giacpy_sagedoc

```
import giacpy_sage
```

```
// Giac share
root-directory:/home/fred-dev/sage/develop/sage.develop/local
giac/
// Giac share
root-directory:/home/fred-dev/sage/develop/sage.develop/local
giac/
Help file
/home/fred-dev/sage/develop/sage.develop/local/share/giac/doc
e_cas not found
Added 0 synonyms
```

```
giacpy_sage?
```

File: /home/fred-dev/sage/develop/sage.develop/local/lib/python2.7/site-packages/giacpy_sage.

Type: <type 'module'>

Definition: giacpy_sage([noargspec])

Docstring:

Name:	giacpy_sage
Summary:	A Cython frontend to the c++ library giac. (Computer Algebra System)
License:	GPL v2 or above
Home-page:	http://www.math.jussieu.fr/~han/xcas/giacpy/
Author:	Frederic Han
Author-email:	frederic.han@imj-prg.fr

This is the sage version of giacpy. Since 0.6 it is named giacpy_sage to avoid confusions

- Giacpy is an interface to the c++ giac library. This interface is built with cython, and the re
- Giac is a general purpose Computer algebra system by Bernard Parisse released under G
  - http://www-fourier.ujf-grenoble.fr/~parisse/giac.html
  - It is build on C and C++ libraries: NTL (arithmetic), GSL (numerics), GMP (b
  - It provides fast algorithms for multivariate polynomial operations (product, G
  - symbolic computations: solver, simplifications, limits/series, integration, sum
  - Linear Algebra with numerical or symbolic coefficients.

AUTHORS:

- Frederic Han (2013-09-23): initial version

EXAMPLES:

- The class Pygen is the main tool to interact from python/sage with the c++ library giac via

  The initialisation of a Pygen just create an object in giac, but the mathematical computatio

  Here A is a Pygen element, and it is ready for any giac function.

```
sage: from giacpy_sage import *    # random
//...
sage: A=Pygen('2+2');A
2+2
sage: A.eval()
4
```

In general, you may prefer to directly create a Pygen and execute the evaluation in giac. ˜

```
sage: a=libgiac('2+2');a;isinstance(a,Pygen)
4
True
```

- Most common usage of this package in sage will be with the libgiac() function. This functi

```
sage: from giacpy_sage import libgiac,giacsettings
sage: x,y,z=libgiac('x,y,z');  # add some giac objects
sage: f=(x+3*y)/(x+z+1)^2 -(x+z+1)^2/(x+3*y)
sage: f.factor()
(3*y-x^2-2*x*z-x-z^2-2*z-1)*(3*y+x^2+2*x*z+3*x+z^2+2*z+1)/((x+z+1
sage: f.normal()
(-x^4-4*x^3*z-4*x^3-6*x^2*z^2-12*x^2*z-5*x^2+6*x*y-4*x*z^3-12*x*z
```

- To obtain more hints on giacpy_sage consider the help of the `libgiac` function.

```
sage: libgiac?
```

- Some settings of giac are available via the `giacsettings` element. (Ex: maximal numbe

```
sage: R=PolynomialRing(QQ,8,'x')
sage: I=sage.rings.ideal.Katsura(R,8)
sage: giacsettings.proba_epsilon=1e-15;
sage: Igiac=libgiac(I.gens());
sage: time Bgiac=Igiac.gbasis([R.gens()],'revlex')
Running a probabilistic check for the reconstructed Groebner basi
Time: CPU 0.46 s, Wall: 0.50 s
sage: giacsettings.proba_epsilon=0;
sage: Igiac=libgiac(I.gens());
sage: time Bgiac=Igiac.gbasis([R.gens()],'revlex')
Time: CPU 2.74 s, Wall: 2.75 s
```

```
sage: from giacpy_sage import *
sage: x=libgiac('x');f=1/(2+sin(5*x))
sage: oldrep=2/5/sqrt(3)*(atan((2*tan(5*x/2)+1)/sqrt(3))+pi*floor
sage: newrep=2/5/sqrt(3)*(atan((-sqrt(3)*sin(5*x)+cos(5*x)+2*sin(
sage: ((f.int()-newrep)*(f.int()-oldrep())).normal()
0
sage: f.series(x,0,3)
1/2-5/4*x+25/8*x^2-125/48*x^3+x^4*order_size(x)
sage: libgiac(sqrt(5)+pi).approx(100)
5.377660631089582934871817052010779119637787758986631545245841837
```

SEEALSO:

`libgiac`, `giacsettings`, `Pygen`,``loadgiacgen``

GETTING HELP:

- To obtain some help on a giac keyword use the help() method. In sage the htmlhelp() met

```
sage: libgiac.gcd?
```

```
"Returns the greatest common divisor of 2 polynomials of several
(Intg or Poly),(Intg or Poly)
gcd(45,75);gcd(15/7,50/9);gcd(x^2-2*x+1,x^3-1);gcd(t^2-2*t+1,t^2+
lcm,euler,modgcd,ezgcd,psrgcd,heugcd,Gcd"
```

- You can find full html documentation about the **giac** functions at:
    - http://www-fourier.ujf-grenoble.fr/~parisse/giac/doc/en/cascmd_en/
    - http://www-fourier.ujf-grenoble.fr/~parisse/giac/doc/fr/cascmd_fr/
    - http://www-fourier.ujf-grenoble.fr/~parisse/giac/doc/el/cascmd_el/
    - or in $SAGE_LOCAL/share/giac/doc/en/cascmd_en/index.html

REMARK:

- Graphics 2D Output via qcas (the qt frontend to giac) is removed in the sage version of gi

```
from giacpy_sage import libgiac
```

```
libgiac?
```

File: /home/fred-dev/sage/develop/sage.develop/local/lib/python2.7/site-packages/giacpy_sage.

Type: <type ‘instance’>

Definition: libgiac(s)

Docstring:

This function evaluate a python/sage object with the giac library. It creates in python/sage

- **First Example**:

```
sage: from giacpy_sage import libgiac
sage: x,y=libgiac('x,y')
sage: (x+2*y).cos().texpand()
cos(x)*(2*cos(y)^2-1)-sin(x)*2*cos(y)*sin(y)
```

- **Coercion, Pygen and internal giac variables**:

The most usefull objects will be the Python object of type Pygen.

```
sage: from giacpy_sage import *
sage: x,y,z=libgiac('x,y,z')
sage: f=sum([x[i] for i in range(5)])^15/(y+z);f.coeff(x[0],
(455*(x[1])^3+1365*(x[1])^2*x[2]+1365*(x[1])^2*x[3]+1365*(x[
```

Warning: The complex number sqrt(-1) is exported in python as I. (But it may appe

```
sage: libgiac((1+I*sqrt(3))^3).normal(); libgiac(1+I)
-8
1+i
```

Python integers and reals can be directly converted to giac.

```
sage: from giacpy_sage import *
sage: a=libgiac(2^1024);a.nextprime();(libgiac(1.234567)).er
1797693134862315907729305190789024733617976978942306572734З0
0.9191788641
```

The Python object y defined above is of type Pygen. It is not an internal giac variat

```
sage: from giacpy_sage import *
sage: libgiac('y:=1');y
1
y
sage: libgiac.purge('y')
1
sage: libgiac('y')
y
```

There are some natural coercion to Pygen elements:

```
sage: from giacpy_sage import *
sage: libgiac(pi)>3.14 ; libgiac(pi) >3.15 ; libgiac(3)==3
True
False
True
```

- **Lists of Pygen and Giac lists**:

Here l1 is a giac list and l2 is a python list of Pygen type objects.

```
sage: from giacpy_sage import *
sage: l1=libgiac(range(10)); l2=[1/(i^2+1) for i in l1]
sage: sum(l2)
33054527/16762850
```

So l1+l1 is done in giac and means a vector addition. But l2+l2 is done in Python s

```
sage: from giacpy_sage import *
sage: l1+l1
[0,2,4,6,8,10,12,14,16,18]
sage: l2+l2
[1, 1/2, 1/5, 1/10, 1/17, 1/26, 1/37, 1/50, 1/65, 1/82, 1, 1
```

Here V is not a Pygen element. We need to push it to giac to use a giac method lik

```
sage: from giacpy_sage import *
sage: V=[ [x[i]^j for i in range(8)] for j in range(8)]
sage: libgiac(V).dim()
[8,8]
sage: libgiac.det_minor(V).factor()
(x[6]-(x[7]))*(x[5]-(x[7]))*(x[5]-(x[6]))*(x[4]-(x[7]))*(x[4
```

- **Modular objects with %**

```
sage: from giacpy_sage import *
sage: V=libgiac.ranm(5,6) % 2;
sage: V.ker().rowdim()+V.rank()
6
sage: a=libgiac(7)%3;a;a%0;7%3
1 % 3
1
1
```

Do not confuse with the python integers:

```
sage: type(7%3)==type(a);type(a)==type(7%3)
False
False
```

- **Syntaxes with reserved or unknown Python/sage symbols**:

  In general equations needs symbols such as = < > that have another meaning in P

```
sage: from giacpy_sage import *
sage: x=libgiac('x')
sage: (1+2*sin(3*x)).solve(x).simplify()
Warning, argument is not an equation, solving 1+2*sin(3*x)=0
list[-pi/18,7*pi/18]

sage: libgiac.solve('sin(3*x)>2*sin(x)',x)
...
RuntimeError: Unable to find numeric values solving equation
```

  You can also add some hypothesis to a giac symbol:

```
sage: libgiac.assume('x>-pi && x<pi')
x
sage: libgiac.solve('sin(3*x)>2*sin(x)',x)
list[((x>(-5*pi/6)) and (x<(-pi/6))),((x>0) and (x<(pi/6))),
```

  To remove those hypothesis use the giac function: purge

```
sage: libgiac.purge('x')
assume[[],[line[-pi,pi]],[-pi,pi]]
sage: libgiac.solve('x>0')
list[x>0]
```

  Same problems with the ..

```
sage: from giacpy_sage import *
sage: x=libgiac('x')
sage: f=1/(5+cos(4*x))
sage: oldrep=1/2/(2*sqrt(6))*(atan(2*tan(4*x/2)/sqrt(6))+pi*
sage: newrep=1/2/(2*sqrt(6))*(atan((-sqrt(6)*sin(4*x)+2*sin(
sage: ((f.int(x)-newrep)*(f.int(x)-oldrep)).normal()
0
sage: libgiac.fMax(f,'x=-0..pi').simplify()
pi/4,3*pi/4
sage: libgiac.fMax.help()
"Returns the abscissa of the maximum of the expression.
Expr,[Var]
fMax(-x^2+2*x+1,x)
fMin"
sage: libgiac.sum(1/(1+x^2),'x=0..infinity').simplify()
(pi*exp(pi)^2+pi+exp(pi)^2-1)/(2*exp(pi)^2-2)
```

- **From giac to sage**:

  One can convert a Pygen element to sage with the sage method. Get more details

```
sage: Pygen.sage?
```

```
sage: from giacpy_sage import *
sage: L=libgiac('[1,sqrt(5),[1.3,x]]')
sage: L.sage()       # All entries are converted recursively
[1, sqrt(5), [1.3, x]]
```

  To obtain matrices and vectors, use the `matrix` and `vector` commands. Get mor

```
sage: Pygen._matrix_?
```

```
sage: Pygen._vector_?
```

```
sage: n=var('n');A=matrix([[1,2],[-1,1]])
sage: B=libgiac(A).matpow(n)    # We compute the symbolic pc
sage: C=matrix(SR,B); C         # We convert B to sage
[                    1/2*(I*sqrt(2) + 1)^n + 1/2*(-I*sqrt(2
[ 1/4*I*sqrt(2)*(I*sqrt(2) + 1)^n - 1/4*I*sqrt(2)*(-I*sqrt(2
sage: (C.subs(n=3)-A^3).expand()
[0 0]
[0 0]
```

MEMENTO of usual GIAC functions:

- *Expand with simplification*
  - `ratnormal`, `normal`, `simplify` (from the fastest to the most sophi
  - NB: `expand` function doesn't regroup nor cancel terms, so it could be
- *Factor/Regroup*
  - `factor`, `factors`, `regroup`, `cfactor`, `ifactor`
- *Misc*
  - `unapply`, `op`, `subst`
- *Polynomials/Fractions*
  - `coeff`, `gbasis`, `greduce`, `lcoeff`, `pcoeff`, `canonical_form`,
  - `proot`, `poly2symb`, `symb2poly`, `posubLMQ`, `poslbdLMQ`, `VAS`, `tc
  - `gcd`, `egcd`, `lcm`, `quo`, `rem`, `quorem`, `abcuv`, `chinrem`,
  - `peval`, `horner`, `lagrange`, `ptayl`, `spline`, `sturm`, `sturmab`
  - `partfrac`, `cpartfrac`
- *Memory/Variables*
  - `assume`, `about`, `purge`, `ans`
- *Calculus/Exact*
  - `linsolve`, `solve`, `csolve`, `desolve`, `seqsolve`, `reverse_rsol
  - `limit`, `series`, `sum`, `diff`, `fMax`, `fMin`,
  - `integrate`, `subst`, `ibpdv`, `ibpu`, `preval`
- *Calculus/Exp, Log, powers*
  - `exp2pow`, `hyp2exp`, `expexpand`, `lin`, `lncollect`, `lnexpand`, `po
- *Trigo*
  - `trigexpand`, `tsimplify`, `tlin`, `tcollect`,
  - `halftan`, `cos2sintan`, `sin2costan`, `tan2sincos`, `tan2cossin
  - `exp2trig`, `trig2exp`
  - `atrig2ln`, `acos2asin`, `acos2atan`, `asin2acos`, `asin2atan`, `at
- *Linear Algebra*
  - `identity`, `matrix`, `makemat`, `syst2mat`, `matpow`
  - `det`, `det_minor`, `rank`, `ker`, `image`, `rref`, `simplex_reduce`,
  - `egv`, `egvl`, `eigenvalues`, `pcar`, `pcar_hessenberg`, `pmin`,
  - `jordan`, `adjoint_matrix`, `companion`, `hessenberg`, `transpos
  - `cholesky`, `lll`, `lu`, `qr`, `svd`, `a2q`, `gauss`, `gramschmidt`, `q2a`, `is

- *Finite Fieds*
    - `%`, `% 0`, `mod`, `GF`, `powmod`
- *Integers*
    - `gcd`, `iabcuv`, `ichinrem`, `idivis`, `iegcd`,
    - `ifactor`, `ifactors`, `iquo`, `iquorem`, `irem`,
    - `is_prime, is_pseudoprime`, `lcm`, `mod`, `nextprime`, `pa2b2`, `pr`
- *List*
    - `append`, `accumulate_head_tail`, `concat`, `head`, `makelist`, `me`
- *Set*
    - `intersect`, `minus`, `union`, `is_element`, `is_included`