)(4,5), (1,3,5,2,6,4),
(1,5,6)(2,4,3)]
```

```
G.simplified().as_permutation_group()
```

```
Permutation Group with generators [(1,2)(3,5)(4,6), (1,3,4)(2,5,6)]
```

§5. Braid Groups

The last parent presented is the braid groups. These are a particular case of finitely presented group, but there are several specific methods for them. They are created in a similar way as before:

```
BraidGroup(4,'x,y,z')
```

```
Braid group on 4 strands
```

```
B=BraidGroup(4)
```

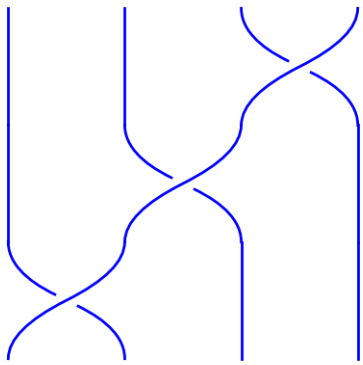
```
B.inject_variables()
```

```
Defining s0, s1, s2
```

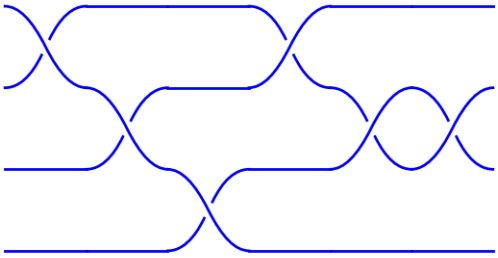
```
a=s0*s1*s2
```

Braids can be plotted. By default they are represented from bottom to top. The option "orientation" allows to change it to top to bottom, or left to right.

```
a.plot()
```



```
B([1,2,3,-1,-2,-2]).plot(orientation='left-right')
```

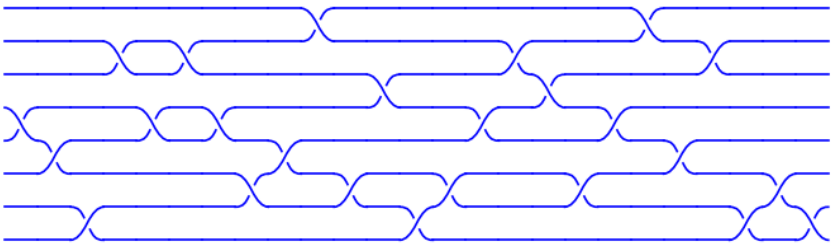


Now we create a random braid with 8 strands and around 30 generators.

```
C=BraidGroup(8)
l=[randint(-7,7) for i in range(30)]
l=filter(lambda a:a!=0,l)
b=C(l)
b
```

```
s3*s4^-1*s6^-1*s1*s3*s1*s3*s5*s4^-1*s0*s5^-1*s2*s6^-1*s5^-1*s3^-1*
s1^-1*s2*s5^-1*s3^-1*s0*s4^-1*s1^-1*s6^-1*s5^-1*s6
```

```
b.plot(orientation='left-right',gap=0.1)
```



We can also compute several invariants such as the left normal form, Burau matrix, Lawrence-Krammer-Bigelow matrix or the induced permutation.

```
c=B([1,-2,-2,-3,1,1,1,2,2])
c.left_normal_form()
```

```
(s0^-1*s1^-1*s2^-1*s0^-1*s1^-1*s0^-2*s1^-1*s2^-1*s0^-1*s1^-1*s0^-1,
s0*s2*s1*s0, s1*s0*s2, s2*s1*s0, s0, s0, s0*s1, s1)
```

```
c.burau_matrix()
```

$$\begin{pmatrix} t^4-2t^3+3t^2-3t+2 & -t^6+3t^5-6t^4+7t^3-5t^2+2t & t^6-3t^5+5t^4-5t^3+2t^2 & t-1 \\ -t^3+t^2-t+1 & t^5-2t^4+3t^3-2t^2+t & -t^5+2t^4-2t^3+t^2 & 0 \\ t-2+2t^{-1}-t^{-2} & -t^3+3t^2-4t+3-t^{-1} & t^3-3t^2+3t-1 & 1-t^{-1}+t^{-2} \\ 0 & -1+t^{-1} & 1 & 1-t^{-1} \end{pmatrix}$$

```
p=b.permutation()
p
```

```
[7, 2, 5, 4, 6, 1, 3, 8]
```

```
p.cycle_tuples()
```

```
[(1, 7, 3, 5, 6), (2,), (4,), (8,)]
```

```
m=a.LKB_matrix()
m
```

$$\begin{pmatrix} 0 & 0 & -x^6y + x^5y - x^3y & 0 & -x^6y + x^5y & -x^6y + x^5y \\ 0 & 0 & -x^5y + x^4y & 0 & -x^5y + x^4y - x^3y & -x^5y + x^4y \\ 0 & 0 & -x^4y + x^3y & 0 & -x^4y + x^3y & -x^4y \\ 1 & 0 & x^5y - 2x^4y + x^3y - x^2 + x & 0 & x^5y - 2x^4y + 2x^3y - x^2y & x^5y - 2x^4y + x^3y \\ 0 & 1 & x^4y - 2x^3y + x^2y - x + 1 & 0 & x^4y - 2x^3y + x^2y & x^4y - x^3y \\ 0 & 0 & x^3y - 2x^2y + xy & 1 & x^3y - 2x^2y + xy - x + 1 & x^3y - x^2y \end{pmatrix}$$

§6. Conclusions and future of the project

We have outlined a few capabilities of this framework that allow basic computations with groups. The implementation should be considered as a first step, with basic functionalities. The intention of the author is to keep improving it, adding more and more methods to the already defined objects.

Some possible future lines of work would be:

- Integration of the `kbmag` Gap package, that allows to find confluent rewriting systems, or automatic structures for certain finitely presented groups. This would allow, in some cases, to have a reduced form for each element of the group.
- Develop morphisms between the defined parents, allowing then to define, for instance, the Alexander matrix with respect to a fixed character.
- Implement several methods for braids: conjugacy problem, super summit sets, gcd and lcm...
- Translation between the Artin and the Birman-Ko-Lee presentations of braids, and computing also de previous invariants with respect to this last one.
- Classification of the dynamic of braids, Bestvina-Handel algorithm and train tracks.
- Use the braid group as a basis to build a class for knots and links.

The author would welcome very much every possible feedback, suggestion or help from people interested in the framework and its possible improvements. The possible improvements in a given direction may largely depend on the interest shown by the users, and also on the help that other possible developers can give.

References

[1] Decker, W., Greuel, G., Pfister, G., and Schönemann, H. Singular 3-1-6 — A computer algebra system for polynomial computations. Available from: `http://www.singular.uni-kl.de`.

[2] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.6.4*, 2013. Available from: `http://www.gap-system.org`.

[3] Jones, E., Oliphant, T., Peterson, P., et al. *SciPy: Open source scientific tools for Python*, 2001. Available from: http://www.scipy.org/.

[4] Maxima. *Maxima, a Computer Algebra System. Version 5.25.1*, 2011. Available from: http://maxima.sourceforge.net/.

[5] The PARI Group. *PARI/GP, version* 2.5.3. Bordeaux, 2012. Available from: http://pari.math.u-bordeaux.fr/.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria, 2013. Available from: http://www.R-project.org.

[7] Stein, W., et al. *Sage Mathematics Software (Version 5.7)*. The Sage Development Team, 2013. Available from: http://www.sagemath.org.

Miguel Ángel Marco-Buzunáriz
Centro Universitario de la Defensa,
Academia General Militar,
Ctra. de Huesca s/n,
50090 Zaragoza, Spain
mmarco@unizar.es