giacpy_sagedoc

import giacpy_sage

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

   - 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 computatic

Here A is a Pygen element, and it is ready for any giac function.
In general, you may prefer to directly create a Pygen and execute the evaluation in giac. This is exactly the mean

```python
time Bgiac=Igiac.gbasis([R.gens()],'revlex')
Running a probabilistic check for the reconstructed Groebner basis
Time: CPU 0.46 s, Wall: 0.50 s
```

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

```python
time Bgiac=Igiac.gbasis([R.gens()],'revlex')
Time: CPU 2.74 s, Wall: 2.75 s
```

SEE ALSO:
- `libgiac`, `giacsettings`, `Pygen`, `loadgiacgen`

GETTING HELP:
- To obtain some help on a giac keyword use the `help()` method. In sage the `htmlhelp()` method for Pygen element

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

```bash
2 sur 7  29/08/2017 13:07
```
"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+1)
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/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 giac.

```python
from giacpy_sage import libgiac
libgiac?
```

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

  ```python
  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.

  ```python
  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[1])^2*x[4])
  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.

```python
sage: from giacpy_sage import *
 sage: a=libgiac(2^1024);a.nextprime();(libgiac(1.234567)).erf
 sage: 1797693134862315907729305190789024733617976978942306572734360.9191788641
```
The Python object \texttt{y} defined above is of type \texttt{Pygen}. It is not an internal \texttt{giac} variable. (Most of the time yo
\begin{verbatim}
sage: from giacpy_sage import *
sage: libgiac('y:=1'); y
1
Y
sage: libgiac.purge('y')
1
sage: libgiac('y')
y
\end{verbatim}

There are some natural coercion to \texttt{Pygen} elements:
\begin{verbatim}
sage: from giacpy_sage import *
sage: libgiac(pi)>3.14 ; libgiac(pi) >3.15 ; libgiac(3)==3
True
False
True
\end{verbatim}

- \textbf{Lists of Pygen and Giac lists:}

Here \texttt{l1} is a giac list and \texttt{l2} is a python list of Pygen type objects.
\begin{verbatim}
sage: from giacpy_sage import *
sage: l1=libgiac(range(10)); l2=[1/(i^2+1) for i in l1]
sage: sum(l2)
33054527/16762850
\end{verbatim}

So \texttt{l1+l1} is done in giac and means a vector addition. But \texttt{l2+l2} is done in Python so

\begin{verbatim}
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
\end{verbatim}

Here \texttt{V} is not a Pygen element. We need to push it to giac to use a giac method like 
\begin{verbatim}
sage: from giacpy_sage import *
sage: V=[ [x[i] 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[4]-(x[7]))*(x[4]-(x[7]))*(x[4]-
\end{verbatim}

- \textbf{Modular objects with \%}

\begin{verbatim}
sage: from giacpy_sage import *
sage: V=libgiac.ranm(5,6) % 2;
\end{verbatim}

\begin{verbatim}
sage: a=libgiac(7)%3;a;a%0;7%3
1 % 3
1
\end{verbatim}

Do not confuse with the python integers:
\begin{verbatim}
sage: type(7%3)==type(a);type(a)==type(7%3)
False
False
\end{verbatim}
### Syntaxes with reserved or unknown Python/sage symbols:

In general, equations need symbols such as \( = \), \( < \), or \( > \) that have another meaning in Python or Sage. So:

```python
from giacpy_sage import *
sage: x=libgiac('x')
sage: (1+2*sin(3*x)).solve(x).simplify()
```

```python
Warning, argument is not an equation, solving 1+2*sin(3*x)=0
```

```python
list[-pi/18,7*pi/18]
sage: libgiac.solve('sin(3*x)>2*sin(x)',x)
```

```python
... RuntimeError: Unable to find numeric values solving equation.
```

You can also add some hypothesis to a giac symbol:

```python
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:

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

### From giac to sage:

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

```python
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 more details with:

```python
sage: from giacpy_sage import *
sage: L=libgiac('[1,sqrt(5),[1.3,x]]')
sage: L._matrix_()
```

```python
(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 power on A via libgic
sage: C=matrix(SR,B); C  # We convert B to sage

MEMENTO of usual GIAC functions:

- Expand with simplification
  - ratnormal, normal, simplify (from the fastest to the most sophisticated)
  - NB: expand function doesn't regroup nor cancel terms, so it could be slow.

- Factor/Regroup
  - factor, factors, regroup, cfactor, ifactor

- Misc
  - unapply, op, subst

- Polynomials/Fractions
  - coeff, gbasis, gred, lcoeff, pcoeff, canonical_form, proot, poly2symb, symb2poly, posubLMQ, posldLMQ, VAS, tc
  - gcd, egcd, lcm, quo, rem, quorem, abcuv, chinrem
  - peval, horner, lagrange, ptyal, spline, stall, sturmb, 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, cos2sin, sin2cos, tan2sin, tan2cos
  - exp2trig, trig2exp
  - atrig2ln, acos2asin, acos2atan, asin2acos, asin2atan, at

- Linear Algebra
  - identity, matrix, makemat, syst2mat, matpow
  - det, det_minor, rank, ker, image, rref, simplex_reduce,
  - eqv, egvl, eigenvalues, pcar, pcar_hessenberg, pmin,
  - jordan, adjoint_matrix, companion, hessenberg, transpos
  - cholesky, lll, lu, qr, svd, a2q, gauss, gramschmidt, q2a, is,
- **Finite Fields**
  - \%, \%, 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