Sage Reference Manual: Interpreter Interfaces

Release 7.5

The Sage Development Team

CONTENTS

1	Common Interface Functionality			
2	Common Interface Functionality through Pexpect			
3	Sage wrapper around pexpect's spawn class and			
4	Interface to Axiom			
5	The Elliptic Curve Factorization Method			
6	5 Interface to 4ti2			
7	Interface to FriCAS			
8	Interface to Frobby for fast computations on monomial ideals.	45		
9	Interface to GAP 9.1 First Examples	49 49 50 50 51		
10	Interface to GAP3 10.1 Obtaining GAP3 10.2 Changing which GAP3 is used 10.3 Functionality and Examples 10.4 Common Pitfalls 10.5 Examples	61 61 62 63 64		
11	Interface to Groebner Fan	71		
12	Pexpect Interface to Giac 12.1 Tutorial	73 74		
13	Interface to the Gnuplot interpreter	81		
14	4 Interface to the GP calculator of PARI/GP			
15	5 Interface for extracting data and generating images from Jmol readable files.			
16	Interface to KASH 16.1 Issues	95 95		

	16.2 Tutorial	
17	Interface to LiE 17.1 Tutorial	103 103
18	Lisp Interface	111
19	Interface to Macaulay2	115
	Interface to Magma 20.1 Parameters	125 126 126
22	Interface to Maple	147
	22.1 Tutorial	147
23	Interface to Mathematica 23.1 Tutorial	158 159
24	Interface to MATLAB 24.1 Tutorial	163
25	Pexpect interface to Maxima 25.1 Tutorial 25.2 Examples involving matrices 25.3 Laplace Transforms 25.4 Continued Fractions 25.5 Special examples 25.6 Miscellaneous 25.7 Interactivity 25.8 Latex Output 25.9 Long Input	170 170 171 171 172 172 173
26	Abstract interface to Maxima	177
27	Library interface to Maxima	195
28	Interface to MuPAD	211
29	Interface to mwrank	215
30	Interface to GNU Octave 30.1 Computation of Special Functions	219 219 220
31	Interface to PHC.	225
32	POV-Ray, The Persistence of Vision Ray Tracer	233

33	Parallel Interface to the Sage interpreter	235
34	Interface to QEPCAD	237
35	Interface to Bill Hart's Quadratic Sieve	265
36	Interfaces to R	269
37	7 Interface to several Rubik's cube solvers.	
38	Interface to Sage	285
39	Interface to Scilab	289
40	Interface to Singular 40.1 Introduction	295 298 298
41	The Tachyon Ray Tracer	317
42	Interface to TIDES	321
43	Interface to the Sage cleaner	329
44	Quitting interfaces	331
45	An interface to read data files	333
46	Indices and Tables	335

Sage provides a unified interface to the best computational software. This is accomplished using both C-libraries (see C/C++ Library Interfaces) and interpreter interfaces, which are implemented using pseudo-tty's, system files, etc. This chapter is about these interpreter interfaces.

Note: Each interface requires that the corresponding software is installed on your computer. Sage includes GAP, PARI, Singular, and Maxima, but does not include Octave (very easy to install), MAGMA (non-free), Maple (non-free), or Mathematica (non-free).

There is overhead associated with each call to one of these systems. For example, computing 2+2 thousands of times using the GAP interface will be slower than doing it directly in Sage. In contrast, the C-library interfaces of C/C++ Library Interfaces incur less overhead.

In addition to the commands described for each of the interfaces below, you can also type e.g., %gap, %magma, etc., to directly interact with a given interface in its state. Alternatively, if X is an interface object, typing X.interact() allows you to interact with it. This is completely different than X.console() which starts a complete new copy of whatever program X interacts with. Note that the input for X.interact() is handled by Sage, so the history buffer is the same as for Sage, tab completion is as for Sage (unfortunately!), and input that spans multiple lines must be indicated using a backslash at the end of each line. You can pull data into an interactive session with X using sage (expression).

The console and interact methods of an interface do very different things. For example, using gap as an example:

- 1. gap.console(): You are completely using another program, e.g., gap/magma/gp Here Sage is serving as nothing more than a convenient program launcher, similar to bash.
- 2. gap.interact(): This is a convenient way to interact with a running gap instance that may be "full of" Sage objects. You can import Sage objects into this gap (even from the interactive interface), etc.

The console function is very useful on occasion, since you get the exact actual program available (especially useful for tab completion and testing to make sure nothing funny is going on).

CONTENTS 1

2 CONTENTS

COMMON INTERFACE FUNCTIONALITY

See the examples in the other sections for how to use specific interfaces. The interface classes all derive from the generic interface that is described in this section.

AUTHORS:

- William Stein (2005): initial version
- William Stein (2006-03-01): got rid of infinite loop on startup if client system missing
- Felix Lawrence (2009-08-21): edited ._sage_() to support lists and float exponents in foreign notation.
- Simon King (2010-09-25): Expect._local_tmpfile() depends on Expect.pid() and is cached; Expect.quit() clears that cache, which is important for forking.
- Jean-Pierre Flori (2010,2011): Split non Pexpect stuff into a parent class.
- Simon King (2015): Improve pickling for InterfaceElement

```
class sage.interfaces.interface. AsciiArtString
    Bases: str
class sage.interfaces.interface. Interface ( name)
    Bases: sage.structure.parent_base.ParentWithBase
    Interface interface object.
    call (function_name, *args, **kwds)
    clear (var)
         Clear the variable named var.
    console ()
    cputime ()
         CPU time since this process started running.
    eval ( code, **kwds)
    execute (*args, **kwds)
    function_call (function, args=None, kwds=None)
         EXAMPLES:
         sage: maxima.quad_qags(x, x, 0, 1, epsrel=1e-4)
         [0.5, 0.55511151231257...e-14,21,0]
         sage: maxima.function_call('quad_qags', [x, x, 0, 1], {'epsrel':'1e-4'})
         [0.5, 0.55511151231257...e-14,21,0]
```

get (var)

Get the value of the variable var.

get_seed ()

Returns the seed used to set the random number generator in this interface. The seed is initialized as None but should be set when the interface starts.

EXAMPLES:

```
sage: s = Singular()
sage: s.set_seed(107)
107
sage: s.get_seed()
107
```

get_using_file (var)

Return the string representation of the variable var in self, possibly using a file. Use this if var has a huge string representation, since it may be way faster.

Warning: In fact unless a special derived class implements this, it will *not* be any faster. This is the case for this class if you're reading it through introspection and seeing this.

help(s)

interact ()

This allows you to interactively interact with the child interpreter. Press Ctrl-D or type 'quit' or 'exit' to exit and return to Sage.

Note: This is completely different than the console() member function. The console function opens a new copy of the child interpreter, whereas the interact function gives you interactive access to the interpreter that is being used by Sage. Use sage(xxx) or interpretername(xxx) to pull objects in from sage to the interpreter.

```
name ( new_name=None)
new ( code)
rand seed ( )
```

Returns a random seed that can be put into set_seed function for any interpreter. This should be overridden if the particular interface needs something other than a small positive integer.

EXAMPLES:

```
from sage.misc.random_testing import random_testing
sage: from sage.interfaces.interface import Interface
sage: i = Interface("")
sage: i.rand_seed() # random
318491487L

sage: s = Singular()
sage: s.rand_seed() # random
365260051L
```

read (filename)

```
sage: filename = tmp_filename()
sage: f = open(filename, 'w')
sage: f.write('x = 2\n')
```

```
sage: f.close()
sage: octave.read(filename) # optional - octave
sage: octave.get('x') # optional - octave
' 2'
sage: import os
sage: os.unlink(filename)
```

set (var, value)

Set the variable var to the given value.

set seed (seed=None)

Sets the random seed for the interpreter and returns the new value of the seed. This is dependent on which interpreter so must be implemented in each separately. For examples see gap.py or singular.py. If seed is None then should generate a random seed.

EXAMPLES:

```
sage: s = Singular()
sage: s.set_seed(1)
1
sage: [s.random(1,10) for i in range(5)]
[8, 10, 4, 9, 1]
sage: from sage.interfaces.interface import Interface
sage: i = Interface("")
sage: i.set_seed()
Traceback (most recent call last):
...
NotImplementedError: This interpreter did not implement a set_seed function
```

Bases: sage.structure.element.Element

Interface element.

attribute (attrname)

If this wraps the object x in the system, this returns the object x attrname. This is useful for some systems that have object oriented attribute access notation.

EXAMPLES:

```
sage: g = gap('SO(1,4,7)')
sage: k = g.InvariantQuadraticForm()
sage: k.attribute('matrix')
[ [ 0*Z(7), Z(7)^0, 0*Z(7), 0*Z(7) ], [ 0*Z(7), 0*Z(7), 0*Z(7), 0*Z(7)],
      [ 0*Z(7), 0*Z(7), Z(7), 0*Z(7)], [ 0*Z(7), 0*Z(7), 0*Z(7)]
```

```
sage: e = gp('ellinit([0,-1,1,-10,-20])')
sage: e.attribute('j')
-122023936/161051
```

```
bool ( )
gen ( n)
```

get_using_file ()

Return this element's string representation using a file. Use this if self has a huge string representation. It'll be way faster.

EXAMPLES:

```
sage: a = maxima(str(2^1000))
sage: a.get_using_file()

→'10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983

→'
```

hasattr (attrname)

Returns whether the given attribute is already defined by this object, and in particular is not dynamically generated.

EXAMPLES:

```
sage: m = maxima('2')
sage: m.hasattr('integral')
True
sage: m.hasattr('gcd')
False
```

is_string()

Tell whether this element is a string.

By default, the answer is negative.

```
name ( new name=None)
```

Returns the name of self. If new_name is passed in, then this function returns a new object identical to self whose name is new_name.

Note that this can overwrite existing variables in the system.

EXAMPLES:

```
sage: x = r([1,2,3]); x
[1] 1 2 3
sage: x.name()
'sage3'
sage: x = r([1,2,3]).name('x'); x
[1] 1 2 3
sage: x.name()
'x'
```

```
sage: s5 = gap.SymmetricGroup(5).name('s5')
sage: s5
SymmetricGroup([1..5])
sage: s5.name()
's5'
```

```
sage ( *args, **kwds)
```

Attempt to return a Sage version of this object.

This method does nothing more than calling _sage_(), simply forwarding any additional arguments.

```
sage: gp(1/2).sage()
1/2
sage: _.parent()
Rational Field
sage: singular.lib("matrix")
```

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: singular.matrix(2,2).sage()
[0 0]
[0 0]
[class sage.interfaces.interface. InterfaceFunction (parent, name)
Bases: sage.structure.sage_object.SageObject
Interface function.

class sage.interfaces.interface. InterfaceFunctionElement (obj, name)
Bases: sage.structure.sage_object.SageObject
Interface function element.
help()
sage.interfaces.interface.is_InterfaceElement(x)
```

COMMON INTERFACE FUNCTIONALITY THROUGH PEXPECT

See the examples in the other sections for how to use specific interfaces. The interface classes all derive from the generic interface that is described in this section.

AUTHORS:

- William Stein (2005): initial version
- William Stein (2006-03-01): got rid of infinite loop on startup if client system missing
- Felix Lawrence (2009-08-21): edited ._sage_() to support lists and float exponents in foreign notation.
- Simon King (2010-09-25): Expect._local_tmpfile() depends on Expect.pid() and is cached; Expect.quit() clears that cache, which is important for forking.
- Jean-Pierre Flori (2010,2011): Split non Pexpect stuff into a parent class.
- Simon King (2010-11-23): Ensure that the interface is started again after a crash, when a command is executed in _eval_line. Allow synchronisation of the GAP interface.
- François Bissey, Bill Page, Jeroen Demeyer (2015-12-09): Upgrade to pexpect 4.0.1 + patches, see trac ticket #10295.

Bases: sage.interfaces.interface.Interface

Expect interface object.

```
clear_prompts ()
```

command ()

Returns the command used in this interface.

EXAMPLES:

```
sage: magma.set_server_and_command(command = 'magma-2.19')
sage: magma.command() # indirect doctest
'magma-2.19'
```

detach ()

Forget the running subprocess: keep it running but pretend that it's no longer running.

```
sage: a = maxima('v')
sage: saved_expect = maxima._expect # Save this to close later
sage: maxima.detach()
sage: a._check_valid()
Traceback (most recent call last):
ValueError: The maxima session in which this object was defined is no longer.
⇔running.
sage: saved_expect.close() # Close child process
```

Calling detach () a second time does nothing:

```
sage: maxima.detach()
```

```
eval (code, strip=True, synchronize=False, locals=None, allow_use_file=True, split_lines='nofile',
        **kwds)
     INPUT:
```

- •code text to evaluate
- •strip bool; whether to strip output prompts, etc. (ignored in the base class).
- •locals None (ignored); this is used for compatibility with the Sage notebook's generic system interface.
- •allow_use_file bool (default: True); if True and code exceeds an interface-specific threshold then code will be communicated via a temporary file rather that the character-based interface. If False then the code will be communicated via the character interface.
- •split lines Tri-state (default: "nofile"); if "nofile" then code is sent line by line unless it gets communicated via a temporary file. If True then code is sent line by line, but some lines individually might be sent via temporary file. Depending on the interface, this may transform grammatical code into ungrammatical input. If False, then the whole block of code is evaluated
- •**kwds All other arguments are passed onto the _eval_line method. An often useful example is reformat=False.

```
expect ()
interrupt (tries=5, timeout=2.0, quit_on_fail=True)
is_local()
is remote ()
is running ()
    Return True if self is currently running.
path ()
pid()
    Return the PID of the underlying sub-process.
```

REMARK:

If the interface terminates unexpectedly, the original PID will still be used. But if it was terminated using quit (), a new sub-process with a new PID is automatically started.

```
sage: pid = gap.pid()
sage: gap.eval('quit;')
''
sage: pid == gap.pid()
True
sage: gap.quit()
sage: pid == gap.pid()
False
```

quit (verbose=False, timeout=None)

Quit the running subprocess.

INPUT:

•verbose - (boolean, default False) print a message when quitting this process?

EXAMPLES:

Calling quit () a second time does nothing:

```
sage: maxima.quit(verbose=True)
```

server ()

Returns the server used in this interface.

EXAMPLES:

set_server_and_command (server=None, command=None, server_tmpdir=None, ulimit=None) Changes the server and the command to use for this interface. This raises a Runtime error if the interface is already started.

EXAMPLES:

user_dir()

```
class sage.interfaces.expect. ExpectElement (parent, value, is_name=False, name=None)
    Bases: sage.interfaces.interface.InterfaceElement
    Expect element.

class sage.interfaces.expect. ExpectFunction (parent, name)
    Bases: sage.interfaces.interface.InterfaceFunction
    Expect function.

class sage.interfaces.expect. FunctionElement (obj, name)
    Bases: sage.interfaces.interface.InterfaceFunctionElement
    Expect function element.

class sage.interfaces.expect. StdOutContext (interface, silent=False, stdout=None)
    A context in which all communation between Sage and a subprocess interfaced via pexpect is printed to stdout.
sage.interfaces.expect. console (cmd)

class sage.interfaces.expect. gc_disabled
    Bases: object

This is a "with" statement context manager. Garbage collection is disabled within its scope. Nested usage is
```

EXAMPLES:

properly handled.

```
sage: import gc
sage: from sage.interfaces.expect import qc_disabled
sage: gc.isenabled()
True
sage: with gc_disabled():
      print(gc.isenabled())
. . . . :
         with gc_disabled():
      print(ge.100.
print(gc.isenabled())
          print(gc.isenabled())
. . . . :
. . . . :
False
False
False
sage: gc.isenabled()
True
```

sage.interfaces.expect. is_ExpectElement (x)

SAGE WRAPPER AROUND PEXPECT'S SPAWN CLASS AND

the ptyprocess's PtyProcess class.

AUTHOR:

- Jeroen Demeyer (2015-02-01): initial version, see trac ticket #17686.
- Jeroen Demeyer (2015-12-04): add support for pexpect 4 + ptyprocess, see trac ticket #10295.

Quit the child process: send the quit string, close the pseudo-tty and kill the process.

This function returns immediately, it doesn't wait for the child process to die.

EXAMPLES:

```
sage: from sage.interfaces.sagespawn import SageSpawn
sage: s = SageSpawn("sleep 1000")
sage: s.close()
sage: while s.isalive(): # long time (5 seconds)
....: sleep(0.1)
```

terminate_async (interval=5.0)

Terminate the child process group asynchronously.

This function returns immediately, while the child is slowly being killed in the background.

INPUT:

•interval – (default: 5) how much seconds to wait between sending two signals.

EXAMPLES:

Run an infinite loop in the shell:

```
sage: from sage.interfaces.sagespawn import SageSpawn
sage: s = SageSpawn("sh", ["-c", "while true; do sleep 1; done"])
```

Check that the process eventually dies after calling terminate async:

```
...: else:
...: break # process got killed
```

class sage.interfaces.sagespawn. SageSpawn (*args, **kwds)

Bases: pexpect.pty_spawn.spawn

Spawn a subprocess in a pseudo-tty.

- •*args, **kwds: see pexpect.spawn.
- •name human-readable name for this process, used for display purposes only.
- •quit_string (default: None) if not None , send this string to the child process before killing it.

EXAMPLES:

```
sage: from sage.interfaces.sagespawn import SageSpawn
sage: SageSpawn("sleep 1", name="Sleeping Beauty")
Sleeping Beauty with PID ... running ...
```

```
expect_peek (*args, **kwds)
```

Like expect () but restore the read buffer such that it looks like nothing was actually read. The next reading will continue at the current position.

EXAMPLES:

```
sage: from sage.interfaces.sagespawn import SageSpawn
sage: E = SageSpawn("sh", ["-c", "echo hello world"])
sage: _ = E.expect_peek("w")
sage: E.read()
'hello world\r\n'
```

expect_upto (*args, **kwds)

Like expect () but restore the read buffer starting from the matched string. The next reading will continue starting with the matched string.

```
sage: from sage.interfaces.sagespawn import SageSpawn
sage: E = SageSpawn("sh", ["-c", "echo hello world"])
sage: _ = E.expect_upto("w")
sage: E.read()
'world\r\n'
```

CHAPTER

FOUR

INTERFACE TO AXIOM

TODO:

- Evaluation using a file is not done. Any input line with more than a few thousand characters would hang the system, so currently it automatically raises an exception.
- All completions of a given command.
- Interactive help.

Axiom is a free GPL-compatible (modified BSD license) general purpose computer algebra system whose development started in 1973 at IBM. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Axiom has implementations of many functions relating to the invariant theory of the symmetric group S_n . For many links to Axiom documentation see http://wiki.axiom-developer.org.

AUTHORS:

• Bill Page (2006-10): Created this (based on Maxima interface)

Note: Bill Page put a huge amount of effort into the Sage Axiom interface over several days during the Sage Days 2 coding sprint. This is contribution is greatly appreciated.

- William Stein (2006-10): misc touchup.
- Bill Page (2007-08): Minor modifications to support axiom4sage-0.3

Note: The axiom4sage-0.3.spkg is based on an experimental version of the FriCAS fork of the Axiom project by Waldek Hebisch that uses pre-compiled cached Lisp code to build Axiom very quickly with clisp.

If the string "error" (case insensitive) occurs in the output of anything from axiom, a RuntimeError exception is raised.

EXAMPLES: We evaluate a very simple expression in axiom.

The type of a is AxiomElement, i.e., an element of the axiom interpreter.

```
sage: type(a) #optional - axiom
<class 'sage.interfaces.axiom.AxiomElement'>
sage: parent(a) #optional - axiom
Axiom
```

The underlying Axiom type of a is also available, via the type method:

```
sage: a.type() #optional - axiom
PositiveInteger
```

We factor $x^5 - y^5$ in Axiom in several different ways. The first way yields a Axiom object.

```
sage: F = axiom.factor('x^5 - y^5'); F #optional - axiom
4     3     2     2     3     4
- (y - x)(y + x y + x y + x y + x )
sage: type(F) #optional - axiom
<class 'sage.interfaces.axiom.AxiomElement'>
sage: F.type() #optional - axiom
Factored Polynomial Integer
```

Note that Axiom objects are normally displayed using "ASCII art".

```
sage: a = axiom(2/3); a  #optional - axiom
2
-
3
sage: a = axiom('x^2 + 3/7'); a  #optional - axiom
2  3
x + -
7
```

The axiom.eval command evaluates an expression in axiom and returns the result as a string. This is exact as if we typed in the given line of code to axiom; the return value is what Axiom would print out.

We can create the polynomial f as a Axiom polynomial, then call the factor method on it. Notice that the notation f.factor() is consistent with how the rest of Sage works.

```
sage: f = axiom('x^5 - y^5')  #optional - axiom
sage: f^2  #optional - axiom
10    5    5    10
y    - 2x y + x
sage: f.factor()  #optional - axiom
4    3    2    2    3    4
- (y - x)(y + x y + x y + x y + x )
```

Control-C interruption works well with the axiom interface, because of the excellent implementation of axiom. For example, try the following sum but with a much bigger range, and hit control-C.

```
sage: axiom('1/100 + 1/101') #optional - axiom
201
----
10100
sage: a = axiom('(1 + sqrt(2))^5'); a #optional - axiom
```

```
+-+
29\|2 + 41
```

TESTS:

We check to make sure the subst method works with keyword arguments.

```
sage: a = axiom(x+2); a #optional - axiom
x + 2
sage: a.subst(x=3) #optional - axiom
5
```

We verify that Axiom floating point numbers can be converted to Python floats.

```
sage: float(axiom(2)) #optional - axiom
2.0
```

Bases: sage.interfaces.axiom.PanAxiom

Create an instance of the Axiom interpreter.

TESTS:

```
sage: axiom == loads(dumps(axiom))
True
```

console ()

Spawn a new Axiom command-line session.

EXAMPLES:

```
sage: axiom.console() #not tested

AXIOM Computer Algebra System

Version: Axiom (January 2009)

Timestamp: Sunday January 25, 2009 at 07:08:54

Issue )copyright to view copyright notices.

Issue )summary for a summary of useful system commands.

Issue )quit to leave AXIOM and return to shell.
```

```
class sage.interfaces.axiom. AxiomElement (parent, value, is_name=False, name=None)
```

Bases: sage.interfaces.axiom.PanAxiomElement

```
class sage.interfaces.axiom. AxiomExpectFunction ( parent, name)
```

Bases: sage.interfaces.axiom.PanAxiomExpectFunction

TESTS:

```
sage: axiom.upperCase_q
upperCase?
sage: axiom.upperCase_e
upperCase!
```

```
class sage.interfaces.axiom. AxiomFunctionElement (object, name)
```

Bases: sage.interfaces.axiom.PanAxiomFunctionElement

TESTS:

```
sage: a = axiom('"Hello"') #optional - axiom
sage: a.upperCase_q #optional - axiom
upperCase?
sage: a.upperCase_e #optional - axiom
upperCase!
sage: a.upperCase_e() #optional - axiom
"HELLO"
```

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect

Interface to a PanAxiom interpreter.

get (var)

Get the string value of the Axiom variable var.

EXAMPLES:

```
sage: axiom.set('xx', '2')  #optional - axiom
sage: axiom.get('xx')  #optional - axiom
'2'
sage: a = axiom('(1 + sqrt(2))^5')  #optional - axiom
sage: axiom.get(a.name())  #optional - axiom
' +-+\r\n 29\\|2 + 41'
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: axiom.set('xx', '2') #optional - axiom
sage: axiom.get('xx') #optional - axiom
'2'
```

class sage.interfaces.axiom. PanAxiomElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.expect.ExpectElement

as_type (type)

Returns self as type.

EXAMPLES:

```
sage: a = axiom(1.2); a  #optional - axiom
1.2
sage: a.as_type(axiom.DoubleFloat) #optional - axiom
1.2
sage: _.type()  #optional - axiom
DoubleFloat
```

comma (*args)

Returns a Axiom tuple from self and args.

```
sage: two = axiom(2) #optional - axiom
sage: two.comma(3) #optional - axiom
[2,3]
sage: two.comma(3,4) #optional - axiom
[2,3,4]
sage: _.type() #optional - axiom
Tuple PositiveInteger
```

type ()

Returns the type of an AxiomElement.

EXAMPLES:

```
sage: axiom(x+2).type() #optional - axiom
Polynomial Integer
```

unparsed_input_form ()

Get the linear string representation of this object, if possible (often it isn't).

EXAMPLES:

```
sage: a = axiom(x^2+1); a #optional - axiom
2
x + 1
sage: a.unparsed_input_form() #optional - axiom
'x*x+1'
```

class sage.interfaces.axiom. PanAxiomExpectFunction (parent, name)

Bases: sage.interfaces.expect.ExpectFunction

TESTS:

```
sage: axiom.upperCase_q
upperCase?
sage: axiom.upperCase_e
upperCase!
```

class sage.interfaces.axiom. PanAxiomFunctionElement (object, name)

Bases: sage.interfaces.expect.FunctionElement

TESTS:

```
sage: a = axiom('"Hello"') #optional - axiom
sage: a.upperCase_q #optional - axiom
upperCase?
sage: a.upperCase_e #optional - axiom
upperCase!
sage: a.upperCase_e() #optional - axiom
"HELLO"
```

sage.interfaces.axiom.axiom_console ()

Spawn a new Axiom command-line session.

```
Issue )copyright to view copyright notices.
Issue )summary for a summary of useful system commands.
Issue )quit to leave AXIOM and return to shell.
```

sage.interfaces.axiom. is_AxiomElement (x)

Returns True of x is of type AxiomElement.

EXAMPLES:

```
sage: from sage.interfaces.axiom import is_AxiomElement
sage: is_AxiomElement(axiom(2)) #optional - axiom
True
sage: is_AxiomElement(2)
False
```

sage.interfaces.axiom. reduce_load_Axiom ()

Returns the Axiom interface object defined in sage.interfaces.axiom.

```
sage: from sage.interfaces.axiom import reduce_load_Axiom
sage: reduce_load_Axiom()
Axiom
```

THE ELLIPTIC CURVE FACTORIZATION METHOD

The elliptic curve factorization method (ECM) is the fastest way to factor a **known composite** integer if one of the factors is relatively small (up to approximately 80 bits / 25 decimal digits). To factor an arbitrary integer it must be combined with a primality test. The *ECM.factor()* method is an example for how to combine ECM with a primality test to compute the prime factorization of integers.

Sage includes GMP-ECM, which is a highly optimized implementation of Lenstra's elliptic curve factorization method. See http://ecm.gforge.inria.fr for more about GMP-ECM.

AUTHORS:

These people wrote GMP-ECM: Pierrick Gaudry, Jim Fougeron, Laurent Fousse, Alexander Kruppa, Dave Newman, Paul Zimmermann

BUGS:

Output from ecm is non-deterministic. Doctests should set the random seed, but currently there is no facility to do so.

Create an interface to the GMP-ECM elliptic curve method factorization program.

See http://ecm.gforge.inria.fr

INPUT:

- •B1 integer. Stage 1 bound
- •B2 integer. Stage 2 bound (or interval B2min-B2max)

In addition the following keyword arguments can be used:

- •x0 integer x. use x as initial point
- •sigma integer s. Use s as curve generator [ecm]
- •A integer a. Use a as curve parameter [ecm]
- •k integer n. Perform >= n steps in stage 2
- •power integer n. Use x^n for Brent-Suyama's extension
- •dickson integer n. Use n-th Dickson's polynomial for Brent-Suyama's extension
- •c integer n. Perform n runs for each input
- •pm1 boolean. perform P-1 instead of ECM
- •pp1 boolean. perform P+1 instead of ECM
- •q boolean. quiet mode

- •v boolean. verbose mode
- •timestamp boolean. print a time stamp with each number
- •mpzmod boolean. use GMP's mpz_mod for mod reduction
- •modmuln boolean. use Montgomery's MODMULN for mod reduction
- •redc boolean. use Montgomery's REDC for mod reduction
- •nobase2 boolean. Disable special base-2 code
- •base2 integer n. Force base 2 mode with 2^n+1 (n>0) or 2^n-1 (n<0)
- •save string filename. Save residues at end of stage 1 to file
- •savea string filename. Like -save, appends to existing files
- •resume string filename. Resume residues from file, reads from stdin if file is "-"
- •primetest boolean. Perform a primality test on input
- •treefile string. Store product tree of F in files f.0 f.1 ...
- •i integer. increment B1 by this constant on each run
- •I integer f. auto-calculated increment for B1 multiplied by f scale factor.
- •inp string. Use file as input (instead of redirecting stdin)
- •b boolean. Use breadth-first mode of file processing
- •d boolean. Use depth-first mode of file processing (default)
- •one boolean. Stop processing a candidate if a factor is found (looping mode)
- •n boolean. Run ecm in 'nice' mode (below normal priority)
- •nn boolean. Run ecm in 'very nice' mode (idle priority)
- •t integer n. Trial divide candidates before P-1, P+1 or ECM up to n.
- •ve integer n. Verbosely show short (< n character) expressions on each loop
- •cofdec boolean. Force cofactor output in decimal (even if expressions are used)
- •B2scale integer. Multiplies the default B2 value
- •go integer. Preload with group order val, which can be a simple expression, or can use N as a placeholder for the number being factored.
- •prp string. use shell command cmd to do large primality tests
- •prplen integer, only candidates longer than this number of digits are 'large'
- •prpval integer. value>=0 which indicates the prp command foundnumber to be PRP.
- •prptmp file. outputs n value to temp file prior to running (NB. gets deleted)
- •prplog file. otherwise get PRP results from this file (NB. gets deleted)
- •prpyes string. Literal string found in prplog file when number is PRP
- •prpno string. Literal string found in prplog file when number is composite
- **factor** (n, factor_digits=None, B1=2000, proof=False, **kwds)

Return a probable prime factorization of n.

Combines GMP-ECM with a primality test, see $is_prime()$. The primality test is provable or probabilistic depending on the proof flag.

Moreover, for small n PARI is used directly.

Warning: There is no mathematical guarantee that the factors returned are actually prime if proof=False (default). It is extremely likely, though. Currently, there are no known examples where this fails.

INPUT:

- •n a positive integer
- •factor_digits integer or None (default). Optional guess at how many digits are in the smallest factor.
- •B1 initial lower bound, defaults to 2000 (15 digit factors). Used if factor_digits is not specified.
- •proof boolean (default: False). Whether to prove that the factors are prime.
- •kwds keyword arguments to pass to ecm-gmp. See help for *ECM* for more details.

OUTPUT:

A list of integers whose product is n.

Note: Trial division should typically be performed, but this is not implemented (yet) in this method.

If you suspect that n is the product of two similarly-sized primes, other methods (such as a quadratic sieve – use the qsieve command) will usually be faster.

The best known algorithm for factoring in the case where all factors are large is the general number field sieve. This is not implemented in Sage; You probably want to use a cluster for problems of this size.

EXAMPLES:

```
sage: ecm.factor(602400691612422154516282778947806249229526581)
[45949729863572179, 13109994191499930367061460439]
sage: ecm.factor((2^197 + 1)/3) # long time
[197002597249, 1348959352853811313, 251951573867253012259144010843]
sage: ecm.factor(179427217^13) == [179427217] * 13
True
```

find_factor (n, factor_digits=None, B1=2000, **kwds)

Return a factor of n.

See also factor() if you want a prime factorization of n.

INPUT:

- •n a positive integer,
- •factor_digits integer or None (default). Decimal digits estimate of the wanted factor.
- •B1 integer. Stage 1 bound (default 2000). This is used as bound if factor_digits is not specified.
- •kwds optional keyword parameters.

OUTPUT:

List of integers whose product is n. For certain lengths of the factor, this is the best algorithm to find a factor.

Note: ECM is not a good primality test. Not finding a factorization is only weak evidence for n being prime. You should run a **good** primality test before calling this function.

EXAMPLES:

```
sage: f = ECM()
sage: n = 508021860739623467191080372196682785441177798407961
sage: f.find_factor(n)
[79792266297612017, 6366805760909027985741435139224233]
```

Note that the input number cannot have more than 4095 digits:

```
sage: f=2^2^14+1
sage: ecm.find_factor(f)
Traceback (most recent call last):
...
ValueError: n must have at most 4095 digits
```

get_last_params ()

Return the parameters (including the curve) of the last ecm run.

In the case that the number was factored successfully, this will return the parameters that yielded the factorization.

OUTPUT:

A dictionary containing the parameters for the most recent factorization.

EXAMPLES:

```
sage: ecm.factor((2^197 + 1)/3)  # long time
[197002597249, 1348959352853811313, 251951573867253012259144010843]
sage: ecm.get_last_params()  # random output
{'poly': 'x^1', 'sigma': '1785694449', 'B1': '8885', 'B2': '1002846'}
```

interact ()

Interactively interact with the ECM program.

EXAMPLES:

```
sage: ecm.interact() # not tested
```

```
one_curve ( n, factor_digits=None, B1=2000, algorithm='ECM', **kwds)
```

Run one single ECM (or P-1/P+1) curve on input n.

Note that trying a single curve is not particularly useful by itself. One typically needs to run over thousands of trial curves to factor n.

INPUT:

- •n a positive integer
- $\hbox{\tt •factor_digits -- integer. Decimal digits estimate of the wanted factor.}$
- •B1 integer. Stage 1 bound (default 2000)
- •algorithm either "ECM" (default), "P-1" or "P+1"

OUTPUT:

a list [p,q] where p and q are integers and n = p * q. If no factor was found, then p = 1 and q = n.

Warning: Neither p nor q in the output is guaranteed to be prime.

EXAMPLES:

```
sage: f = ECM()
sage: n = 508021860739623467191080372196682785441177798407961
sage: f.one_curve(n, B1=10000, sigma=11)
[1, 508021860739623467191080372196682785441177798407961]
sage: f.one_curve(n, B1=10000, sigma=1022170541)
[79792266297612017, 6366805760909027985741435139224233]
sage: n = 432132887883903108009802143314445113500016816977037257
sage: f.one_curve(n, B1=500000, algorithm="P-1")
[67872792749091946529, 6366805760909027985741435139224233]
sage: n = 2088352670731726262548647919416588631875815083
sage: f.one_curve(n, B1=2000, algorithm="P+1", x0=5)
[328006342451, 6366805760909027985741435139224233]
```

recommended_B1 (factor_digits)

Return recommended B1 setting.

INPUT:

•factor_digits - integer. Number of digits.

OUTPUT:

Integer. Recommended settings from http://www.mersennewiki.org/index.php/Elliptic_Curve_Method

EXAMPLES:

```
sage: ecm.recommended_B1(33)
1000000
```

time (n, factor_digits, verbose=False)

Print a runtime estimate.

BUGS:

This method should really return something and not just print stuff on the screen.

INPUT:

- •n a positive integer
- •factor_digits the (estimated) number of digits of the smallest factor

OUTPUT:

An approximation for the amount of time it will take to find a factor of size factor_digits in a single process on the current computer. This estimate is provided by GMP-ECM's verbose option on a single run of a curve.

```
sage: n = next_prime(11^23)*next_prime(11^37)
sage: ecm.time(n, 35)  # random output
Expected curves: 910, Expected time: 23.95m

sage: ecm.time(n, 30, verbose=True)  # random output
GMP-ECM 6.4.4 [configured with MPIR 2.6.0, --enable-asm-redc] [ECM]
Running on localhost.localdomain
```

```
Input number is.
-304481639541418099574459496544854621998616257489887231115912293 (63 digits)
Using MODMULN [mulredc:0, sqrredc:0]
Using B1=250000, B2=128992510, polynomial Dickson(3), sigma=3244548117
dF=2048, k=3, d=19110, d2=11, i0=3
Expected number of curves to find a factor of n digits:
35 40 45 50 55 60 65 70 75 80
4911 70940 1226976 2.5e+07 5.8e+08 1.6e+10 2.7e+13 4e+18 5.4e+23 Inf
Step 1 took 230ms
Using 10 small primes for NTT
Estimated memory usage: 4040K
Initializing tables of differences for F took Oms
Computing roots of F took 9ms
Building F from its roots took 16ms
Computing 1/F took 9ms
Initializing table of differences for G took Oms
Computing roots of G took 8ms
Building G from its roots took 16ms
Computing roots of G took 7ms
Building G from its roots took 16ms
Computing G * H took 6ms
Reducing G * H mod F took 5ms
Computing roots of G took 7ms
Building G from its roots took 17ms
Computing G * H took 5ms
Reducing G * H mod F took 5ms
Computing polyeval(F,G) took 34ms
Computing product of all F(g_i) took Oms
Step 2 took 164ms
Expected time to find a factor of n digits:
35 40 45 50 55 60 65 70 75 80
32.25m 7.76h 5.60d 114.21d 7.27y 196.42y 337811y 5e+10y 7e+15y Inf
Expected curves: 4911, Expected time: 32.25m
```

INTERFACE TO 4TI2

http://www.4ti2.de

You must have the 4ti2 Sage package installed on your computer for this interface to work.

Use sage -i 4ti2 to install the package.

AUTHORS:

- Mike Hansen (2009): Initial version.
- Bjarke Hammersholt Roune (2009-06-26): Added Groebner, made code usable as part of the Sage library and added documentation and some doctests.
- Marshall Hampton (2011): Minor fixes to documentation.

This object defines an interface to the program 4ti2. Each command 4ti2 has is exposed as one method.

```
call ( command, project, verbose=True)
```

Run the 4ti2 program command on the project named project in the directory ().

INPUT:

- •command The 4ti2 program to run.
- •project The file name of the project to run on.
- •verbose Display the output of 4ti2 if True.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[6,10,15]], "test_file")
sage: four_ti_2.call("groebner", "test_file", False) # optional - 4ti2
sage: four_ti_2.read_matrix("test_file.gro") # optional - 4ti2
[-5 0 2]
[-5 3 0]
```

circuits (mat=None, project=None)

Run the 4ti2 program circuits on the parameters. See http://www.4ti2.de/ for details.

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.circuits([1,2,3]) # optional - 4ti2
[ 0 3 -2]
[ 2 -1 0]
[ 3 0 -1]
```

directory ()

Return the directory where the input files for 4ti2 are written by Sage and where 4ti2 is run.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import FourTi2
sage: f = FourTi2("/tmp/")
sage: f.directory()
'/tmp/'
```

graver (mat=None, lat=None, project=None)

Run the 4ti2 program graver on the parameters. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.graver([1,2,3]) # optional - 4ti2
[ 2 -1   0]
[ 3   0 -1]
[ 1   1 -1]
[ 1   -2   1]
[ 0   3 -2]
sage: four_ti_2.graver(lat=[[1,2,3],[1,1,1]]) # optional - 4ti2
[ 1   0 -1]
[ 0   1   2]
[ 1   1   1]
[ 2   1   0]
```

groebner (mat=None, lat=None, project=None)

Run the 4ti2 program groebner on the parameters. This computes a Toric Groebner basis of a matrix. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: A = [6,10,15]
sage: four_ti_2.groebner(A) # optional - 4ti2
[-5  0  2]
[-5  3  0]
sage: four_ti_2.groebner(lat=[[1,2,3],[1,1,1]]) # optional - 4ti2
[-1  0  1]
[ 2  1  0]
```

hilbert (mat=None, lat=None, project=None)

Run the 4ti2 program hilbert on the parameters. See http://www.4ti2.de/ for details.

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.hilbert(four_ti_2._magic3x3()) # optional - 4ti2
[2 0 1 0 1 2 1 2 0]
[1 0 2 2 1 0 0 2 1]
[0 2 1 2 1 0 1 0 2]
[1 2 0 0 1 2 2 0 1]
[1 1 1 1 1 1 1 1 1]
sage: four_ti_2.hilbert(lat=[[1,2,3],[1,1,1]]) # optional - 4ti2
[2 1 0]
```

```
[0 1 2]
[1 1 1]
```

minimize (mat=None, lat=None)

Run the 4ti2 program minimize on the parameters. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.minimize() # optional - 4ti2
Traceback (most recent call last):
...
NotImplementedError: 4ti2 command 'minimize' not implemented in Sage.
```

ppi (*n*)

Run the 4ti2 program ppi on the parameters. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.ppi(3) # optional - 4ti2
[-2  1  0]
[ 0 -3  2]
[-1 -1  1]
[-3  0  1]
[ 1 -2  1]
```

qsolve (mat=None, rel=None, sign=None, project=None)

Run the 4ti2 program qsolve on the parameters. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: A = [[1,1,1],[1,2,3]]
sage: four_ti_2.qsolve(A) # optional - 4ti2
[[], [ 1 -2 1]]
```

rays (mat=None, project=None)

Run the 4ti2 program rays on the parameters. See http://www.4ti2.de/ for details.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.rays(four_ti_2._magic3x3()) # optional - 4ti2
[0 2 1 2 1 0 1 0 2]
[1 0 2 2 1 0 0 2 1]
[1 2 0 0 1 2 2 0 1]
[2 0 1 0 1 2 1 2 0]
```

read matrix (filename)

Read a matrix in 4ti2 format from the file filename in directory directory () .

INPUT:

•filename - The name of the file to read from.

OUTPUT:

The data from the file as a matrix over **Z**.

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[1,2,3],[3,4,6]], "test_file")
sage: four_ti_2.read_matrix("test_file")
[1 2 3]
[3 4 6]
```

temp_project ()

Return an input project file name that has not been used yet.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.temp_project()
'project_...'
```

write_array (array, nrows, ncols, filename)

Write the matrix array of integers (can be represented as a list of lists) to the file filename in directory directory () in 4ti2 format. The matrix must have nrows rows and ncols columns.

INPUT:

- •array A matrix of integers. Can be represented as a list of lists.
- •nrows The number of rows in array.
- •ncols The number of columns in array.
- •file A file name not including a path.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_array([[1,2,3],[3,4,5]], 2, 3, "test_file")
```

write_matrix (mat, filename)

Write the matrix mat to the file filename in 4ti2 format.

INPUT:

- •mat A matrix of integers or something that can be converted to that.
- •filename A file name not including a path.

EXAMPLES:

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_matrix([[1,2],[3,4]], "test_file")
```

write_single_row (row, filename)

Write the list row to the file filename in 4ti2 format as a matrix with one row.

INPUT:

- •row A list of integers.
- •filename A file name not including a path.

```
sage: from sage.interfaces.four_ti_2 import four_ti_2
sage: four_ti_2.write_single_row([1,2,3,4], "test_file")
```

zsolve (mat=None, rel=None, rhs=None, sign=None, lat=None, project=None)

Run the 4ti2 program zsolve on the parameters. See http://www.4ti2.de/ for details.

CHAPTER

SEVEN

INTERFACE TO FRICAS

Todo

• fricas(dilog(x)) should be dilog(-(x-1)), and some more conversions in sage.functions are missing

FriCAS is a free GPL-compatible (modified BSD license) general purpose computer algebra system based on Axiom. The FriCAS website can be found at http://fricas.sourceforge.net/.

AUTHORS:

- Mike Hansen (2009-02): Split off the FriCAS interface from the Axiom interface.
- Martin Rubey, Bill Page (2016-08): Completely separate from Axiom, implement more complete translation from FriCAS to SageMath types.

EXAMPLES:

```
sage: fricas('3 * 5')
  → optional - fricas
15
sage: a = fricas(3) * fricas(5); a
  → optional - fricas
15
```

The type of a is FriCASElement, i.e., an element of the FriCAS interpreter:

The underlying FriCAS type of a is also available, via the type method:

FriCAS objects are normally displayed using "ASCII art":

```
-
3
sage: fricas('x^2 + 3/7') #

→ optional - fricas
2 3
x + -
7
```

Functions defined in FriCAS are available as methods of the fricas object:

We can also create a FriCAS polynomial and apply the function factor from FriCAS. The notation f.factor() is consistent with how the rest of SageMath works:

For many FriCAS types, translation to an appropriate SageMath type is available:

Control-C interruption works well with the FriCAS interface. For example, try the following sum but with a much bigger range, and hit control-C:

Let us demonstrate some features of FriCAS. FriCAS can guess a differential equation for the generating function for integer partitions:

```
[x ]f(x):
    2 3 (iv) 2 2, 3 ,,, 2 2,,, 2
    x f(x) f (x) + (20x f(x) f (x) + 5x f(x) )f (x) - 39x f(x) f (x)

+
    2 , 2 2, 3 ,, 2, 4
    (12x f(x)f(x) - 15x f(x) f (x) + 4f(x) )f (x) + 6x f (x)

+
    , 3 2, 2
    10x f(x)f(x) - 16f(x) f (x)

=
    0
,
    f(x) = 1 + x + 2x + 3x + O(x)]
```

FriCAS can solve linear ordinary differential equations:

```
sage: fricas.set("y", "operator y")
→optional - fricas
sage: fricas.set("deq", "x^3*D(y x, x, 3) + x^2*D(y x, x, 2) - 2*x*D(y x, x) + 2*y x -
\rightarrow 2*x^4") # optional - fricas
sage: fricas.set("sol", "solve(deq, y, x)"); fricas("sol")
→optional - fricas
           5 3 2
                                                  3
                                            2
                                                         3
        x - 10x + 20x + 4  2x - 3x + 1 x - 1 x - 3x - 1
[particular= -----, basis= [-----, ----, -----]]
             15x
                                          X
sage: fricas("sol.particular").sage()
                                                                     #__
→optional - fricas
1/15*(x^5 - 10*x^3 + 20*x^2 + 4)/x
sage: fricas("sol.basis").sage()
→optional - fricas
[(2*x^3 - 3*x^2 + 1)/x, (x^3 - 1)/x, (x^3 - 3*x^2 - 1)/x]
sage: fricas.eval(")clear values y deq sol")
⇔optional - fricas
```

FriCAS can expand expressions into series:

```
2 1 2 31 2
x - - x + --- x + O(x)
          360
sage: a.coefficient(9/2).sage()
                                                                   #
→optional - fricas
sage: x = fricas("x::TaylorSeries Fraction Integer")
→optional - fricas
sage: y = fricas("y::TaylorSeries Fraction Integer")
→optional - fricas
sage: 2*(1+2*x+sqrt(1-4*x)-2*x*y).recip()
⇔optional - fricas
                       2 2 3
                                   4
  1 + (x y + x) + 2x + (x y + 2x y + 6x) + (4x y + 18x)
         4 2 5
                       6
                               5 2
    3 3
  (x y + 3x y + 13x y + 57x) + (6x y + 40x y + 186x)
                                                7 2
          5 3
                 6 2
                                         6 3
  (x y + 4x y + 21x y + 130x y + 622x) + (8x y + 66x y + 432x y + 2120x)
                 7 3
          6 4
                           8 2
                                    9
  (x y + 5x y + 30x y + 220x y + 1466x y + 7338x) + O(11)
```

FriCAS does some limits right:

```
 \begin{array}{lll} \textbf{class} \ \texttt{sage.interfaces.fricas. FriCAS} \ (\textit{name='fricas'}, & \textit{command='fricas} & \textit{-nox} & \textit{-noclef'}, \\ & \textit{script\_subdirectory=None}, & \textit{logfile=None}, & \textit{server=None}, \\ & \textit{server\_tmpdir=None}) \end{array}
```

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect

Interface to a FriCAS interpreter.

console ()

Spawn a new FriCAS command-line session.

EXAMPLES:

```
sage: fricas.console() # not_

→tested

FriCAS (AXIOM fork) Computer Algebra System

Version: FriCAS 1.0.5

Timestamp: Thursday February 19, 2009 at 06:57:33

Issue )copyright to view copyright notices.

Issue )summary for a summary of useful system commands.

Issue )quit to leave AXIOM and return to shell.
```

eval (code, strip=True, synchronize=False, locals=None, allow_use_file=True, split_lines='nofile', reformat=True, **kwds) Evaluate code using FriCAS.

Except reformat, all arguments are passed to sage.interfaces.expect.Expect.eval().

INPUT:

•reformat -bool; remove the output markers when True.

This can also be used to pass system commands to FriCAS.

EXAMPLES:

get (var)

Get the string representation of the value (more precisely, the OutputForm) of a variable or expression in FriCAS.

If FriCAS cannot evaluate var an error is raised.

EXAMPLES:

```
sage: fricas.set('xx', '2')
→optional - fricas
sage: fricas.get('xx')
⇔optional - fricas
121
sage: a = fricas('(1 + sqrt(2))^5')
→optional - fricas
sage: fricas.get(a.name())
→optional - fricas
  +-+\r\n29\\\ + 41
sage: fricas.get('(1 + sqrt(2))^5')
→optional - fricas
  +-+\r\n29\\|2 + 41'
sage: fricas.new('(1 + sqrt(2))^5')
→optional - fricas
  +-+
29 | 2 + 41
```

get_boolean (var)

Return the value of a FriCAS boolean as a boolean, without checking that it is a boolean.

TESTS:

get_integer (var)

Return the value of a FriCAS integer as an integer, without checking that it is an integer.

TESTS:

get_string (var)

Return the value of a FriCAS string as a string, without checking that it is a string.

TESTS:

We test that strings are returned properly:

get_unparsed_InputForm (var)

Return the unparsed InputForm as a string.

Todo

•catch errors, especially when InputForm is not available:

- -for example when integration returns "failed"
- -UnivariatePolynomial
- •should we provide workarounds, too?

TESTS:

```
sage: fricas.get_unparsed_InputForm('1..3')
    →optional - fricas
'1..3$Segment(Integer())'
```

set (var, value)

Set a variable to a value in FriCAS.

INPUT:

•var, value: strings, the first representing a valid FriCAS variable identifier, the second a FriCAS expression.

OUTPUT: None

class sage.interfaces.fricas. FriCASElement (parent, value, is_name=False, name=None)
 Bases: sage.interfaces.expect.ExpectElement

Instances of this class represent objects in FriCAS.

Using the method sage () we can translate some of them to SageMath objects:

```
_sage_ ( )
```

Convert self to a Sage object.

EXAMPLES:

Floats:

```
sage: fricas(2.1234).sage()
                                                                      #__
→optional - fricas
2.12340000000000
sage: _.parent()
→optional - fricas
Real Field with 53 bits of precision
sage: a = RealField(100)(pi)
⇔optional - fricas
sage: fricas(a).sage()
                                                                      #__
→optional - fricas
3.1415926535897932384626433833
sage: _.parent()
→optional - fricas
Real Field with 100 bits of precision
sage: fricas(a).sage() == a
→optional - fricas
True
sage: fricas(2.0).sage()
⇔optional - fricas
2.000000000000000
sage: _.parent()
                                                                      #__
→optional - fricas
Real Field with 53 bits of precision
```

Algebraic numbers:

Integers modulo n:

```
sage: fricas("((42^17)^1783)::IntegerMod(5^(5^5))").sage() == Integers(5^(5^5))((42^17)^1783) # optional - fricas
True
```

We can also convert FriCAS's polynomials to Sage polynomials:

```
sage: a = fricas(x^2 + 1); a.typeOf()
                                                                      #__
→optional - fricas
Polynomial(Integer)
sage: a.sage()
→optional - fricas
x^2 + 1
sage: _.parent()
→optional - fricas
Univariate Polynomial Ring in x over Integer Ring
sage: fricas('x^2 + y^2 + 1/2').sage()
                                                                      #__
→optional - fricas
y^2 + x^2 + 1/2
sage: _.parent()
→optional - fricas
Multivariate Polynomial Ring in y, x over Rational Field
sage: fricas("1$Polynomial Integer").sage()
                                                                      #___
→optional - fricas
sage: fricas("x^2/2").sage()
                                                                      #__
→optional - fricas
1/2*x^2
```

Rational functions:

Expressions:

```
sage: fricas("sin(x+y)/exp(z)*log(1+%e)").sage()
                                                                       #__
→optional - fricas
e^{(-z)} * log(e + 1) * sin(x + y)
sage: fricas("factorial(n)").sage()
→optional - fricas
factorial(n)
sage: fricas("integrate(sin(x+y), x=0..1)").sage()
                                                                       #
→optional - fricas
-\cos(y + 1) + \cos(y)
sage: fricas("integrate(x*sin(1/x), x=0..1)").sage()
                                                                       #__
→optional - fricas
'failed'
sage: fricas("integrate(\sin((x^2+1)/x), x)").sage()
                                                                       #__
→optional - fricas
integral(sin((x^2 + 1)/x), x)
```

Todo

•Converting matrices and lists takes much too long.

Matrices:

Lists:

Error handling:

```
sage: s = fricas.guessPade("[fibonacci i for i in 0..10]"); s
                                                   #__
→optional - fricas
      X
  n
[[[x ]- -----]]
      2
     x + x - 1
sage: s.sage()
                                                   #___
→optional - fricas
Traceback (most recent call last):
NotImplementedError: The translation of the FriCAS Expression rootOfADE(n,...
\hookrightarrow ()) to sage is not yet implemented.
sage: s = fricas("series(sqrt(1+x), x=0)"); s
                                                   #___
→optional - fricas
   1 1 2 1 3 5 4 7 5 21 6 33 7 429 8
 8
          16
                   128
                       256
                                 1024
                                       2048
                                                32768
 715 9 2431 10 11
 ---- x - ---- x + O(x)
 65536
        262144
sage: s.sage()
                                                   #__
→optional - fricas
Traceback (most recent call last):
NotImplementedError: The translation of the FriCAS object
              1 3
                    5 4
                          7 5
                                 21
                                     6
                                        33
 256
                                 1024
                                         2048
        8
             16
                   128
                                                32768
```

```
+
715 9 2431 10 11
----- x - ----- x + O(x )
65536 262144

to sage is not yet implemented:
An error occurred when FriCAS evaluated 'unparse(...::InputForm)':

Cannot convert the value from type Any to InputForm .
```

bool ()

Coerce the expression into a boolean.

EXAMPLES:

gen(n)

Return an error, since the n-th generator in FriCAS is not well defined.

```
class sage.interfaces.fricas. FriCASExpectFunction (parent, name)
```

Bases: sage.interfaces.expect.ExpectFunction

Translate the pythonized function identifier back to a FriCAS operation name.

TESTS:

class sage.interfaces.fricas. FriCASFunctionElement (object, name)

Bases: sage.interfaces.expect.FunctionElement

Make FriCAS operation names valid python function identifiers.

TESTS:

sage.interfaces.fricas.fricas_console()

Spawn a new FriCAS command-line session.

EXAMPLES:

```
sage: fricas_console() # not_

→tested

FriCAS (AXIOM fork) Computer Algebra System

Version: FriCAS 1.0.5

Timestamp: Thursday February 19, 2009 at 06:57:33

Issue )copyright to view copyright notices.

Issue )summary for a summary of useful system commands.

Issue )quit to leave AXIOM and return to shell.
```

sage.interfaces.fricas.is_FriCASElement (x)

Return True if x is of type FriCASElement.

EXAMPLES:

sage.interfaces.fricas.reduce_load_fricas()

Returns the FriCAS interface object defined in :sage.interfaces.fricas.

CHAPTER

EIGHT

INTERFACE TO FROBBY FOR FAST COMPUTATIONS ON MONOMIAL IDEALS.

The software package Frobby provides a number of computations on monomial ideals. The current main feature is the socle of a monomial ideal, which is largely equivalent to computing the maximal standard monomials, the Alexander dual or the irreducible decomposition.

Operations on monomial ideals are much faster than algorithms designed for ideals in general, which is what makes a specialized library for these operations on monomial ideals useful.

AUTHORS:

• Bjarke Hammersholt Roune (2008-04-25): Wrote the Frobby C++ program and the initial version of the Python interface.

NOTES:

The official source for Frobby is http://www.broune.com/frobby, which also has documentation and papers describing the algorithms used.

```
class sage.interfaces.frobby. Frobby
```

alexander_dual (monomial_ideal)

This function computes the Alexander dual of the passed-in monomial ideal. This ideal is the one corresponding to the simplicial complex whose faces are the complements of the nonfaces of the simplicial complex corresponding to the input ideal.

INPUT:

•monomial_ideal – The monomial ideal to decompose.

OUTPUT:

The monomial corresponding to the Alexander dual.

EXAMPLES:

This is a simple example of computing irreducible decomposition.

We see how it is much faster to compute this with frobby than the built-in procedure for simplicial complexes.

sage: t=simplicial_complexes.PoincareHomologyThreeSphere() # optional - frobby sage: R=PolynomialRing(QQ,16,'x') # optional - frobby sage: I=R.ideal([prod([R.gen(i-1) for i in a]) for a in t.facets()]) # optional - frobby sage: len(frobby.alexander_dual(I).gens()) # optional - frobby 643

associated_primes (monomial_ideal)

This function computes the associated primes of the passed-in monomial ideal.

INPUT:

•monomial_ideal – The monomial ideal to decompose.

OUTPUT:

A list of the associated primes of the monomial ideal. These ideals are constructed in the same ring as monomial_ideal is.

EXAMPLES:

```
sage: R.<d,b,c>=QQ[] # optional - frobby
sage: I=[d*b*c,b^2*c,b^10,d^10]*R # optional - frobby
sage: frobby.associated_primes(I) # optional - frobby
[Ideal (d, b) of Multivariate Polynomial Ring in d, b, c over Rational Field,
Ideal (d, b, c) of Multivariate Polynomial Ring in d, b, c over Rational_
→Field]
```

dimension (monomial ideal)

This function computes the dimension of the passed-in monomial ideal.

INPUT:

•monomial ideal – The monomial ideal to decompose.

OUTPUT:

The dimension of the zero set of the ideal.

EXAMPLES:

```
sage: R.<d,b,c>=QQ[] # optional - frobby
sage: I=[d*b*c,b^2*c,b^10,d^10]*R # optional - frobby
sage: frobby.dimension(I) # optional - frobby
1
```

hilbert (monomial_ideal)

Computes the multigraded Hilbert-Poincare series of the input ideal. Use the -univariate option to get the univariate series.

The Hilbert-Poincare series of a monomial ideal is the sum of all monomials not in the ideal. This sum can be written as a (finite) rational function with $(x_1 - 1)(x_2 - 1)...(x_n - 1)$ in the denominator, assuming the variables of the ring are $x_1, x_2, ..., x_n$. This action computes the polynomial in the numerator of this fraction.

INPUT:

monomial_ideal - A monomial ideal.

OUTPUT:

A polynomial in the same ring as the ideal.

irreducible_decomposition (monomial_ideal)

This function computes the irreducible decomposition of the passed-in monomial ideal. I.e. it computes the unique minimal list of irreducible monomial ideals whose intersection equals monomial_ideal.

INPUT:

•monomial_ideal – The monomial ideal to decompose.

OUTPUT:

A list of the unique irredundant irreducible components of monomial_ideal. These ideals are constructed in the same ring as monomial ideal is.

EXAMPLES:

This is a simple example of computing irreducible decomposition.

```
sage: (x, y, z) = QQ['x,y,z'].gens() # optional - frobby
sage: id = ideal(x ** 2, y ** 2, x * z, y * z) # optional - frobby
sage: decom = frobby.irreducible_decomposition(id) # optional - frobby
sage: true_decom = [ideal(x, y), ideal(x ** 2, y ** 2, z)] # optional - frobby
sage: set(decom) == set(true_decom) # use sets to ignore order # optional - frobby
True
```

We now try the special case of the zero ideal in different rings.

We should also try PolynomialRing(QQ, names=[]), but it has a bug which makes that impossible (see trac ticket #3028).

```
sage: rings = [ZZ['x'], CC['x,y']] # optional - frobby
sage: allOK = True # optional - frobby
sage: for ring in rings: # optional - frobby
... id0 = ring.ideal(0) # optional - frobby
... decom0 = frobby.irreducible_decomposition(id0) # optional - frobby
... allOK = allOK and decom0 == [id0] # optional - frobby
sage: allOK # optional - frobby
True
```

Finally, we try the ideal that is all of the ring in different rings.

```
sage: rings = [ZZ['x'], CC['x,y']] # optional - frobby
sage: allOK = True # optional - frobby
sage: for ring in rings: # optional - frobby
... idl = ring.ideal(1) # optional - frobby
... decom1 = frobby.irreducible_decomposition(idl) # optional - frobby
... allOK = allOK and decom1 == [idl] # optional - frobby
sage: allOK # optional - frobby
True
```



CHAPTER

NINE

INTERFACE TO GAP

Sage provides an interface to the GAP system. This system provides extensive group theory, combinatorics, etc.

The GAP interface will only work if GAP is installed on your computer; this should be the case, since GAP is included with Sage. The interface offers three pieces of functionality:

- 1. gap_console() A function that dumps you into an interactive command-line GAP session.
- 2. gap (expr) Evaluation of arbitrary GAP expressions, with the result returned as a string.
- 3. gap.new(expr) Creation of a Sage object that wraps a GAP object. This provides a Pythonic interface to GAP. For example, if f=gap.new(10), then f.Factors() returns the prime factorization of 10 computed using GAP.

9.1 First Examples

We factor an integer using GAP:

```
sage: n = gap(20062006); n
20062006
sage: n.parent()
Gap
sage: fac = n.Factors(); fac
[ 2, 17, 59, 73, 137 ]
sage: fac.parent()
Gap
sage: fac[1]
```

9.2 GAP and Singular

This example illustrates conversion between Singular and GAP via Sage as an intermediate step. First we create and factor a Singular polynomial.

```
sage: singular(389)
389
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: f = singular('9*x^16-18*x^13*y^2-9*x^12*y^3+9*x^10*y^4-18*x^11*y^2+36*x^8*y^4+18*x^7*y^5-18*x^5*y^6+9*x^6*y^4-18*x^3*y^6-9*x^2*y^7+9*y^8')
sage: F = f.factorize()
sage: print(F)
[1]:
```

```
_[1]=9
_[2]=x^6-2*x^3*y^2-x^2*y^3+y^4
_[3]=-x^5+y^2
[2]:
1,1,2
```

Next we convert the factor $-x^5 + y^2$ to a Sage multivariate polynomial. Note that it is important to let x and y be the generators of a polynomial ring, so the eval command works.

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: s = F[1][3].sage_polystring(); s
'-x**5+y**2'
sage: g = eval(s); g
-x^5 + y^2
```

Next we create a polynomial ring in GAP and obtain its indeterminates:

```
sage: R = gap.PolynomialRing('Rationals', 2); R
PolynomialRing( Rationals, ["x_1", "x_2"] )
sage: I = R.IndeterminatesOfPolynomialRing(); I
[ x_1, x_2 ]
```

In order to eval g in GAP, we need to tell GAP to view the variables x0 and x1 as the two generators of R. This is the one tricky part. In the GAP interpreter the object I has its own name (which isn't I). We can access its name using I.name().

```
sage: _ = gap.eval("x := %s[1];; y := %s[2];; "%(I.name(), I.name()))
```

Now x_0 and x_1 are defined, so we can construct the GAP polynomial f corresponding to g:

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: f = gap(str(g)); f
-x_1^5+x_2^2
```

We can call GAP functions on f. For example, we evaluate the GAP Value function, which evaluates f at the point (1,2).

```
sage: f.Value(I, [1,2])
3
sage: g(1,2)  # agrees
3
```

9.3 Saving and loading objects

Saving and loading GAP objects (using the dumps method, etc.) is *not* supported, since the output string representation of Gap objects is sometimes not valid input to GAP. Creating classes that wrap GAP objects is supported, via simply defining the a _gap_init_ member function that returns a string that when evaluated in GAP constructs the object. See groups/perm_gps/permgroup.py for a nontrivial example of this.

9.4 Long Input

The GAP interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using gap.eval for long input is much less robust, and is not recommended.

```
sage: t = '"%s"'%10^10000 # ten thousand character string.
sage: a = gap(t)
```

9.5 Changing which GAP is used

Use this code to change which GAP interpreter is run. E.g.,

```
import sage.interfaces.gap
sage.interfaces.gap_cmd = "/usr/local/bin/gap"
```

AUTHORS:

- David Joyner and William Stein: initial version(s)
- William Stein (2006-02-01): modified gap_console command so it uses exactly the same startup command as Gap.__init__.
- William Stein (2006-03-02): added tab completions: gap.[tab], x = gap(...), x.[tab], and docs, e.g., gap.function? and x.function?

Interface to the GAP interpreter.

AUTHORS:

•William Stein and David Joyner

console ()

Spawn a new GAP command-line session.

EXAMPLES:

cputime (t=None)

Returns the amount of CPU time that the GAP session has used. If t is not None, then it returns the difference between the current CPU time and t.

```
sage: t = gap.cputime()
sage: t #random
0.1360000000000001
```

```
sage: gap.Order(gap.SymmetricGroup(5))
120
sage: gap.cputime(t) #random
0.0599999999999998
```

get (var, use_file=False)

Get the string representation of the variable var.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
```

help (s, pager=True)

Print help on a given topic.

EXAMPLES:

```
sage: print(gap.help('SymmetricGroup', pager=False))

50 Group Libraries

When you start GAP, it already knows several groups. Currently GAP initially knows the following groups:
...
```

save_workspace ()

Save the GAP workspace.

TESTS:

We make sure that trac ticket #9938 (GAP does not start if the path to the GAP workspace file contains more than 82 characters) is fixed:

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
```

set_seed (seed=None)

Sets the seed for gap interpeter. The seed should be an integer.

```
sage: g = Gap()
sage: g.set_seed(0)
```

```
sage: [g.Random(1,10) for i in range(5)]
         [2, 3, 3, 4, 2]
class sage.interfaces.gap. GapElement (parent, value, is_name=False, name=None)
     Bases: sage.interfaces.gap.GapElement_generic
     str ( use_file=False)
         EXAMPLES:
         sage: print(gap(2))
         2
class sage.interfaces.gap. GapElement_generic ( parent, value, is_name=False, name=None)
     Bases:
                          sage.interfaces.tab completion.ExtraTabCompletion
     sage.interfaces.expect.ExpectElement
     Generic interface to the GAP3/GAP4 interpreters.
     AUTHORS:
        •William Stein and David Joyner (interface for GAP4)
        •Franco Saliola (Feb 2010): refactored to separate out the generic code
     bool ()
         EXAMPLES:
         sage: bool(gap(2))
         True
         sage: gap(0).bool()
         False
         sage: gap('false').bool()
         False
     is string()
         Tell whether this element is a string.
         EXAMPLES:
         sage: gap('"abc"').is_string()
         sage: gap('[1,2,3]').is_string()
         False
class sage.interfaces.gap. GapFunction ( parent, name)
     Bases: sage.interfaces.expect.ExpectFunction
class sage.interfaces.gap. GapFunctionElement (obj, name)
     Bases: sage.interfaces.expect.FunctionElement
class sage.interfaces.gap. Gap_generic ( name, prompt,
                                                             command=None,
                                                                              server=None.
                                             server_tmpdir=None,
                                                                              ulimit=None,
                                             maxread=None,
                                                                   script_subdirectory=None,
                                             restart_on_ctrlc=False,
                                                                        verbose_start=False,
                                             init code=[],
                                                            max_startup_time=None,
                                            file=None, eval_using_file_cutoff=0, do_cleaner=True,
                                             remote_cleaner=False,
                                                                     path=None,
                                                                                    termi-
                                             nal_echo=True)
                          sage.interfaces.tab_completion.ExtraTabCompletion
     Bases:
```

```
sage.interfaces.expect.Expect
```

Generic interface to the GAP3/GAP4 interpreters.

AUTHORS:

- •William Stein and David Joyner (interface for GAP4)
- •Franco Saliola (Feb 2010): refactored to separate out the generic code

```
eval (x, newlines=False, strip=True, split lines=True, **kwds)
```

Send the code in the string s to the GAP interpreter and return the output as a string.

INPUT:

- •s string containing GAP code.
- •newlines bool (default: True); if False, remove all backslash-newlines inserted by the GAP output formatter.
- •strip -ignored
- •split_lines bool (default: True); if True then each line is evaluated separately. If False, then the whole block of code is evaluated all at once.

EXAMPLES:

```
sage: gap.eval('2+2')
'4'
sage: gap.eval('Print(4); #test\n Print(6);')
sage: gap.eval('Print("#"); Print(6);')
'#6'
sage: gap.eval('4; \n 6;')
'4\n6'
sage: gap.eval('if 3>2 then\nPrint("hi");\nfi;')
sage: gap.eval('## this is a test\nPrint("OK")')
'OK'
sage: gap.eval('Print("This is a test. Oh no, a #"); # but this is a,
→comment\nPrint("OK")')
'This is a test. Oh no, a #OK'
sage: gap.eval('if 4>3 then')
sage: gap.eval('Print("Hi how are you?")')
'Hi how are you?'
sage: gap.eval('fi')
```

function_call (function, args=None, kwds=None)

Calls the GAP function with args and kwds.

EXAMPLES:

```
sage: gap.function_call('SymmetricGroup', [5])
SymmetricGroup( [ 1 .. 5 ] )
```

If the GAP function does not return a value, but prints something to the screen, then a string of the printed output is returned.

```
sage: s = gap.function_call('Display', [gap.SymmetricGroup(5).

→CharacterTable()])
sage: type(s)
```

```
<class 'sage.interfaces.interface.AsciiArtString'>
sage: s.startswith('CT')
True
```

TESTS:

If the function call is too long, two gap.eval calls are made since returned values from commands in a file cannot be handled properly:

```
sage: g = Gap()
sage: g.function_call("ConjugacyClassesSubgroups", sage.interfaces.gap.

GapElement(g, 'SymmetricGroup(2)', name = 'a_variable_with_a_very_very_very_

olong_name'))
[ ConjugacyClassSubgroups(SymmetricGroup([1 .. 2]),Group([()])),
    ConjugacyClassSubgroups(SymmetricGroup([1 .. 2]),SymmetricGroup([1 .. 2]))]
```

When the command itself is so long that it warrants use of a temporary file to be communicated to GAP, this does not cause problems since the file will contain a single command:

get_record_element (record, name)

Return the element of a GAP record identified by name.

INPUT:

- •record a GAP record
- \bullet name -str

OUTPUT:

• Gap Element

EXAMPLES:

```
sage: rec = gap('rec( a := 1, b := "2" )')
sage: gap.get_record_element(rec, 'a')
1
sage: gap.get_record_element(rec, 'b')
2
```

TESTS:

```
sage: rec = gap('rec( a := 1, b := "2" )')
sage: type(gap.get_record_element(rec, 'a'))
<class 'sage.interfaces.gap.GapElement'>
```

interrupt (tries=None, timeout=1, quit_on_fail=True)

Interrupt the GAP process

Gap installs a SIGINT handler, we call it directly instead of trying to sent Ctrl-C. Unlike <code>interrupt()</code> , we only try once since we are knowing what we are doing.

Sometimes GAP dies while interrupting.

EXAMPLES:

```
sage: gap._eval_line('while(1=1) do i:=1;; od;', wait_for_prompt=False);
''
sage: rc = gap.interrupt(timeout=1)
sage: [ gap(i) for i in range(10) ] # check that it is still working
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

TESTS:

```
sage: gap('"finished computation"'); gap.interrupt(); gap('"ok"')
finished computation
True
ok
```

load_package (pkg, verbose=False)

Load the Gap package with the given name.

If loading fails, raise a RuntimeError exception.

TESTS:

unbind (var)

Clear the variable named var.

EXAMPLES:

```
sage: gap.set('x', '2')
sage: gap.get('x')
'2'
sage: gap.unbind('x')
sage: gap.get('x')
Traceback (most recent call last):
...
RuntimeError: Gap produced error output
Error, Variable: 'x' must have a value
...
```

version()

Returns the version of GAP being used.

EXAMPLES:

```
sage: print(gap.version())
4.8...
```

```
sage.interfaces.gap. gap_command ( use_workspace_cache=True, local=True)
sage.interfaces.gap. gap_console ( )
    Spawn a new GAP command-line session.
```

Note that in gap-4.5.7 you cannot use a workspace cache that had no commandline to restore a gap session with commandline.

TESTS:

```
sage: import subprocess
sage: from sage.interfaces.gap import gap_command
sage: cmd = 'echo "quit;" | ' + gap_command(use_workspace_cache=False)[0]
sage: gap_startup = subprocess.check_output(cmd, shell=True, stderr=subprocess.

STDOUT)
sage: 'http://www.gap-system.org' in gap_startup
True
sage: 'Error' not in gap_startup
True
sage: 'sorry' not in gap_startup
True
```

sage.interfaces.gap. **gap_reset_workspace** (max_workspace_size=None, verbose=False) Call this to completely reset the GAP workspace, which is used by default when Sage first starts GAP.

The first time you start GAP from Sage, it saves the startup state of GAP in a file \$HOME/.sage/gap/workspace-HASH\$, where HASH is a hash of the directory where Sage is installed.

This is useful, since then subsequent startup of GAP is at least 10 times as fast. Unfortunately, if you install any new code for GAP, it won't be noticed unless you explicitly load it, e.g., with gap.load_package("my_package")

The packages sonata, guava, factint, gapdoc, grape, design, toric, and laguna are loaded in all cases before the workspace is saved, if they are available.

TESTS:

Check that gap_reset_workspace still works when GAP_DIR doesn't exist, see trac ticket #14171:

Check that the race condition from trac ticket #14242 has been fixed. We temporarily need to change the worksheet filename.

```
sage: ORIGINAL_WORKSPACE = sage.interfaces.gap.WORKSPACE
sage: sage.interfaces.gap.WORKSPACE = tmp_filename()
```

```
sage: from multiprocessing import Process
sage: import time
sage: gap = Gap() # long time (reset GAP session)
sage: P = [Process(target=gap, args=("14242",)) for i in range(4)]
sage: for p in P: # long time, indirect doctest
...: p.start()
...: time.sleep(0.2)
sage: for p in P: # long time
...: p.join()
sage: os.unlink(sage.interfaces.gap.WORKSPACE) # long time
sage: sage.interfaces.gap.WORKSPACE = ORIGINAL_WORKSPACE
```

sage.interfaces.gap. get_gap_memory_pool_size ()

Get the gap memory pool size for new GAP processes.

EXAMPLES:

```
sage: from sage.interfaces.gap import get_gap_memory_pool_size
sage: get_gap_memory_pool_size() # random output
1534059315
```

sage.interfaces.gap. ${\tt gfq_gap_to_sage}$ (x,F)

INPUT:

- •x GAP finite field element
- •F Sage finite field

OUTPUT: element of F

EXAMPLES:

```
sage: x = gap('Z(13)')
sage: F = GF(13, 'a')
sage: F(x)
2
sage: F(gap('0*Z(13)'))
0
sage: F = GF(13^2, 'a')
sage: x = gap('Z(13)')
sage: F(x)
2
sage: x = gap('Z(13)')
sage: F(x)
12*a + 11
sage: F.multiplicative_generator()^3
12*a + 11
```

TESTS:

Check that trac ticket #18048 is fixed:

```
sage: K.<a> = GF(16)
sage: b = a^2 + a
sage: K(b._gap_())
a^2 + a
```

AUTHOR:

•David Joyner and William Stein

```
sage.interfaces.gap. intmod_gap_to_sage (x)
INPUT:
```

•x – Gap integer mod ring element

EXAMPLES:

```
sage: a = gap(Mod(3, 18)); a
ZmodnZObj(3, 18)
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
sage: b.parent()
Ring of integers modulo 18
sage: a = gap(Mod(3, 17)); a
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
sage: b.parent()
Finite Field of size 17
sage: a = gap(Mod(0, 17)); a
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
sage: b.parent()
Finite Field of size 17
sage: a = gap(Mod(3, 65537)); a
ZmodpZObj( 3, 65537 )
sage: b = sage.interfaces.gap.intmod_gap_to_sage(a); b
sage: b.parent()
Ring of integers modulo 65537
```

sage.interfaces.gap. $is_GapElement$ (x)

Returns True if x is a GapElement.

EXAMPLES:

```
sage: from sage.interfaces.gap import is_GapElement
sage: is_GapElement(gap(2))
True
sage: is_GapElement(2)
False
```

```
sage.interfaces.gap. reduce_load ()
```

This is for backwards compatibility only.

To be precise, it only serves at unpickling the invalid gap elements that are stored in the pickle jar.

```
sage: from sage.interfaces.gap import reduce_load
sage: reduce_load()
doctest:...: DeprecationWarning: This function is only used to unpickle invalid_
--objects
See http://trac.sagemath.org/18848 for details.
<repr(<sage.interfaces.gap.GapElement at ...>) failed:
ValueError: The session in which this object was defined is no longer running.>
```

By trac ticket #18848, pickling actually often works:

```
sage: loads(dumps(gap([1,2,3])))
[ 1, 2, 3 ]
```

```
sage.interfaces.gap. reduce_load_GAP ()
```

Returns the GAP interface object defined in sage.interfaces.gap.

EXAMPLES:

```
sage: from sage.interfaces.gap import reduce_load_GAP
sage: reduce_load_GAP()
Gap
```

```
sage.interfaces.gap. set_gap_memory_pool_size ( size_in_bytes)
```

Set the desired gap memory pool size.

Subsequently started GAP/libGAP instances will use this as default. Currently running instances are unchanged.

GAP will only reserve size_in_bytes address space. Unless you actually start a big GAP computation, the memory will not be used. However, corresponding swap space will be reserved so that GAP will always be able to use the reserved address space if needed. While nothing is actually written to disc as long as you don't run a big GAP computation, the reserved swap space will not be available for other processes.

INPUT:

•size_in_bytes - integer. The desired memory pool size.

CHAPTER

TEN

INTERFACE TO GAP3

This module implements an interface to GAP3.

AUTHORS:

- Franco Saliola (February 2010)
- Christian Stump (March 2016)

Warning: The experimental package for GAP3 is Jean Michel's pre-packaged GAP3, which is a minimal GAP3 distribution containing packages that have no equivalent in GAP4, see trac ticket #20107 and also

https://webusers.imj-prg.fr/~jean.michel/gap3/

10.1 Obtaining GAP3

Instead of installing the experimental GAP3 package, one can as well install by hand either of the following two versions of GAP3:

• Frank Luebeck maintains a GAP3 Linux executable, optimized for i686 and statically linked for jobs of 2 GByte or more:

http://www.math.rwth-aachen.de/~Frank.Luebeck/gap/GAP3

• or you can download GAP3 from the GAP website below. Since GAP3 is no longer supported, it may not be easy to install this version.

http://www.gap-system.org/Gap3/Download3/download.html

10.2 Changing which GAP3 is used

Warning: There is a bug in the pexpect module (see trac ticket #8471) that prevents the following from working correctly. For now, just make sure that gap3 is in your PATH.

Sage assumes that GAP3 can be launched with the command gap3; that is, Sage assumes that the command gap3 is in your PATH. If this is not the case, then you can start GAP3 using the following command:

sage: gap3 = Gap3(command='/usr/local/bin/gap3') #not tested

10.3 Functionality and Examples

The interface to GAP3 offers the following functionality.

1. gap3 (expr) - Evaluation of arbitrary GAP3 expressions, with the result returned as a Sage object wrapping the corresponding GAP3 element:

```
sage: a = gap3('3+2') #optional - gap3
sage: a #optional - gap3
5
sage: type(a) #optional - gap3

<class 'sage.interfaces.gap3.GAP3Element'>
```

```
sage: S5 = gap3('SymmetricGroup(5)')  #optional - gap3
sage: S5  #optional - gap3
Group((1,5), (2,5), (3,5), (4,5))
sage: type(S5)  #optional - gap3

<class 'sage.interfaces.gap3.GAP3Record'>
```

This provides a Pythonic interface to GAP3. If <code>gap_function</code> is the name of a GAP3 function, then the syntax <code>gap_element.gap_function()</code> returns the <code>gap_element</code> obtained by evaluating the command <code>gap_function(gap_element)</code> in GAP3:

```
sage: S5.Size() #optional - gap3
120
sage: S5.CharTable() #optional - gap3
CharTable( Group( (1,5), (2,5), (3,5), (4,5) ) )
```

Alternatively, you can instead use the syntax gap3.gap_function(gap_element):

If $gap_element$ corresponds to a GAP3 record, then $gap_element.recfield$ provides a means to access the record element corresponding to the field recfield:

```
sage: S5.IsRec()
                                                     #optional - gap3
true
                                                     #optional - gap3
sage: S5.recfields()
['isDomain', 'isGroup', 'identity', 'generators', 'operations',
'isPermGroup', 'isFinite', '1', '2', '3', '4', 'degree']
sage: S5.identity
                                                     #optional - gap3
()
sage: S5.degree
                                                     #optional - gap3
sage: S5.1
                                                     #optional - gap3
(1,5)
sage: S5.2
                                                     #optional - gap3
(2,5)
```

2. By typing %gap3 or gap3.interact() at the command-line, you can interact directly with the underlying GAP3 session.

```
sage: gap3.interact() #not tested
```

```
--> Switching to Gap3 <--
gap3:
```

3. You can start a new GAP3 session as follows:

```
sage: gap3.console()
                                                    #not tested
                                Lehrstuhl D fuer Mathematik
             #######
           ###
                  ####
                                 RWTH Aachen
          ##
                     ##
         ##
                     #
                                   #######
                                                       ########
        ##
                                        ##
                                                      ## #
        ##
                                          ##
                                                         #
                                                                ##
        ####
                    ##
                                 ##
                                         #
                                                         #
                                                                 ##
                   ###
                                 ##
                                                         ##
         #####
                                          ##
                                                         #######
           #########
                    ##
                                Version 3
                   ###
                                Release 4.4
                  ## #
                                 18 Apr 97
                 ## #
                ##
                     # Alice Niemeyer, Werner Nickel, Martin Schoenert
               ##
                     # Johannes Meier, Alex Wegner,
                                                        Thomas Bischops
                    # Frank Celler, Juergen Mnich, Udo Polis
## Thomas Breuer, Goetz Pfeiffer, Hans U. Besche
                        Volkmar Felsch, Heiko Theissen, Alexander Hulpke
               ######
                        Ansgar Kaup, Akos Seress, Erzsebet Horvath
                        Bettina Eick
                        For help enter: ?<return>
gap>
```

4. The interface also has access to the GAP3 help system:

10.4 Common Pitfalls

1. If you want to pass a string to GAP3, then you need to wrap it in single quotes as follows:

```
sage: gap3('"This is a GAP3 string"') #optional - gap3
"This is a GAP3 string"
```

This is particularly important when a GAP3 package is loaded via the $RequirePackage\ method$ (note that one can instead use the load_package method):

```
sage: gap3.RequirePackage('"chevie"') #optional - gap3
```

10.4. Common Pitfalls 63

10.5 Examples

Load a GAP3 package:

```
sage: gap3.load_package("chevie") #optional - gap3
sage: gap3.version() # random #optional - gap3
'lib: v3r4p4 1997/04/18, src: v3r4p0 1994/07/10, sys: usg gcc ansi'
```

Working with GAP3 lists. Note that GAP3 lists are 1-indexed:

```
sage: L = gap3([1,2,3])
                                                         #optional - gap3
sage: L[1]
                                                         #optional - gap3
sage: L[2]
                                                         #optional - gap3
                                                         #optional - gap3
sage: 3 in L
True
sage: 4 in L
                                                         #optional - gap3
False
sage: m = gap3([[1,2],[3,4]])
                                                         #optional - gap3
                                                         #optional - gap3
sage: m[2,1]
sage: [1,2] in m
                                                         #optional - gap3
sage: [3,2] in m
                                                         #optional - gap3
sage: gap3([1,2]) in m
                                                         #optional - gap3
True
```

Controlling variable names used by GAP3:

```
sage: gap3('2', name='x')
2
sage: gap3('x')  #optional - gap3
2
sage: gap3.unbind('x')  #optional - gap3
sage: gap3('x')  #optional - gap3
Traceback (most recent call last):
...
TypeError: Gap3 produced error output
Error, Variable: 'x' must have a value
...
```

class sage.interfaces.gap3. GAP3Element (parent, value, is_name=False, name=None)
 Bases: sage.interfaces.gap.GapElement_generic

A GAP3 element

Note: If the corresponding GAP3 element is a GAP3 record, then the class is changed to a GAP3Record.

INPUT:

- •parent the GAP3 session
- •value the GAP3 command as a string
- •is name bool (default: False); if True, then value is the variable name for the object

•name – str (default: None); the variable name to use for the object. If None, then a variable name is generated.

Note: If you pass E, X or Z for name, then an error is raised because these are sacred variable names in GAP3 that should never be redefined. Sage raises an error because GAP3 does not!

EXAMPLES:

```
sage: from sage.interfaces.gap3 import GAP3Element #optional - gap3
sage: gap3 = Gap3() #optional - gap3
sage: GAP3Element(gap3, value='3+2') #optional - gap3
5
sage: GAP3Element(gap3, value='sage0', is_name=True) #optional - gap3
5
```

TESTS:

AUTHORS:

•Franco Saliola (Feb 2010)

```
class sage.interfaces.gap3. GAP3Record ( parent, value, is_name=False, name=None)
    Bases: sage.interfaces.gap3.GAP3Element
```

A GAP3 record

Note: This class should not be called directly, use GAP3Element instead. If the corresponding GAP3 element is a GAP3 record, then the class is changed to a GAP3Record.

AUTHORS:

•Franco Saliola (Feb 2010)

operations ()

Return a list of the GAP3 operations for the record.

OUTPUT:

•list of strings - operations of the record

EXAMPLES:

10.5. Examples 65

recfields ()

Return a list of the fields for the record. (Record fields are akin to object attributes in Sage.)

OUTPUT:

•list of strings - the field records

EXAMPLES:

```
sage: S5 = gap3.SymmetricGroup(5) #optional - gap3
sage: S5.recfields() #optional - gap3
['isDomain', 'isGroup', 'identity', 'generators',
  'operations', 'isPermGroup', 'isFinite', '1', '2',
  '3', '4', 'degree']
sage: S5.degree #optional - gap3
```

class sage.interfaces.gap3. Gap3 (command='gap3')

Bases: sage.interfaces.gap.Gap_generic

A simple Expect interface to GAP3.

EXAMPLES:

```
sage: from sage.interfaces.gap3 import Gap3
sage: gap3 = Gap3(command='gap3')
```

TESTS:

```
sage: gap3(2) == gap3(3)  # optional - gap3
False
sage: gap3(2) == gap3(2)  # optional - gap3
True
sage: gap3._tab_completion()  # optional - gap3
[]
```

We test the interface behaves correctly after a keyboard interrupt:

We test that the interface busts out of GAP3's break loop correctly:

```
sage: f = gap3('function(L) return L[0]; end;;') #optional - gap3
sage: f([1,2,3]) #optional - gap3
Traceback (most recent call last):
...
RuntimeError: Gap3 produced error output
Error, List Element: <position> must be a positive integer at return L[0] ...
```

AUTHORS:

•Franco Saliola (Feb 2010)

console ()

Spawn a new GAP3 command-line session.

EXAMPLES:

```
sage: gap3.console()
                                                     #not tested
             #######
                                 Lehrstuhl D fuer Mathematik
                  ####
                                  RWTH Aachen
           ###
                     ##
          ##
                                    #######
                                                        ########
         ##
                      #
        ##
                                        ##
                                                       ###
                                                                 ##
        ##
                                          ##
                                                          #
                                                                 ##
        ####
                    ##
                                  ##
                                          #
                                                          #
                                                                  ##
         #####
                   ###
                                  ##
                                          ##
                                                          ##
                                                                 ##
                                  #########
                                                          #######
           #
                                  Version 3
                                  Release 4.4
                                                          #
                                  18 Apr 97
                     # Alice Niemeyer, Werner Nickel, Martin Schoenert
                ##
                     # Johannes Meier, Alex Wegner, Thomas Bischops
               ##
                    # Frank Celler, Juergen Mnich, Udo Polis
## Thomas Breuer, Goetz Pfeiffer, Hans U. Besche
              ##
              ###
                        Volkmar Felsch, Heiko Theissen, Alexander Hulpke
               #####
                        Ansgar Kaup, Akos Seress,
                                                         Erzsebet Horvath
                        Bettina Eick
                        For help enter: ?<return>
gap>
```

cputime (t=None)

Returns the amount of CPU time that the GAP session has used in seconds. If t is not None, then it returns the difference between the current CPU time and t.

EXAMPLES:

help (topic, pager=True)

Print help on the given topic.

INPUT:

```
•topic - string
```

EXAMPLES:

10.5. Examples 67

```
This section describes together with the following sectio... help system. The help system lets you read the manual inter...
```

```
sage: gap3.help('SymmetricGroup', pager=False) #optional - gap3
no section with this name was found
```

TESTS:

```
sage: m = gap3([[1,2,3],[4,5,6]]); m
                                             #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: gap3.help('help', pager=False)
                                               #optional - gap3
Help _
sage: m
                                               #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: m.Print()
                                               #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: gap3.help('Group', pager=False)
                                               #optional - gap3
Group _
sage: m
                                                #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
sage: m.Print()
                                               #optional - gap3
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
```

sage.interfaces.gap3.gap3_console()

Spawn a new GAP3 command-line session.

EXAMPLES:

```
sage: gap3.console()
                                             #not tested
           #######
                           Lehrstuhl D fuer Mathematik
         ###
               ####
                             RWTH Aachen
        ##
                ##
        ##
                  #
                              #######
                                               #########
                             # ##
       ##
                                              ########
       ##
                                   ##
                                                  #
                                                       ##
                                   #
                            ##
       ####
                ##
                                                  #
                                                       ##
                            ## ##
        #####
                ###
                                                ##
                                                       ##
         ########
                                                #######
                 ##
                           Version 3
                ###
                            Release 4.4
                ####
                            18 Apr 97
               ## #
                  # Alice Niemeyer, Werner Nickel, Martin Schoenert
              ##
                  # Johannes Meier, Alex Wegner,
                                                 Thomas Bischops
                  # Frank Celler, Juergen Mnich, Udo Polis
            ### ## Thomas Breuer, Goetz Pfeiffer, Hans U. Besche
             ###### Volkmar Felsch, Heiko Theissen, Alexander Hulpke
                     Ansgar Kaup, Akos Seress, Erzsebet Horvath
                     Bettina Eick
                     For help enter: ?<return>
gap>
```

sage.interfaces.gap3.gap3_version ()

Return the version of GAP3 that you have in your PATH on your computer.

10.5. Examples 69

CHAPTER

ELEVEN

INTERFACE TO GROEBNER FAN

AUTHOR:

- Anders Nedergaard Jensen: Write gfan C++ program, which implements algorithms many of which were invented by Jensen, Komei Fukuda, and Rekha Thomas.
- William Stein (2006-03-18): wrote gfan interface (first version)
- Marshall Hampton (2008-03-17): modified to use gfan-0.3, subprocess instead of os.popen2

TODO – much functionality of gfan-0.3 is still not exposed:

class sage.interfaces.gfan. Gfan

Interface to Anders Jensen's Groebner Fan program.

CHAPTER

TWELVE

PEXPECT INTERFACE TO GIAC

(You should prefer the cython interface: giacpy_sage and its libgiac command)

(adapted by F. Han from William Stein and Gregg Musiker maple's interface)

You must have the Giac interpreter installed and available as the command \mbox{giac} in your PATH in order to use this interface. You need a giac version supporting "giac –sage" (roughly after 0.9.1). In this case you do not have to install any optional Sage packages. If giac is not already installed, you can download binaries or sources or spkg (follow the sources link) from the homepage:

Homepage http://www-fourier.ujf-grenoble.fr/~parisse/giac.html

Type giac.[tab] for a list of all the functions available from your Giac install. Type giac.[tab]? for Giac's help about a given function. Type giac(...) to create a new Giac object, and giac.eval(...) to run a string using Giac (and get the result back as a string).

If the giac spkg is installed, you should find the full html documentation there:

```
$SAGE_LOCAL/share/giac/doc/en/cascmd_local/index.html
```

EXAMPLES:

```
sage: giac('3 * 5')
sage: giac.eval('ifactor(2005)')
'5*401'
sage: giac.ifactor(2005)
2005
sage: l=giac.ifactors(2005); 1; 1[2]
[5,1,401,1]
401
sage: giac.fsolve('x^2 = \cos(x) + 4', 'x','0..5')
[1.9140206190...
sage: giac.factor('x^5 - y^5')
(x-y) * (x^4+x^3*y+x^2*y^2+x*y^3+y^4)
sage: R.\langle x, y \rangle = QQ[]; f = (x+y)^5; f2 = giac(f); (f-f2).normal()
sage: x, y = \text{giac}('x, y'); \text{ giac.int}(y/(\cos(2*x) + \cos(x)), x)
y*2*((-(tan(x/2)))/6+(-2*1/6/sqrt(3))*ln(abs(6*tan(x/2)-2*sqrt(3))/abs(6*tan(x/2)-2*sqrt(3))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))/abs(6*tan(x/2))
  \hookrightarrow2)+2*sqrt(3))))
```

If the string "error" (case insensitive) occurs in the output of anything from Giac, a RuntimeError exception is raised.

12.1 Tutorial

AUTHORS:

• Gregg Musiker (2006-02-02): initial version.

(adapted to giac by F.Han)

This tutorial is based on the Maple Tutorial for number theory from http://www.math.mun.ca/~drideout/m3370/numtheory.html.

There are several ways to use the Giac Interface in Sage. We will discuss two of those ways in this tutorial.

1. If you have a giac expression such as

```
factor( (x^5-1));
```

We can write that in sage as

```
sage: giac('factor(x^5-1)')
(x-1)*(x^4+x^3+x^2+x+1)
```

Notice, there is no need to use a semicolon.

2. Since Sage is written in Python, we can also import giac commands and write our scripts in a pythonic way. For example, factor() is a giac command, so we can also factor in Sage using

```
sage: giac('(x^5-1)').factor()
(x-1)*(x^4+x^3+x^2+x+1)
```

where expression.command() means the same thing as command(expression) in Giac. We will use this second type of syntax whenever possible, resorting to the first when needed.

```
sage: giac('(x^12-1)/(x-1)').normal()
x^11+x^10+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1
```

The normal command will reduce a rational function to the lowest terms. In giac, simplify is slower than normal because it tries more sophisticated simplifications (ex algebraic extensions) The factor command will factor a polynomial with rational coefficients into irreducible factors over the ring of integers (if your default configuration of giac (cf. .xcasrc) has not allowed square roots). So for example,

```
sage: giac('(x^12-1)').factor()
(x-1)*(x+1)*(x^2+1)*(x^2-x+1)*(x^2+x+1)*(x^4-x^2+1)
```

```
sage: giac('(x^28-1)').factor() (x-1)*(x+1)*(x^2+1)*(x^6-x^5+x^4-x^3+x^2-x+1)*(x^6+x^5+x^4+x^3+x^2+x+1)*(x^12-x^10+x^4+x^6+x^6+x^4-x^2+1)
```

Another important feature of giac is its online help. We can access this through sage as well. After reading the description of the command, you can press q to immediately get back to your original prompt.

Incidentally you can always get into a giac console by the command.

```
sage: giac.console()  # not tested
sage: !giac  # not tested
```

Note that the above two commands are slightly different, and the first is preferred.

For example, for help on the giac command factors, we type

```
sage: giac.help('factors') # not tested
```

```
sage: alpha = giac((1+sqrt(5))/2)
sage: beta = giac(1-sqrt(5))/2
sage: f19 = alpha^19 - beta^19/sqrt(5)
sage: f19
(sqrt(5)/2+1/2)^19-((-sqrt(5)+1)/2)^19/sqrt(5)
sage: (f19-(5778*sqrt(5)+33825)/5).normal()
0
```

Let's say we want to write a giac program now that squares a number if it is positive and cubes it if it is negative. In giac, that would look like

```
mysqcu := proc(x)
if x > 0 then x^2;
else x^3; fi;
end;
```

In Sage, we write

```
sage: mysqcu = giac('proc(x) if x > 0 then x^2 else x^3 fi end')
sage: mysqcu(5)
25
sage: mysqcu(-5)
-125
```

More complicated programs should be put in a separate file and loaded.

Bases: sage.interfaces.expect.Expect

Interface to the Giac interpreter.

You must have the optional Giac interpreter installed and available as the command giac in your PATH in order to use this interface. Try the command: print(giac_install_hints()) for more informations on giac installation.

Type giac. [tab] for a list of all the functions available from your Giac install. Type giac. [tab]? for Giac's help about a given function. Type giac(...) to create a new Giac object.

Full html documentation for giac is avaible from your giac installation at \$PREFIX /share/giac/doc/en/cascmd_en/index.html

EXAMPLES:

Any Giac instruction can be evaluated as a string by the giac command. You can access the giac functions by adding the giac. prefix to the usual Giac name.

```
sage: l=giac('normal((y+sqrt(2))^4)'); 1
y^4+4*sqrt(2)*y^3+12*y^2+8*sqrt(2)*y+4
sage: f=giac('(u,v)->{ if (u<v){ [u,v] } else { [v,u] }}');f(1,2),f(3,1)
([1,2], [1,3])</pre>
```

The output of the giac command is a Giac object, and it can be used for another giac command.

```
sage: 1.factors()
[y+sqrt(2),4]
sage: giac('(x^12-1)').factor()
(x-1)*(x+1)*(x^2+1)*(x^2-x+1)*(x^2+x+1)*(x^4-x^2+1)
```

12.1. Tutorial 75

```
sage: giac('assume(y>0)'); giac('y^2=3').solve('y')
y
...[sqrt(3)]
```

You can create some Giac elements and avoid many quotes like this:

Polynomials or elements of SR can be evaluated directly by the giac interface.

```
sage: R.<a,b>=QQ[];f=(2+a+b);p=giac.gcd(f^3+5*f^5,f^2+f^5);p;R(p);
a^2+2*a*b+4*a+b^2+4*b+4
a^2 + 2*a*b + b^2 + 4*a + 4*b + 4
```

Variable names in python and giac are independent.

```
sage: a=sqrt(2);giac('Digits:=30;a:=5');a,giac('a'),giac(a),giac(a).evalf()
30
(sqrt(2), 5, sqrt(2), 1.41421356237309504880168872421)
```

clear (var)

Clear the variable named var.

EXAMPLES:

```
sage: giac.set('xx', '2')
sage: giac.get('xx')
'2'
sage: giac.clear('xx')
sage: giac.get('xx')
```

completions (s)

Return all commands that complete the command starting with the string s.

EXAMPLES:

```
sage: c = giac.completions('cas')
sage: 'cas_setup' in c
True
```

console ()

Spawn a new Giac command-line session.

```
sage: giac_console() # not tested - giac
...
Homepage http://www-fourier.ujf-grenoble.fr/~parisse/giac.html
Released under the GPL license 3.0 or above
See http://www.gnu.org for license details
```

```
Press CTRL and D simultaneously to finish session
Type ?commandname for help
0>>
```

cputime (t=None)

Returns the amount of CPU time that the Giac session has used. If t is not None, then it returns the difference between the current CPU time and t.

EXAMPLES:

eval (code, strip=True, **kwds)

Send the code x to the Giac interpreter. Remark: To enable multi-lines codes in the notebook magic mode: giac, the n are removed before sending the code to giac.

INPUT:

```
•code - str
```

•strip – Default is True and removes \n

EXAMPLES:

```
sage: giac.eval("2+2;\n3")
'4,3'
sage: giac.eval("2+2;\n3",False)
'4\n3'
sage: s='g(x):={\nx+1;\nx+2;\n}'
sage: giac(s)
(x)->{
x+1;
x+2;
}
sage: giac.g(5)
7
```

expect ()

Returns the pexpect object for this Giac session.

EXAMPLES:

```
sage: m = Giac()
sage: m.expect() is None
True
sage: m._start()
sage: m.expect()
Giac with PID ... running .../giac --sage
sage: m.quit()
```

get (var)

Get the value of the variable var.

12.1. Tutorial 77

EXAMPLES:

```
sage: giac.set('xx', '2')
sage: giac.get('xx')
'2'
```

help (str)

Display Giac help about str. This is the same as typing "?str" in the Giac console.

INPUT:

•str - a string to search for in the giac help system

EXAMPLES:

```
sage: giac.help('Psi')  # not tested - depends of giac and $LANG
Psi(a,n)=nth-derivative of the function DiGamma (=ln@Gamma) at point a_
\hookrightarrow (Psi(a,0)=Psi(a))...
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: giac.set('xx', '2')
sage: giac.get('xx')
'2'
```

version ()

Wrapper for giac's version().

EXAMPLES:

```
sage: giac.version()
"giac...
```

class sage.interfaces.giac. GiacElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.expect. Expect Element

```
integral (var='x', min=None, max=None)
```

Return the integral of self with respect to the variable x.

INPUT:

```
•var - variable
```

•min - default: None

•max - default: None

Returns the definite integral if xmin is not None, otherwise returns an indefinite integral.

```
sage: y=giac('y');f=(sin(2*y)/y).integral(y).simplify(); f
Si(2*y)
sage: f.diff(y).simplify()
sin(2*y)/y
```

```
sage: f = giac('exp(x^2)').integral('x',0,1) ; f 1.46265174...
```

```
sage: x,y=giac('x'),giac('y');integrate(cos(x+y),'x=0..pi').simplify()
-2*sin(y)
```

integrate (var='x', min=None, max=None)

Return the integral of self with respect to the variable x.

INPUT:

var - variablemin - default: Nonemax - default: None

Returns the definite integral if xmin is not None, otherwise returns an indefinite integral.

EXAMPLES:

```
sage: y=giac('y');f=(sin(2*y)/y).integral(y).simplify(); f
Si(2*y)
sage: f.diff(y).simplify()
sin(2*y)/y
```

```
sage: f = giac('exp(x^2)').integral('x',0,1) ; f
1.46265174...
sage: x,y=giac('x'),giac('y');integrate(cos(x+y),'x=0..pi').simplify()
-2*sin(y)
```

sum (var, min=None, max=None)

Return the sum of self with respect to the variable x.

INPUT:

var - variablemin - default: Nonemax - default: None

Returns the definite integral if xmin is not None, otherwise returns an indefinite integral.

EXAMPLES:

```
sage: giac('1/(1+k^2)').sum('k',-oo,+infinity).simplify()
(pi*exp(pi)^2+pi)/(exp(pi)^2-1)
```

unapply (var)

Creates a Giac function in the given arguments from a Giac symbol.

EXAMPLES:

```
sage: f=giac('y^3+1+t')
sage: g=(f.unapply('y,t'))
sage: g
(y,t)->y^3+1+t
sage: g(1,2)
4
```

class sage.interfaces.giac. GiacFunction (parent, name)

Bases: sage.interfaces.expect.ExpectFunction

12.1. Tutorial 79

EXAMPLES:

sage.interfaces.giac.reduce_load_Giac()

Returns the giac object created in sage.interfaces.giac.

```
sage: from sage.interfaces.giac import reduce_load_Giac
sage: reduce_load_Giac()
Giac
```

INTERFACE TO THE GNUPLOT INTERPRETER

```
class sage.interfaces.gnuplot. Gnuplot
     Bases: sage.structure.sage_object.SageObject
     Interface to the Gnuplot interpreter.
     console ()
     gnuplot ()
     interact ( cmd)
     plot ( cmd, file=None, verbose=True, reset=True)
           Draw the plot described by cmd, and possibly also save to an eps or png file.
           INPUT:
              •cmd - string
              •file - string (default: None), if specified save plot to given file, which may be either an eps (default)
               or png file.
              •verbose - print some info
              •reset - True: reset gnuplot before making graph
           OUTPUT: displays graph
           Note: Note that ^{\land} s are replaced by ** s before being passed to gnuplot.
     plot3d (f, xmin=-1, xmax=1, ymin=-1, ymax=1, zmin=-1, zmax=1, title=None, samples=25, isosam-1
                ples=20, xlabel='x', ylabel='y', interact=True)
     plot3d_parametric (f = \cos(u)*(3 + v*\cos(u/2)), \sin(u)*(3 + v*\cos(u/2)),
                                                                                              v*sin(u/2)',
                                range1='[u=-pi:pi]', range2='[v=-0.2:0.2]', samples=50, title=None,
                                interact=True)
           Draw a parametric 3d surface and rotate it interactively.
           INPUT:
              •f - (string) a function of two variables, e.g., \cos(u)*(3 + v*\cos(u/2)), \sin(u)*(3 + v*\cos(u/2)),
               v*sin(u/2)
              •range1 - (string) range of values for one variable, e.g., '[u=-pi:pi]'
              •range2 - (string) range of values for another variable, e.g., '[v=-0.2:0.2]'
              •samples - (int) number of sample points to use
              •title - (string) title of the graph.
```

EXAMPLES:

sage.interfaces.gnuplot.gnuplot_console ()

CHAPTER

FOURTEEN

INTERFACE TO THE GP CALCULATOR OF PARI/GP

Type gp. [tab] for a list of all the functions available from your Gp install. Type gp. [tab]? for Gp's help about a given function. Type gp(...) to create a new Gp object, and gp.eval(...) to evaluate a string using Gp (and get the result back as a string).

EXAMPLES: We illustrate objects that wrap GP objects (gp is the PARI interpreter):

```
sage: M = gp('[1,2;3,4]')
sage: M
[1, 2; 3, 4]
sage: M * M
[7, 10; 15, 22]
sage: M + M
[2, 4; 6, 8]
sage: M.matdet()
-2
```

```
sage: E = gp.ellinit([1,2,3,4,5])
sage: E.ellglobalred()
[10351, [1, -1, 0, -1], 1, [11, 1; 941, 1], [[1, 5, 0, 1], [1, 5, 0, 1]]]
sage: E.ellan(20)
[1, 1, 0, -1, -3, 0, -1, -3, -3, -1, 0, 1, -1, 0, -1, 5, -3, 4, 3]
```

```
sage: primitive_root(7)
3
sage: x = gp("znlog( Mod(2,7), Mod(3,7))")
sage: 3^x % 7
2
```

GP has a powerful very efficient algorithm for numerical computation of integrals.

```
sage: gp("a = intnum(x=0,6,sin(x))")
0.03982971334963397945434770208  # 32-bit
0.039829713349633979454347702077075594548 # 64-bit
sage: gp("a")
0.03982971334963397945434770208  # 32-bit
0.039829713349633979454347702077075594548 # 64-bit
sage: gp.kill("a")
sage: gp("a")
a
```

Note that gp ASCII plots do work in Sage, as follows:

The GP interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

```
sage: t = '"%s"'%10^10000 # ten thousand character string.
sage: a = gp.eval(t)
sage: a = gp(t)
```

In Sage, the PARI large Galois groups datafiles should be installed by default:

```
sage: f = gp('x^9 - x - 2')
sage: f.polgalois()
[362880, -1, 34, "S9"]
```

TESTS:

Test error recovery:

```
sage: x = gp('1/0')
Traceback (most recent call last):
...

TypeError: Error executing code in GP:
CODE:
    sage[...]=1/0;
PARI/GP ERROR:
    *** at top-level: sage[...]=1/0
    ***
    *** _/_: impossible inverse in gdiv: 0.
```

AUTHORS:

• William Stein

- David Joyner: some examples
- William Stein (2006-03-01): added tab completion for methods: gp.[tab] and x = gp(blah); x.[tab]
- William Stein (2006-03-01): updated to work with PARI 2.2.12-beta
- William Stein (2006-05-17): updated to work with PARI 2.2.13-beta

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect

Interface to the PARI gp interpreter.

Type gp.[tab] for a list of all the functions available from your Gp install. Type gp.[tab]? for Gp's help about a given function. Type gp(...) to create a new Gp object, and gp.eval(...) to evaluate a string using Gp (and get the result back as a string).

INPUT:

- •stacksize (int, default 10000000) the initial PARI stacksize in bytes (default 10MB)
- \bullet script_subdirectory (string, default None) name of the subdirectory of SAGE_EXTCODE/pari from which to read scripts
- •logfile (string, default None) log file for the pexpect interface
- •server name of remote server
- •server_tmpdir name of temporary directory on remote server
- •init_list_length (int, default 1024) length of initial list of local variables.
- •seed (int, default random) random number generator seed for pari

EXAMPLES:

```
sage: Gp()
PARI/GP interpreter
```

console ()

Spawn a new GP command-line session.

EXAMPLES:

```
sage: gp.console() # not tested
GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)
amd64 running linux (x86-64/GMP-4.2.1 kernel) 64-bit version
compiled: Jul 21 2010, gcc-4.6.0 20100705 (experimental) (GCC)
(readline v6.0 enabled, extended help enabled)
```

cputime (t=None)

cputime for pari - cputime since the pari process was started.

INPUT:

•t - (default: None); if not None, then returns time since t

Warning: If you call gettime explicitly, e.g., gp.eval('gettime'), you will throw off this clock.

```
sage: gp.cputime()  # random output
0.008000000000000000002
sage: gp.factor('2^157-1')
[852133201, 1; 60726444167, 1; 1654058017289, 1; 2134387368610417, 1]
sage: gp.cputime()  # random output
0.269000000000000000
```

get (var)

Get the value of the GP variable var.

INPUT:

•var (string) – a valid GP variable identifier

EXAMPLES:

```
sage: gp.set('x', '2')
sage: gp.get('x')
'2'
```

get_default (var)

Return the current value of a PARI gp configuration variable.

INPUT:

•var (string) - the name of a PARI gp configuration variable. (See gp.default() for a list.)

OUTPUT:

(string) the value of the variable.

EXAMPLES:

```
sage: gp.get_default('log')
0
sage: gp.get_default('datadir')
'.../share/pari'
sage: gp.get_default('seriesprecision')
16
sage: gp.get_default('realprecision')
28  # 32-bit
38  # 64-bit
```

get_precision()

Return the current PARI precision for real number computations.

EXAMPLES:

```
sage: gp.get_precision()
28  # 32-bit
38  # 64-bit
```

get_real_precision ()

Return the current PARI precision for real number computations.

```
sage: gp.get_precision()
28  # 32-bit
38  # 64-bit
```

get_series_precision()

Return the current PARI power series precision.

EXAMPLES:

```
sage: gp.get_series_precision()
16
```

help (command)

Returns GP's help for command.

EXAMPLES:

```
sage: gp.help('gcd')
'gcd(x,{y}): greatest common divisor of x and y.'
```

kill (var)

Kill the value of the GP variable var.

INPUT:

•var (string) - a valid GP variable identifier

EXAMPLES:

```
sage: gp.set('xx', '22')
sage: gp.get('xx')
'22'
sage: gp.kill('xx')
sage: gp.get('xx')
```

new_with_bits_prec (s, precision=0)

Creates a GP object from s with precision bits of precision. GP actually automatically increases this precision to the nearest word (i.e. the next multiple of 32 on a 32-bit machine, or the next multiple of 64 on a 64-bit machine).

```
sage: pi_def = gp(pi); pi_def
3.141592653589793238462643383
                                                 # 32-bit
3.1415926535897932384626433832795028842
                                                # 64-bit
sage: pi_def.precision()
28
                                                # 32-bit
                                                 # 64-bit
sage: pi_150 = gp.new_with_bits_prec(pi, 150)
sage: new_prec = pi_150.precision(); new_prec
48
                                                # 32-bit
57
                                                 # 64-bit
sage: old_prec = gp.set_precision(new_prec); old_prec
28
                                                 # 32-bit
38
                                                 # 64-bit
sage: pi_150
3.14159265358979323846264338327950288419716939938 # 32-bit
3.14159265358979323846264338327950288419716939937510582098 # 64-bit
sage: gp.set_precision(old_prec)
48
                                                 # 32-bit
57
                                                 # 64-bit
sage: gp.get_precision()
```

```
28 # 32-bit
38 # 64-bit
```

set (var, value)

Set the GP variable var to the given value.

INPUT:

- •var (string) a valid GP variable identifier
- •value a value for the variable

EXAMPLES:

```
sage: gp.set('x', '2')
sage: gp.get('x')
'2'
```

set_default (var, value)

Set a PARI gp configuration variable, and return the old value.

INPUT:

- •var (string) the name of a PARI gp configuration variable. (See gp.default() for a list.)
- •value the value to set the variable to.

EXAMPLES:

```
sage: old_prec = gp.set_default('realprecision', 110)
sage: gp.get_default('realprecision')
115
sage: gp.set_default('realprecision', old_prec)
115
sage: gp.get_default('realprecision')
28  # 32-bit
38  # 64-bit
```

set precision (prec)

Sets the PARI precision (in decimal digits) for real computations, and returns the old value.

Note: PARI/GP rounds up precisions to the nearest machine word, so the result of <code>get_precision()</code> is not always the same as the last value inputted to <code>set_precision()</code>.

EXAMPLES:

set_real_precision (prec)

Sets the PARI precision (in decimal digits) for real computations, and returns the old value.

Note: PARI/GP rounds up precisions to the nearest machine word, so the result of get_precision() is not always the same as the last value inputted to set_precision().

EXAMPLES:

set_seed (seed=None)

Sets the seed for gp interpeter. The seed should be an integer.

EXAMPLES:

```
sage: g = Gp()
sage: g.set_seed(1)
1
sage: [g.random() for i in range(5)]
[1546275796, 879788114, 1745191708, 771966234, 1247963869]
```

set_series_precision (prec=None)

Sets the PARI power series precision, and returns the old precision.

EXAMPLES:

```
sage: old_prec = gp.set_series_precision(50); old_prec
16
sage: gp.get_series_precision()
50
sage: gp.set_series_precision(old_prec)
50
sage: gp.get_series_precision()
16
```

version ()

Returns the version of GP being used.

EXAMPLES:

```
sage: gp.version() # not tested
((2, 4, 3), 'GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)')
```

class sage.interfaces.gp. GpElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.expect.ExpectElement

EXAMPLES: This example illustrates dumping and loading GP elements to compressed strings.

```
sage: a = gp(39393)
sage: loads(a.dumps()) == a
True
```

Since dumping and loading uses the string representation of the object, it need not result in an identical object from the point of view of PARI:

```
sage: E = gp('ellinit([1,2,3,4,5])')
sage: loads(dumps(E)) == E
True
sage: x = gp.Pi()/3
sage: loads(dumps(x)) == x
False
sage: x
1.047197551196597746154214461  # 32-bit
1.0471975511965977461542144610931676281 # 64-bit
sage: loads(dumps(x))
1.047197551196597746154214461  # 32-bit
1.047197551196597746154214461  # 32-bit
1.047197551196597746154214461  # 32-bit
```

The two elliptic curves look the same, but internally the floating point numbers are slightly different.

bool ()

EXAMPLES:

```
sage: gp(2).bool()
True
sage: bool(gp(2))
True
sage: bool(gp(0))
False
```

is_string()

Tell whether this element is a string.

EXAMPLES:

```
sage: gp('"abc"').is_string()
True
sage: gp('[1,2,3]').is_string()
False
```

```
class sage.interfaces.gp. GpFunction ( parent, name)
```

Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.gp. GpFunctionElement (obj, name)

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.gp. gp_console ()

Spawn a new GP command-line session.

EXAMPLES:

```
sage: gp.console() # not tested
GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)
amd64 running linux (x86-64/GMP-4.2.1 kernel) 64-bit version
compiled: Jul 21 2010, gcc-4.6.0 20100705 (experimental) (GCC)
(readline v6.0 enabled, extended help enabled)
```

sage.interfaces.gp. gp_version ()

```
sage: gp.version() # not tested
((2, 4, 3), 'GP/PARI CALCULATOR Version 2.4.3 (development svn-12577)')
```

```
sage.interfaces.gp. is_GpElement (x)
```

Returns True of x is a GpElement.

EXAMPLES:

```
sage: from sage.interfaces.gp import is_GpElement
sage: is_GpElement(gp(2))
True
sage: is_GpElement(2)
False
```

```
sage.interfaces.gp. reduce_load_GP ()
```

Returns the GP interface object defined in sage.interfaces.gp.

```
sage: from sage.interfaces.gp import reduce_load_GP
sage: reduce_load_GP()
PARI/GP interpreter
```

Sage Reference Manual: Interpreter Interfaces, Release 7.5	

INTERFACE FOR EXTRACTING DATA AND GENERATING IMAGES FROM JMOL READABLE FILES.

JmolData is a no GUI version of Jmol useful for extracting data from files Jmol reads and for generating image files. AUTHORS:

- Jonathan Gutow (2012-06-14): complete doctest coverage
- Jonathan Gutow (2012-03-21): initial version

class sage.interfaces.jmoldata. JmolData

Bases: sage.structure.sage_object.SageObject

Todo

Create an animated image file (GIF) if spin is on and put data extracted from a file into a variable/string/structure to return

export_image (targetfile, datafile, datafile_cmd='script', image_type='PNG', figsize=5, **kwds) This executes JmolData.jar to make an image file.

INPUT:

- •targetfile the full path to the file where the image should be written.
- •datafile full path to the data file Jmol can read or text of a script telling Jmol what to read or load. If it is a script and the platform is cygwin, the filenames in the script should be in native windows format.
- •datafile_cmd (default 'script') 'load' or 'script' should be "load" for a data file.
- •image_type (default "PNG") 'PNG' 'JPG' or 'GIF'
- •figsize number (default 5) equal to (pixels/side)/100

OUTPUT:

Image file, .png, .gif or .jpg (default .png)

Note: Examples will generate an error message if a functional Java Virtual Machine (JVM) is not installed on the machine the Sage instance is running on.

Warning: Programmers using this module should check that the JVM is available before making calls to avoid the user getting error messages. Check for the JVM using the function <code>is_jvm_available()</code>, which returns True if a JVM is available.

EXAMPLES:

Use Jmol to load a pdb file containing some DNA from a web data base and make an image of the DNA. If you execute this in the notebook, the image will appear in the output cell:

Use Jmol to save an image of a 3-D object created in Sage. This method is used internally by plot3d to generate static images. This example doesn't have correct scaling:

```
sage: from sage.interfaces.jmoldata import JmolData
sage: JData = JmolData()
sage: D=dodecahedron()
sage: from sage.misc.misc import SAGE_TMP
sage: archive_name=os.path.join(SAGE_TMP, "archive.jmol.zip")
sage: D.export_jmol(archive_name) #not scaled properly...need some more,
\rightarrowsteps.
sage: archive_native = archive_name
sage: import sys
sage: if sys.platform == 'cygwin':
....: from subprocess import check_output, STDOUT
         archive_native = check_output(['cygpath', '-w', archive_native],
. . . . :
                                        stderr=STDOUT).rstrip()
sage: script = 'set defaultdirectory "{0}"\n script SCRIPT\n'.format(archive_
sage: testfile = os.path.join(SAGE_TMP, "testimage.png")
sage: JData.export_image(targetfile=testfile, datafile=script, image_type="PNG
→") # optional -- java
sage: print(os.path.exists(testfile)) # optional -- java
True
```

is_jvm_available()

Returns True if the Java Virtual Machine is available and False if not.

EXAMPLES:

Check that it returns a boolean:

```
sage: from sage.interfaces.jmoldata import JmolData
sage: JData = JmolData()
sage: type(JData.is_jvm_available())
<type 'bool'>
```

CHAPTER

SIXTEEN

INTERFACE TO KASH

Sage provides an interface to the KASH computer algebra system, which is a *free* (as in beer!) but *closed source* program for algebraic number theory that shares much common code with Magma. To use KASH, you must install the appropriate optional Sage package by typing something like "sage -i kash3-linux-2005.11.22" or "sage -i kash3_osx-2005.11.22". For a list of optional packages type "sage -optional". If you type one of the above commands, the (about 16MB) package will be downloaded automatically (you don't have to do that).

It is not enough to just have KASH installed on your computer. Note that the KASH Sage package is currently only available for Linux and OSX. If you need Windows, support contact me (wstein@gmail.com).

The KASH interface offers three pieces of functionality:

- 1. kash_console() A function that dumps you into an interactive command-line KASH session. Alternatively,
 - type ! kash from the Sage prompt.
- 2. kash(expr) Creation of a Sage object that wraps a KASH object. This provides a Pythonic interface to KASH. For example, if f=kash.new(10), then f.Factors() returns the prime factorization of 10 computed using KASH.
- 3. kash.function_name (args ...) Call the indicated KASH function with the given arguments are return the result as a KASH object.
- 4. kash.eval (expr) Evaluation of arbitrary KASH expressions, with the result returned as a string.

16.1 Issues

For some reason hitting Control-C to interrupt a calculation doesn't work correctly. (TODO)

16.2 Tutorial

The examples in this tutorial require that the optional kash package be installed.

16.2.1 Basics

Basic arithmetic is straightforward. First, we obtain the result as a string.

```
sage: kash.eval('(9 - 7) * (5 + 6)') # optional -- kash
'22'
```

Next we obtain the result as a new KASH object.

```
sage: a = kash('(9 - 7) * (5 + 6)'); a  # optional -- kash
22
sage: a.parent()  # optional -- kash
Kash
```

We can do arithmetic and call functions on KASH objects:

16.2.2 Integrated Help

Use the kash.help(name) command to get help about a given command. This returns a list of help for each of the definitions of name. Use print kash.help(name) to nicely print out all signatures.

16.2.3 Arithmetic

Using the kash.new command we create Kash objects on which one can do arithmetic.

16.2.4 Variable assignment

Variable assignment using kash is takes place in Sage.

In particular, a is not defined as part of the KASH session itself.

```
sage: kash.eval('a') # optional -- kash
"Error, the variable 'a' must have a value"
```

Use a . name () to get the name of the KASH variable:

```
sage: a.name() # somewhat random; optional - kash
'sage0'
sage: kash(a.name()) # optional -- kash
32233
```

16.2.5 Integers and Rationals

We illustrate arithmetic with integers and rationals in KASH.

```
sage: F = kash.Factorization(4352)  # optional -- kash
sage: F[1]  # optional -- kash
<2, 8>
sage: F[2]  # optional -- kash
<17, 1>
sage: F  # optional -- kash
[ <2, 8>, <17, 1> ], extended by:
ext1 := 1,
ext2 := Unassign
```

Note: For some very large numbers KASH's integer factorization seems much faster than PARI's (which is the default in Sage).

```
sage: kash.GCD(15,25)
                                                 # optional -- kash
sage: kash.LCM(15,25)
                                                 # optional -- kash
                                                 # optional -- kash
sage: kash.Div(25,15)
                                                 # optional -- kash
sage: kash(17) % kash(5)
                                                 # optional -- kash
sage: kash.IsPrime(10007)
TRUE
                                                 # optional -- kash
sage: kash.IsPrime(2005)
FALSE
sage: kash.NextPrime(10007)
                                                 # optional -- kash
10009
```

16.2.6 Real and Complex Numbers

```
sage: kash.Precision()
                                 # optional -- kash
sage: kash('R')
                                 # optional -- kash
Real field of precision 30
sage: kash.Precision(40)
                                 # optional -- kash
sage: kash('R')
                                 # optional -- kash
Real field of precision 40
sage: z = kash('1 + 2*I')
                                 # optional -- kash
                                 # optional -- kash
# optional -- kash
sage: kash.Cos('1.24')
                                 # optional -- kash
0.3247962844387762365776934156973803996992
sage: kash('1.24').Cos()
                                 # optional -- kash
```

16.2. Tutorial 97

```
0.3247962844387762365776934156973803996992
sage: kash.Exp('1.24')
                                            # optional -- kash
3.455613464762675598057615494121998175400
sage: kash.Precision(30)
                                            # optional -- kash
sage: kash.Log('3+4*I')
                                            # optional -- kash
1.60943791243410037460075933323 + 0.927295218001612232428512462922 * I
sage: kash.Log('I')
                                            # optional -- kash
1.57079632679489661923132169164*I
sage: kash.Sqrt(4)
                                            # optional -- kash
sage: kash.Sqrt(2)
                                            # optional -- kash
1.41421356237309504880168872421
sage: kash.Floor('9/5')
                                            # optional -- kash
sage: kash.Floor('3/5')
                                            # optional -- kash
sage: x_c = kash('3+I')
                                            # optional -- kash
                                            # optional -- kash
sage: x_c.Argument()
0.321750554396642193401404614359
sage: x_c.Imaginary()
                                            # optional -- kash
```

16.2.7 Lists

Note that list appends are completely different in KASH than in Python. Use underscore after the function name for the mutation version.

```
sage: v = kash([1,2,3]); v
                                                # optional -- kash
[ 1, 2, 3 ]
sage: v[1]
                                                # optional -- kash
                                                # optional -- kash
sage: v[3]
sage: v.Append([5])
                                                # optional -- kash
[ 1, 2, 3, 5 ]
sage: v
                                                # optional -- kash
[ 1, 2, 3 ]
sage: v.Append_([5, 6])
                                               # optional -- kash
SUCCESS
                                                # optional -- kash
sage: v
[ 1, 2, 3, 5, 6 ]
sage: v.Add(5)
                                                # optional -- kash
[ 1, 2, 3, 5, 6, 5 ]
sage: v
                                                # optional -- kash
[ 1, 2, 3, 5, 6 ]
sage: v.Add_(5)
                                                # optional -- kash
SUCCESS
sage: v
                                                # optional -- kash
[ 1, 2, 3, 5, 6, 5 ]
```

The Apply command applies a function to each element of a list.

```
:: sage: L = kash([1,2,3,4]) # optional – kash sage: L.Apply('i -> 3*i') # optional – kash [ 3, 6, 9, 12 ] sage: L # optional – kash [ 1, 2, 3, 4 ] sage: L.Apply('IsEven') # optional – kash [ FALSE, TRUE, FALSE, TRUE ] sage: L # optional – kash [ 1, 2, 3, 4 ]
```

16.2.8 Ranges

the following are examples of ranges.

```
sage: L = kash('[1..10]')  # optional -- kash
sage: L
[ 1 .. 10 ]
sage: L = kash('[2,4..100]')  # optional -- kash
sage: L
[ 2, 4 .. 100 ]
```

16.2.9 Sequences

16.2.10 Tuples

16.2.11 Polynomials

```
sage: f = kash('X^3 + X + 1')  # optional -- kash
sage: f + f  # optional -- kash
2*X^3 + 2*X + 2
sage: f * f  # optional -- kash
X^6 + 2*X^4 + 2*X^3 + X^2 + 2*X + 1
sage: f.Evaluate(10)  # optional -- kash
1011
sage: Qx = kash.PolynomialAlgebra('Q')  # optional -- kash
sage: Qx.gen(1)**5 + kash('7/3')  # sage1 below somewhat random; optional -- kash
sage1.1^5 + 7/3
```

16.2.12 Number Fields

We create an equation order.

```
sage: f = kash('X^5 + 4*X^4 - 56*X^2 -16*X + 192') # optional -- kash
sage: OK = f.EquationOrder() # optional -- kash
sage: OK
Equation Order with defining polynomial X^5 + 4*X^4 - 56*X^2 - 16*X + 192 over Z
```

16.2. Tutorial 99

```
NG.3,
_NG.4,
_NG.5
sage: O.Discriminant()
                                   # optional -- kash
1364202618880
sage: 0.MaximalOrder() # name sage2 below somewhat random; optional -- kash
Maximal Order of sage2
sage: 0 = kash.MaximalOrder('X^3 - 77')
                                                       # optional -- kash
sage: I = 0.Ideal(5,[2, 1, 0])
                                                        # optional -- kash
                         # name sage14 below random; optional -- kash
sage: I
Ideal of sage14
Two element generators:
[5, 0, 0]
[2, 1, 0]
                                            # optional -- kash
sage: F = I.Factorisation()
           # name sage14 random; optional -- kash
sage: F
<Prime Ideal of sage14
Two element generators:
[5, 0, 0]
[2, 1, 0], 1>
```

Determining whether an ideal is principal.

```
sage: I.IsPrincipal() # optional -- kash
FALSE, extended by:
ext1 := Unassign
```

Computation of class groups and unit groups:

```
sage: f = kash('X^5 + 4*X^4 - 56*X^2 -16*X + 192')  # optional -- kash
sage: O = kash.EquationOrder(f)  # optional -- kash
sage: OK = O.MaximalOrder()  # optional -- kash
sage: OK.ClassGroup()  # name sage32 below random; optional -- kash
Abelian Group isomorphic to Z/6
   Defined on 1 generator
   Relations:
   6*sage32.1 = 0, extended by:
   ext1 := Mapping from: grp^abl: sage32 to ids/ord^num: _AA
```

16.2.13 Function Fields

```
sage: k = kash.FiniteField(25)  # optional -- kash
sage: kT = k.RationalFunctionField()  # optional -- kash
sage: kTy = kT.PolynomialAlgebra()  # optional -- kash
sage: T = kT.gen(1)  # optional -- kash
sage: y = kTy.gen(1)  # optional -- kash
sage: f = y**3 + T**4 + 1  # optional -- kash
```

16.3 Long Input

The KASH interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using kash.eval for long input is much less robust, and is not recommended.

```
sage: a = kash(range(10000)) # optional -- kash
```

Note that KASH seems to not support string or integer literals with more than 1024 digits, which is why the above example uses a list unlike for the other interfaces.

```
 \begin{array}{c} \textbf{class} \text{ sage.interfaces.kash. Kash } (\textit{max\_workspace\_size=None}, & \textit{maxread=None}, \\ \textit{script\_subdirectory=None}, & \textit{restart\_on\_ctrlc=True}, & log-\\ \textit{file=None}, \textit{server=None}, \textit{server\_tmpdir=None}) \\ \textbf{Bases: } \textit{sage.interfaces.expect.Expect} \end{array}
```

Interface to the Kash interpreter.

AUTHORS:

•William Stein and David Joyner

```
console ()
```

```
eval ( x, newlines=False, strip=True, **kwds)
```

Send the code in the string s to the Kash interpreter and return the output as a string.

INPUT:

- •s string containing Kash code.
- •newlines bool (default: True); if False, remove all backslash-newlines inserted by the Kash output formatter.

•strip -ignored

get (var)

Get the value of the variable var.

help (name=None)

Return help on KASH commands.

Returns help on all commands with a given name. If name is None, return the location of the installed Kash HTML documentation.

EXAMPLES:

16.3. Long Input 101

```
sage: X = kash.help('IntegerRing') # optional -- kash
```

There is one entry in X for each item found in the documentation for this function: If you type print (X[0]) you will get help on about the first one, printed nicely to the screen.

AUTHORS:

•Sebastion Pauli (2006-02-04): during Sage coding sprint

```
help_search (name)
set (var, value)
    Set the variable var to the given value.

version ()

class sage.interfaces.kash. KashDocumentation
    Bases: list

class sage.interfaces.kash. KashElement (parent, value, is_name=False, name=None)
    Bases: sage.interfaces.expect.ExpectElement

sage.interfaces.kash. is_KashElement (x)

sage.interfaces.kash. kash_console ()

sage.interfaces.kash. kash_version ()

sage.interfaces.kash. reduce_load_Kash ()
```

CHAPTER

SEVENTEEN

INTERFACE TO LIE

LiE is a software package under development at CWI since January 1988. Its purpose is to enable mathematicians and physicists to obtain on-line information as well as to interactively perform computations of a Lie group theoretic nature. It focuses on the representation theory of complex semisimple (reductive) Lie groups and algebras, and on the structure of their Weyl groups and root systems.

Type lie.[tab] for a list of all the functions available from your LiE install. Type lie.[tab]? for LiE's help about a given function. Type lie(...) to create a new LiE object, and lie.eval(...) to run a string using LiE (and get the result back as a string).

To access the LiE interpreter directly, run lie_console().

EXAMPLES:

```
sage: a4 = lie('A4') # optional - lie
sage: lie.diagram('A4')
                        # optional - lie
0---0---0
  2 3 4
1
Α4
sage: lie.diagram(a4)
                             # optional - lie
0---0---0
  2 3 4
A 4
sage: a4.diagram()
                             # optional - lie
0---0---0
1 2 3 4
sage: a4.Cartan()
                              # optional - lie
    [[2,-1,0,0]
    ,[-1, 2,-1, 0]
    ,[ 0,-1, 2,-1]
    ,[0,0,-1,2]
sage: lie.LR_tensor([3,1],[2,2]) # optional - lie
1X[5,3]
```

17.1 Tutorial

The following examples are taken from Section 2.1 of the LiE manual.

You can perform basic arithmetic operations in LiE.

Vectors in LiE are created using square brackets. Notice that the indexing in LiE is 1-based, unlike Python/Sage which is 0-based.

```
sage: v = lie('[3,2,6873,-38]') # optional - lie
sage: v # optional - lie
[3,2,6873,-38]
sage: v[3] # optional - lie
6873
sage: v+v # optional - lie
[6,4,13746,-76]
sage: v*v # optional - lie
47239586
sage: v+234786 # optional - lie
[3,2,6873,-38,234786]
sage: v-3 # optional - lie
[3,2,-38]
sage: v^v # optional - lie
[3,2,6873,-38,3,2,6873,-38]
```

You can also work with matrices in LiE.

```
sage: m = lie('[[1,0,3,3],[12,4,-4,7],[-1,9,8,0],[3,-5,-2,9]]') # optional - lie
sage: m # optional - lie
     [[ 1, 0, 3,3]
     ,[12, 4,-4,7]
     ,[-1, 9, 8,0]
     , [3, -5, -2, 9]
sage: print(lie.eval('*'+m._name)) # optional - lie
     [[1, 12, -1, 3]
     ,[0, 4, 9,-5]
     ,[3,-4, 8,-2]
     ,[3, 7, 0, 9]
sage: m^3 # optional - lie
     [[ 220, 87, 81, 375]
     ,[-168,-1089, 13,1013]
     ,[1550, 357,-55,1593]
     ,[-854, -652, 98,-170]
     1
sage: v*m # optional - lie
[-6960, 62055, 55061, -319]
sage: m*v # optional - lie
[20508, -27714, 54999, -14089]
```

```
sage: v*m*v # optional - lie
378549605
sage: m+v # optional - lie
   [[ 1, 0,  3,  3]
   ,[12, 4, -4,  7]
   ,[-1, 9,  8,  0]
   ,[ 3, -5, -2,  9]
   ,[ 3, 2,6873, -38]
   ]

sage: m-2 # optional - lie
   [[ 1, 0, 3, 3]
   ,[-1, 9, 8, 0]
   ,[ 3, -5, -2, 9]
   ]
}
```

LiE handles multivariate (Laurent) polynomials.

```
sage: lie('X[1,2]') # optional - lie
1X[1,2]
sage: -3*_ # optional - lie
-3X[1,2]
sage: _ + lie('4X[-1,4]') # optional - lie
4X[-1,4] - 3X[ 1,2]
sage: _^2 # optional - lie
16X[-2,8] - 24X[ 0,6] + 9X[ 2,4]
sage: lie('(4X[-1,4]-3X[1,2])*(X[2,0]-X[0,-4])') # optional - lie
-4X[-1, 0] + 3X[ 1,-2] + 4X[ 1, 4] - 3X[ 3, 2]
sage: _ - _ # optional - lie
0X[0,0]
```

You can call LiE's built-in functions using lie.functionname.

```
sage: lie.partitions(6) # optional - lie
     [[6,0,0,0,0,0]
    , [5,1,0,0,0,0]
    ,[4,2,0,0,0,0]
    ,[4,1,1,0,0,0]
    ,[3,3,0,0,0,0]
    ,[3,2,1,0,0,0]
    ,[3,1,1,1,0,0]
    ,[2,2,2,0,0,0]
    ,[2,2,1,1,0,0]
    ,[2,1,1,1,1,0]
    ,[1,1,1,1,1,1]
sage: lie.diagram('E8') # optional - lie
       0 2
   -0---0---0
      4 5
               6
                  7
   3
F.8
```

You can define your own functions in LiE using lie.eval . Once you've defined a function (say f), you can call it using lie.f; however, user-defined functions do not show up when using tab-completion.

17.1. Tutorial 105

LiE's help can be accessed through lie.help('functionname') where functionname is the function you want to receive help for.

```
sage: print(lie.help('diagram')) # optional - lie
diagram(g). Prints the Dynkin diagram of g, also indicating
  the type of each simple component printed, and labeling the nodes as
  done by Bourbaki (for the second and further simple components the
  labels are given an offset so as to make them disjoint from earlier
  labels). The labeling of the vertices of the Dynkin diagram prescribes
  the order of the coordinates of root- and weight vectors used in LiE.
```

This can also be accessed with lie.functionname? .

With the exception of groups, all LiE data types can be converted into native Sage data types by calling the .sage() method.

Integers:

```
sage: a = lie('1234') # optional - lie
sage: b = a.sage(); b # optional - lie
1234
sage: type(b) # optional - lie
<type 'sage.rings.integer.Integer'>
```

Vectors:

```
sage: a = lie('[1,2,3]') # optional - lie
sage: b = a.sage(); b # optional - lie
[1, 2, 3]
sage: type(b) # optional - lie
<... 'list'>
```

Matrices:

```
sage: a = lie('[[1,2],[3,4]]') # optional - lie
sage: b = a.sage(); b # optional - lie
[1 2]
[3 4]
sage: type(b) # optional - lie
<type 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
```

Polynomials:

```
sage: a = lie('X[1,2] - 2*X[2,1]') # optional - lie
sage: b = a.sage(); b # optional - lie
-2*x0^2*x1 + x0*x1^2
```

```
sage: type(b) # optional - lie
<type 'sage.rings.polynomial.multi_polynomial_libsingular.MPolynomial_libsingular'>
```

Text:

```
sage: a = lie('"text"') # optional - lie
sage: b = a.sage(); b # optional - lie
'text'
sage: type(b) # optional - lie
<type 'str'>
```

LiE can be programmed using the Sage interface as well. Section 5.1.5 of the manual gives an example of a function written in LiE's language which evaluates a polynomial at a point. Below is a (roughly) direct translation of that program into Python / Sage.

```
sage: def eval_pol(p, pt): # optional - lie
          s = 0
          for i in range(1,p.length().sage()+1):
. . .
              m = 1
. . .
              for j in range(1,pt.size().sage()+1):
                 m \neq pt[j]^p.expon(i)[j]
              s += p.coef(i)*m
          return s
sage: a = lie('X[1,2]') # optional - lie
sage: b1 = lie('[1,2]') # optional - lie
sage: b2 = lie('[2,3]') # optional - lie
sage: eval_pol(a, b1) # optional - lie
sage: eval_pol(a, b2) # optional - lie
18
```

AUTHORS:

- Mike Hansen 2007-08-27
- William Stein (template)

Interface to the LiE interpreter.

Type lie. [tab] for a list of all the functions available from your LiE install. Type lie. [tab]? for LiE's help about a given function. Type lie(...) to create a new LiE object, and lie.eval(...) to run a string using LiE (and get the result back as a string).

```
console ()
```

Spawn a new LiE command-line session.

EXAMPLES:

```
sage: lie.console() # not tested
LiE version 2.2.2 created on Sep 26 2007 at 18:13:19
Authors: Arjeh M. Cohen, Marc van Leeuwen, Bert Lisser.
Free source code distribution
...
```

17.1. Tutorial 107

```
eval ( code, strip=True, **kwds)
```

EXAMPLES:

```
sage: lie.eval('2+2') # optional - lie
'4'
```

function_call (function, args=None, kwds=None)

EXAMPLES:

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: lie.set('x', '2') # optional - lie
sage: lie.get('x') # optional - lie
'2'
```

get_using_file (var)

EXAMPLES:

```
sage: lie.get_using_file('x')
Traceback (most recent call last):
...
NotImplementedError
```

help (command)

Returns a string of the LiE help for command.

EXAMPLES:

```
sage: lie.help('diagram') # optional - lie
'diagram(g)...'
```

read (filename)

EXAMPLES:

```
sage: filename = tmp_filename()
sage: f = open(filename, 'w')
sage: f.write('x = 2\n')
sage: f.close()
sage: lie.read(filename) # optional - lie
sage: lie.get('x') # optional - lie
'2'
sage: import os
sage: os.unlink(filename)
```

set (var, value)

Set the variable var to the given value.

```
sage: lie.set('x', '2') # optional - lie
        sage: lie.get('x')
                                 # optional - lie
         121
    version()
        EXAMPLES:
        sage: lie.version() # optional - lie
         '2.1'
class sage.interfaces.lie. LiEElement (parent, value, is_name=False, name=None)
                        \verb|sage.interfaces.tab_completion.ExtraTabCompletion|\\
    sage.interfaces.expect.ExpectElement
    type ()
        EXAMPLES:
        sage: m = lie('[[1,0,3,3],[12,4,-4,7],[-1,9,8,0],[3,-5,-2,9]]') # optional -__
        sage: m.type() # optional - lie
        'mat'
class sage.interfaces.lie. LiEFunction ( parent, name)
    Bases: sage.interfaces.expect.ExpectFunction
class sage.interfaces.lie. LiEFunctionElement (obj, name)
    Bases: sage.interfaces.expect.FunctionElement
sage.interfaces.lie. is LiEElement (x)
    EXAMPLES:
    sage: from sage.interfaces.lie import is_LiEElement
    sage: 1 = lie(2) # optional - lie
    sage: is_LiEElement(1) # optional - lie
    True
    sage: is_LiEElement(2)
    False
sage.interfaces.lie. lie console ()
    Spawn a new LiE command-line session.
    EXAMPLES:
    sage: from sage.interfaces.lie import lie_console
    sage: lie_console()
                                           # not tested
    LiE version 2.2.2 created on Sep 26 2007 at 18:13:19
    Authors: Arjeh M. Cohen, Marc van Leeuwen, Bert Lisser.
    Free source code distribution
sage.interfaces.lie.lie_version()
    EXAMPLES:
    sage: from sage.interfaces.lie import lie_version
    sage: lie_version() # optional - lie
    '2.1'
sage.interfaces.lie. reduce_load_lie ()
    EXAMPLES:
```

17.1. Tutorial 109

```
sage: from sage.interfaces.lie import reduce_load_lie
sage: reduce_load_lie()
LiE Interpreter
```

CHAPTER

EIGHTEEN

LISP INTERFACE

EXAMPLES:

```
sage: lisp.eval('(* 4 5)')
1201
sage: a = lisp(3); b = lisp(5)
sage: a + b
sage: a * b
sage: a / b
3/5
sage: a - b
sage: a.sin()
0.14112
sage: b.cos()
0.2836622
sage: a.exp()
20.085537
sage: lisp.eval('(+ %s %s)'%(a.name(), b.name()))
181
```

One can define functions and the interface supports object-oriented notation for calling them:

```
sage: lisp.eval('(defun factorial (n) (if (= n 1) 1 (* n (factorial (- n 1)))))')
'FACTORIAL'
sage: lisp('(factorial 10)')
3628800
sage: lisp(10).factorial()
3628800
sage: a = lisp(17)
sage: a.factorial()
355687428096000
```

AUTHORS: – William Stein (first version) – William Stein (2007-06-20): significant improvements.

Bases: sage.interfaces.expect.Expect

```
sage: lisp == loads(dumps(lisp))
True
```

console ()

Spawn a new Lisp command-line session.

EXAMPLES:

```
sage: lisp.console() #not tested
ECL (Embeddable Common-Lisp) ...
Copyright (C) 1984 Taiichi Yuasa and Masami Hagiya
Copyright (C) 1993 Giuseppe Attardi
Copyright (C) 2000 Juan J. Garcia-Ripoll
ECL is free software, and you are welcome to redistribute it
under certain conditions; see file 'Copyright' for details.
Type :h for Help. Top level.
...
```

eval (code, strip=True, **kwds)

EXAMPLES:

```
sage: lisp.eval('(+ 2 2)')
'4'
```

TEST:

Verify that it works when input == output:

```
sage: lisp.eval('2')
'2'
```

function_call (function, args=None, kwds=None)

Calls the Lisp function with given args and kwds. For Lisp functions, the kwds are ignored.

EXAMPLES:

```
sage: lisp.function_call('sin', ['2'])
0.9092974
sage: lisp.sin(2)
0.9092974
```

get (var)

EXAMPLES:

```
sage: lisp.set('x', '2')
sage: lisp.get('x')
'2'
```

help (command)

EXAMPLES:

```
sage: lisp.help('setq')
Traceback (most recent call last):
...
NotImplementedError
```

kill (var)

```
sage: lisp.kill('x')
Traceback (most recent call last):
```

```
...
NotImplementedError
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: lisp.set('x', '2')
sage: lisp.get('x')
'2'
```

TEST:

It must also be possible to eval the variable by name:

```
sage: lisp.eval('x')
'2'
```

version ()

Returns the version of Lisp being used.

EXAMPLES:

```
sage: lisp.version()
'Version information is given by lisp.console().'
```

class sage.interfaces.lisp. LispElement (parent, value, is_name=False, name=None)

Bases: sage.structure.element.RingElement, sage.interfaces.expect.ExpectElement

bool ()

EXAMPLES:

```
sage: lisp(2).bool()
True
sage: lisp(0).bool()
False
sage: bool(lisp(2))
True
```

```
class sage.interfaces.lisp. LispFunction ( parent, name)
```

Bases: sage.interfaces.expect.ExpectFunction

 ${f class}$ sage.interfaces.lisp. LispFunctionElement (obj, name)

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.lisp.is_LispElement (x)

EXAMPLES:

```
sage: from sage.interfaces.lisp import is_LispElement
sage: is_LispElement(lisp(2))
True
sage: is_LispElement(2)
False
```

sage.interfaces.lisp.lisp_console()

Spawn a new Lisp command-line session.

```
sage: lisp.console() #not tested
ECL (Embeddable Common-Lisp) ...
Copyright (C) 1984 Taiichi Yuasa and Masami Hagiya
Copyright (C) 1993 Giuseppe Attardi
Copyright (C) 2000 Juan J. Garcia-Ripoll
ECL is free software, and you are welcome to redistribute it
under certain conditions; see file 'Copyright' for details.
Type :h for Help. Top level.
...
```

sage.interfaces.lisp. reduce_load_Lisp () EXAMPLES:

```
sage: from sage.interfaces.lisp import reduce_load_Lisp
sage: reduce_load_Lisp()
Lisp Interpreter
```

CHAPTER

NINETEEN

INTERFACE TO MACAULAY2

Note: You must have Macaulay2 installed on your computer for this interface to work. Macaulay2 is not included with Sage, but you can obtain it from http://www.math.uiuc.edu/Macaulay2/. Note additional optional Sage packages are required.

Sage provides an interface to the Macaulay2 computational algebra system. This system provides extensive functionality for commutative algebra. You do not have to install any optional packages.

The Macaulay2 interface offers three pieces of functionality:

- Macaulay2_console() A function that dumps you into an interactive command-line Macaulay2 session.
- Macaulay2 (expr) Evaluation of arbitrary Macaulay2 expressions, with the result returned as a string.
- Macaulay2.new(expr) Creation of a Sage object that wraps a Macaulay2 object. This provides a Pythonic interface to Macaulay2. For example, if f=Macaulay2.new(10), then f.gcd(25) returns the GCD of 10 and 25 computed using Macaulay2.

```
sage: print(macaulay2('3/5 + 7/11')) # optional - macaulay2
68
55
sage: f = macaulay2('f = i -> i^3') # optional - macaulay2
                                    # optional - macaulay2
sage: f
                                    # optional - macaulay2
sage: f(5)
125
sage: R = macaulay2('ZZ/5[x,y,z]') # optional - macaulay2
sage: print(R)
                                     # optional - macaulay2
ZZ
--[x..z, Degrees => {3:1}, Heft => {1}, MonomialOrder => {MonomialSize => 32}, ...
→DegreeRank => 1]
                                                         {GRevLex => {3:1} }
                                                         {Position => Up
sage: x = macaulay2('x')
                                    # optional - macaulay2
sage: y = macaulay2('y')
                                    # optional - macaulay2
sage: print((x+y)^5)
                                    # optional - macaulay2
x + y
sage: parent ((x+y)^5)
                                    # optional - macaulay2
Macaulay2
```

```
sage: R = macaulay2('QQ[x,y,z,w]') # optional - macaulay2
sage: f = \text{macaulay2}('x^4 + 2*x*y^3 + x*y^2*w + x*y*z*w + x*y*w^2 + 2*x*z*w^2 + y^4 + ...
\rightarrowy^3*w + 2*y^2*z*w + z^4 + w^4') # optional - macaulay2
                                   # optional - macaulav2
sage: print(f)
                              3
                       2
      3 4
                  4
                                                2
                                                            2.
                                                                      2
x + 2x*y + y + z + x*y + y + y + x*y*z*w + 2y + x*y*w + 2x*z*w + w
sage: g = f * macaulay2('x+y^5') # optional - macaulay2
                                  # optional - macaulay2
sage: print(g.factor())
sage: print(g.factor())
    4     3     4     4     2     3
                                                2
(x + 2x*y + y + z + x*y + y + y + x*y*z*w + 2y + x*y*w + 2x*z*w + w)(y + ...
\hookrightarrow X)
```

AUTHORS:

- Kiran Kedlaya and David Roe (2006-02-05, during Sage coding sprint)
- William Stein (2006-02-09): inclusion in Sage; prompt uses regexp, calling of Macaulay2 functions via call.
- William Stein (2006-02-09): fixed bug in reading from file and improved output cleaning.
- Kiran Kedlaya (2006-02-12): added ring and ideal constructors, list delimiters, is_Macaulay2Element, sage_polystring, __floordiv__, __mod__, __iter__, __len__; stripped extra leading space and trailing newline from output.

TODO:

• get rid of all numbers in output, e.g., in ideal function below.

Interface to the Macaulay2 interpreter.

console ()

Spawn a new M2 command-line session.

EXAMPLES:

```
sage: macaulay2.console() # not tested
Macaulay 2, version 1.1
with packages: Classic, Core, Elimination, IntegralClosure, LLLBases,
→Parsing, PrimaryDecomposition, SchurRings, TangentCone
...
```

cputime (t=None)

EXAMPLES:

```
sage: R = macaulay2("QQ[x,y]")  # optional - macaulay2
sage: x,y = R.gens()  # optional - macaulay2
sage: a = (x+y+1)^20  # optional - macaulay2
sage: macaulay2.cputime()  # optional - macaulay2; random
0.48393700000000001
```

```
eval ( code, strip=True, **kwds)
```

Send the code x to the Macaulay2 interpreter and return the output as a string suitable for input back into Macaulay2, if possible.

INPUT:

```
•code - str
```

•strip – ignored

EXAMPLES:

```
sage: macaulay2.eval("2+2") # optional - macaulay2
4
```

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: macaulay2.set("a", "2") # optional - macaulay2
sage: macaulay2.get("a") # optional - macaulay2
2
```

help(s)

EXAMPLES:

```
sage: macaulay2.help("load") # optional - macaulay2
load -- read Macaulay2 commands

***********************

* "input" -- read Macaulay2 commands and echo

* "notify" -- whether to notify the user when a file is loaded
```

ideal (*gens)

Return the ideal generated by gens.

INPUT:

•gens – list or tuple of Macaulay2 objects (or objects that can be made into Macaulay2 objects via evaluation)

OUTPUT:

the Macaulay2 ideal generated by the given list of gens

EXAMPLES:

```
sage: R2 = macaulay2.ring('QQ', '[x, y]'); R2
                                                         # optional -_
→macaulay2
QQ[x..y, Degrees => {2:1}, Heft => {1}, MonomialOrder => {MonomialSize => 16}
→, DegreeRank => 1]
                                                         \{Lex => 2
                                                         {Position => Up
sage: I = macaulay2.ideal(('y^2 - x^3', 'x - y')); I # optional -_
→macaulay2
         3
ideal (-x + y, x - y)
sage: J = I^3; J.gb().gens().transpose()
                                                         # optional -_
→macaulay2
\{-9\} | y9-3y8+3y7-y6
\{-7\} | xy6-2xy5+xy4-y7+2y6-y5
\{-5\} | x2y3-x2y2-2xy4+2xy3+y5-y4 |
\{-3\} | x3-3x2y+3xy2-y3
```

new_from (type, value)

Returns a new Macaulay2Element of type type constructed from value.

EXAMPLES:

```
sage: 1 = macaulay2.new_from("MutableList", [1,2,3]) # optional - macaulay2
sage: 1
MutableList{...3...}
sage: list(1) # optional - macaulay2
[1, 2, 3]
```

restart ()

Restart Macaulay2 interpreter.

TEST:

```
sage: macaulay2.restart() # optional - macaulay2
```

```
ring ( base_ring='ZZ', vars='[x]', order='Lex')
```

Create a Macaulay2 ring.

INPUT:

- •base_ring base ring (see examples below)
- •vars a tuple or string that defines the variable names
- •order string the monomial order (default: 'Lex')

OUTPUT:

•a Macaulay2 ring (with base ring ZZ)

EXAMPLES:

This is a ring in variables named a through d over the finite field of order 7, with graded reverse lex ordering:

This is a polynomial ring over the rational numbers:

set (var, value)

Set the variable var to the given value.

```
sage: macaulay2.set("a", "2") # optional - macaulay2
sage: macaulay2.get("a") # optional - macaulay2
2
```

use(R)

Use the Macaulay2 ring R.

EXAMPLES:

```
sage: R = macaulay2("QQ[x,y]")
                                                  # optional - macaulay2
sage: P = macaulay2("ZZ/7[symbol x, symbol y]") # optional - macaulay2
sage: macaulay2("x").cls()
                                                  # optional - macaulay2
--[x..y, Degrees => \{2:1\}, Heft => \{1\}, MonomialOrder => \{MonomialSize => 32\}
\hookrightarrow, DegreeRank => 1]
                                                            {GRevLex => {2:1} }
                                                            {Position => Up
sage: macaulay2.use(R)
                                                   # optional - macaulay2
                                                   # optional - macaulay2
sage: macaulay2("x").cls()
QQ[x..y, Degrees => \{2:1\}, Heft => \{1\}, MonomialOrder => \{MonomialSize => 32\}
→, DegreeRank => 1]
                                                            {GRevLex => {2:1} }
                                                            {Position => Up
```

version ()

Returns the version of Macaulay2.

EXAMPLES:

```
sage: macaulay2.version() # optional - macaulay2
(1, 3, 1)
```

```
Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.ExpectElement
```

cls ()

Since class is a keyword in Python, we have to use cls to call Macaulay2's class. In Macaulay2, class corresponds to Sage's notion of parent.

EXAMPLES:

```
sage: macaulay2(ZZ).cls() # optional - macaulay2
Ring
```

dot(x)

EXAMPLES:

```
sage: d = macaulay2.new("MutableHashTable") # optional - macaulay2
sage: d["k"] = 4 # optional - macaulay2
sage: d.dot("k") # optional - macaulay2
4
```

external_string()

repr ()

EXAMPLES:

```
sage: R = \text{macaulay2}("QQ[x,y,z]/(x^3-y^3-z^3)") # optional - macaulay2
sage: x = macaulay2('x')
                                               # optional - macaulay2
                                               # optional - macaulay2
sage: y = macaulay2('y')
sage: print(x+y)
                                               # optional - macaulay2
x + y
sage: print (macaulay2("QQ[x,y,z]"))
                                               # optional - macaulay2
QQ[x..z, Degrees => {3:1}, Heft => {1}, MonomialOrder => {MonomialSize => 32}
→, DegreeRank => 1]
                                                          {GRevLex => {3:1} }
                                                         {Position => Up
sage: print (macaulay2("QQ[x,y,z]/(x+y+z)")) # optional - macaulay2
QQ[x, y, z]
x + y + z
```

sage_polystring()

If this Macaulay2 element is a polynomial, return a string representation of this polynomial that is suitable for evaluation in Python. Thus \star is used for multiplication and $\star\star$ for exponentiation. This function is primarily used internally.

EXAMPLES:

```
sage: R = macaulay2.ring('QQ','(x,y)')  # optional - macaulay2
sage: f = macaulay2('x^3 + 3*y^11 + 5')  # optional - macaulay2
sage: print(f)  # optional - macaulay2

11
x + 3y + 5
sage: f.sage_polystring()  # optional - macaulay2
'x**3+3*y**11+5'
```

sharp(x)

EXAMPLES:

```
sage: a = macaulay2([1,2,3]) # optional - macaulay2
sage: a.sharp(0) # optional - macaulay2
1
```

starstar (x)

The binary operator ★★ in Macaulay2 is usually used for tensor or Cartesian power.

EXAMPLES:

```
sage: a = macaulay2([1,2]).set() # optional - macaulay2
sage: a.starstar(a) # optional - macaulay2
set {(1, 1), (1, 2), (2, 1), (2, 2)}
```

structure_sheaf ()

```
sage: S = macaulay2('QQ[a..d]') # optional - macaulay2
sage: R = S/macaulay2('a^3+b^3+c^3+d^3') # optional - macaulay2
sage: X = R.Proj() # optional - macaulay2
sage: print(X.structure_sheaf()) # optional - macaulay2
oo sage...
```

subs (*args, **kwds)

Note that we have to override the substitute method so that we get the default one from Macaulay2 instead of the one provided by Element.

EXAMPLES:

```
sage: R = macaulay2("QQ[x]")  # optional - macaulay2
sage: P = macaulay2("ZZ/7[symbol x]")  # optional - macaulay2
sage: x, = R.gens()  # optional - macaulay2
sage: a = x^2 + 1  # optional - macaulay2
sage: a = a.substitute(P)  # optional - macaulay2
sage: a.to_sage().parent()  # optional - macaulay2
Univariate Polynomial Ring in x over Finite Field of size 7
```

substitute (*args, **kwds)

Note that we have to override the substitute method so that we get the default one from Macaulay2 instead of the one provided by Element.

EXAMPLES:

```
sage: R = macaulay2("QQ[x]")  # optional - macaulay2
sage: P = macaulay2("ZZ/7[symbol x]")  # optional - macaulay2
sage: x, = R.gens()  # optional - macaulay2
sage: a = x^2 + 1  # optional - macaulay2
sage: a = a.substitute(P)  # optional - macaulay2
sage: a.to_sage().parent()  # optional - macaulay2
Univariate Polynomial Ring in x over Finite Field of size 7
```

to_sage()

```
sage: macaulay2(ZZ).to_sage()
                             # optional - macaulay2
Integer Ring
sage: macaulay2(QQ).to_sage()
                             # optional - macaulay2
Rational Field
sage: macaulay2(2).to_sage()
                              # optional - macaulay2
sage: macaulay2(1/2).to_sage()
                                # optional - macaulay2
1/2
sage: macaulay2(2/1).to_sage()
                                # optional - macaulay2
                                  # optional - macaulay2
sage: _.parent()
Rational Field
sage: macaulay2([1,2,3]).to_sage() # optional - macaulay2
[1, 2, 3]
sage: m = matrix([[1,2],[3,4]])
sage: macaulay2(m).to_sage()
                                  # optional - macaulay2
[1 2]
[3 4]
```

```
sage: macaulay2(QQ['x,y']).to_sage() # optional - macaulay2
Multivariate Polynomial Ring in x, y over Rational Field
sage: macaulay2(QQ['x']).to_sage() # optional - macaulay2
Univariate Polynomial Ring in x over Rational Field
sage: macaulay2(GF(7)['x,y']).to_sage() # optional - macaulay2
Multivariate Polynomial Ring in x, y over Finite Field of size 7
sage: macaulay2(GF(7)).to_sage() # optional - macaulay2
Finite Field of size 7
sage: macaulay2(GF(49, 'a')).to_sage() # optional - macaulay2
Finite Field in a of size 7^2
sage: R. \langle x, y \rangle = QQ[]
sage: macaulay2(x^2+y^2+1).to_sage() # optional - macaulay2
x^2 + y^2 + 1
sage: R = macaulay2("QQ[x,y]")
                                     # optional - macaulay2
sage: I = macaulay2("ideal (x,y)")
                                     # optional - macaulay2
sage: I.to_sage()
                                      # optional - macaulay2
Ideal (x, y) of Multivariate Polynomial Ring in x, y over Rational Field
sage: X = R/I
                  # optional - macaulay2
sage: X.to_sage() # optional - macaulay2
Quotient of Multivariate Polynomial Ring in x, y over Rational Field by the
\rightarrowideal (x, y)
sage: R = macaulay2("QQ^2") # optional - macaulay2
sage: R.to_sage() # optional - macaulay2
Vector space of dimension 2 over Rational Field
sage: m = macaulay2('"hello"') # optional - macaulay2
sage: m.to_sage()
                              # optional - macaulay2
'hello'
```

underscore (x)

EXAMPLES:

```
sage: a = macaulay2([1,2,3]) # optional - macaulay2
sage: a.underscore(0) # optional - macaulay2
1
```

class sage.interfaces.macaulay2. Macaulay2Function (parent, name)

Bases: sage.interfaces.expect.ExpectFunction

sage.interfaces.macaulay2. is_Macaulay2Element (x)

EXAMPLES:

```
sage: from sage.interfaces.macaulay2 import is_Macaulay2Element
sage: is_Macaulay2Element(2) # optional - macaulay2
False
sage: is_Macaulay2Element(macaulay2(2)) # optional - macaulay2
True
```

sage.interfaces.macaulay2. macaulay2_console ()

Spawn a new M2 command-line session.

sage.interfaces.macaulay2. reduce_load_macaulay2 ()

Used for reconstructing a copy of the Macaulay2 interpreter from a pickle.

EXAMPLES:

```
sage: from sage.interfaces.macaulay2 import reduce_load_macaulay2
sage: reduce_load_macaulay2()
Macaulay2
```

sage.interfaces.macaulay2.remove_output_labels (s)

Remove output labels of Macaulay2 from a string.

•s: output of Macaulay2

•s: string

Returns: the input string with n symbols removed from the beginning of each line, where n is the minimal number of spaces or symbols of Macaulay2 output labels (looking like 'o39 = ') present on every non-empty line.

Return type: string

Note: If s consists of several outputs and their lables have different width, it is possible that some strings will have leading spaces (or maybe even pieces of output labels). However, this function will try not cut any messages.

```
sage: from sage.interfaces.macaulay2 import remove_output_labels
sage: output = 'o1 = QQ [x, y]\n\no1 : PolynomialRing\n'
sage: remove_output_labels(output)
'QQ [x, y]\n\nPolynomialRing\n'
```

CHAPTER

TWENTY

INTERFACE TO MAGMA

Sage provides an interface to the Magma computational algebra system. This system provides extensive functionality for number theory, group theory, combinatorics and algebra.

Note: You must have Magma installed on your computer for this interface to work. Magma is not free, so it is not included with Sage, but you can obtain it from http://magma.maths.usyd.edu.au/.

The Magma interface offers three pieces of functionality:

- 1. magma_console() A function that dumps you into an interactive command-line Magma session.
- 2. magma.new(obj) and alternatively magma(obj) Creation of a Magma object from a Sage object obj. This provides a Pythonic interface to Magma. For example, if f=magma.new(10), then f.Factors() returns the prime factorization of 10 computed using Magma. If obj is a string containing an arbitrary Magma expression, then the expression is evaluated in Magma to create a Magma object. An example is magma.new('10 div 3'), which returns Magma integer 3.
- 3. magma.eval (expr) Evaluation of the Magma expression expr, with the result returned as a string.

Type magma.[tab] for a list of all functions available from your Magma. Type magma.Function? for Magma's help about the Magma Function.

20.1 Parameters

Some Magma functions have optional "parameters", which are arguments that in Magma go after a colon. In Sage, you pass these using named function arguments. For example,

```
sage: E = magma('EllipticCurve([0,1,1,-1,0])') # optional - magma
sage: E.Rank(Bound = 5) # optional - magma
0
```

20.2 Multiple Return Values

Some Magma functions return more than one value. You can control how many you get using the nvals named parameter to a function call:

```
sage: n = magma(100)  # optional - magma
sage: n.IsSquare(nvals = 1)  # optional - magma
true
sage: n.IsSquare(nvals = 2)  # optional - magma
```

```
(true, 10)
sage: n = magma(-2006)  # optional - magma
sage: n.Factorization()  # optional - magma
[ <2, 1>, <17, 1>, <59, 1> ]
sage: n.Factorization(nvals=2)  # optional - magma
([ <2, 1>, <17, 1>, <59, 1> ], -1)
```

We verify that an obviously principal ideal is principal:

```
sage: _ = magma.eval('R<x> := PolynomialRing(RationalField())')  # optional - magma
sage: O = magma.NumberField('x^2+23').MaximalOrder()  # optional - magma
sage: I = magma('ideal<%s|%s.1>'%(O.name(),O.name()))  # optional - magma
sage: I.IsPrincipal(nvals=2)  # optional - magma
(true, [1, 0])
```

20.3 Long Input

The Magma interface reads in even very long input (using files) in a robust manner.

```
sage: t = '"%s"'%10^10000 # ten thousand character string. # optional - magma
sage: a = magma.eval(t) # optional - magma
sage: a = magma(t) # optional - magma
```

20.4 Garbage Collection

There is a subtle point with the Magma interface, which arises from how garbage collection works. Consider the following session:

First, create a matrix m in Sage:

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

Then I create a corresponding matrix A in Magma:

```
sage: A = magma(m) # optional - magma
```

It is called _sage_[...] in Magma:

```
sage: s = A.name(); s # optional - magma
'_sage_[...]'
```

It's there:

Now I delete the reference to that matrix:

```
sage: del A # optional - magma
```

Now _sage_[...] is "zeroed out" in the Magma session:

```
sage: magma.eval(s) # optional - magma
'0'
```

If Sage did not do this garbage collection, then every single time you ever create any magma object from a sage object, e.g., by doing magma(m), you would use up a lot of memory in that Magma session. This would lead to a horrible memory leak situation, which would make the Magma interface nearly useless for serious work.

20.5 Other Examples

We compute a space of modular forms with character.

```
sage: N = 20
sage: D = 20
sage: eps_top = fundamental_discriminant(D)
sage: eps = magma.KroneckerCharacter(eps_top, RationalField())
                                                                       # optional -_
                                                                        # optional -_
sage: M2 = magma.ModularForms(eps)
sage: print(M2)
                                                                        # optional -_
→magma
Space of modular forms on Gamma_1(5) ...
sage: print(M2.Basis())
                                                                        # optional -_
→magma
1 + 10*q^2 + 20*q^3 + 20*q^5 + 60*q^7 + \dots
q + q^2 + 2*q^3 + 3*q^4 + 5*q^5 + 2*q^6 + ...
```

In Sage/Python (and sort of C++) coercion of an element x into a structure S is denoted by S(x). This also works for the Magma interface:

```
# optional -
sage: G = magma.DirichletGroup(20)
sage: G.AssignNames(['a', 'b'])
                                                                       # optional -
→maqma
                                                                       # optional -_
sage: (G.1).Modulus()
→magma
sage: e = magma.DirichletGroup(40)(G.1)
                                                                       # optional -
sage: print(e)
                                                                       # optional -
→magma
$.1
                                                                       # optional -_
sage: print(e.Modulus())
→magma
40
```

We coerce some polynomial rings into Magma:

This example illustrates that Sage doesn't magically extend how Magma implicit coercion (what there is, at least) works. The errors below are the result of Magma having a rather limited automatic coercion system compared to Sage's:

```
sage: R.<x> = ZZ[]
sage: x * 5
5*x
sage: x * 1.0
sage: x * (2/3)
2/3*x
                                                                       # optional -_
sage: y = magma(x)
∽magma
                                                                        # optional -
sage: y * 5
→maqma
                                                                        # optional -
sage: y * 1.0
→magma
$.1
                                                                        # optional -_
sage: y * (2/3)
∽magma
Traceback (most recent call last):
TypeError: Error evaluating Magma code.
Argument types given: RngUPolElt[RngInt], FldRatElt
```

AUTHORS:

- William Stein (2005): initial version
- William Stein (2006-02-28): added extensive tab completion and interactive IPython documentation support.
- William Stein (2006-03-09): added nvals argument for magma.functions...

Interface to the Magma interpreter.

Type magma.[tab] for a list of all the functions available from your Magma install. Type magma.Function? for Magma's help about a given Function Type magma(...) to create a new Magma object, and magma.eval(...) to run a string using Magma (and get the result back as a string).

Note: If you do not own a local copy of Magma, try using the magma_free command instead, which uses the free demo web interface to Magma.

If you have ssh access to a remote installation of Magma, you can also set the server parameter to use it.

You must use nvals = 0 to call a function that doesn't return anything, otherwise you'll get an error. (nvals is the number of return values.)

Attach (filename)

Attach the given file to the running instance of Magma.

Attaching a file in Magma makes all intrinsics defined in the file available to the shell. Moreover, if the file doesn't start with the freeze; command, then the file is reloaded whenever it is changed. Note that functions and procedures defined in the file are *not* available. For only those, use magma.load(filename)

INPUT:

•filename - a string

EXAMPLES: Attaching a file that exists is fine:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE'] # optional - magma
sage: magma.attach('%s/magma/sage/basic.m'%SAGE_EXTCODE) # optional - magma
```

Attaching a file that doesn't exist raises an exception:

AttachSpec (filename)

Attach the given spec file to the running instance of Magma.

This can attach numerous other files to the running Magma (see the Magma documentation for more details).

INPUT:

•filename - a string

EXAMPLES:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE']  # optional - magma
sage: magma.attach_spec('%s/magma/spec'%SAGE_EXTCODE)  # optional - magma
sage: magma.attach_spec('%s/magma/spec2'%SAGE_EXTCODE)  # optional - magma
Traceback (most recent call last):
...
RuntimeError: Can't open package spec file .../magma/spec2 for reading (No_
⇒such file or directory)
```

GetVerbose (type)

Get the verbosity level of a given algorithm class etc. in Magma.

INPUT:

•type - string (e.g. 'Groebner'), see Magma documentation

Note: This method is provided to be consistent with the Magma naming convention.

EXAMPLES:

```
sage: magma.SetVerbose("Groebner", 2) # optional - magma
sage: magma.GetVerbose("Groebner") # optional - magma
2
```

SetVerbose (type, level)

Set the verbosity level for a given algorithm class etc. in Magma.

INPUT:

- •type string (e.g. 'Groebner'), see Magma documentation
- •level integer = 0

Note: This method is provided to be consistent with the Magma naming convention.

```
sage: magma.SetVerbose("Groebner", 2) # optional - magma
sage: magma.GetVerbose("Groebner") # optional - magma
2
```

attach (filename)

Attach the given file to the running instance of Magma.

Attaching a file in Magma makes all intrinsics defined in the file available to the shell. Moreover, if the file doesn't start with the freeze; command, then the file is reloaded whenever it is changed. Note that functions and procedures defined in the file are *not* available. For only those, use magma.load(filename)

INPUT:

•filename - a string

EXAMPLES: Attaching a file that exists is fine:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE'] # optional - magma
sage: magma.attach('%s/magma/sage/basic.m'%SAGE_EXTCODE) # optional - magma
```

Attaching a file that doesn't exist raises an exception:

attach_spec (filename)

Attach the given spec file to the running instance of Magma.

This can attach numerous other files to the running Magma (see the Magma documentation for more details).

INPUT:

•filename - a string

EXAMPLES:

```
sage: SAGE_EXTCODE = SAGE_ENV['SAGE_EXTCODE'] # optional - magma
sage: magma.attach_spec('%s/magma/spec'%SAGE_EXTCODE) # optional - magma
sage: magma.attach_spec('%s/magma/spec2'%SAGE_EXTCODE) # optional - magma
Traceback (most recent call last):
...
RuntimeError: Can't open package spec file .../magma/spec2 for reading (No_
⇒such file or directory)
```

bar call (left, name, gens, nvals=1)

This is a wrapper around the Magma constructor

nameleft gens

returning nvals.

INPUT:

- •left something coerceable to a magma object
- •name name of the constructor, e.g., sub, quo, ideal, etc.
- •gens if a list/tuple, each item is coerced to magma; otherwise gens itself is converted to magma
- •nvals positive integer; number of return values

OUTPUT: a single magma object if nvals == 1; otherwise a tuple of nvals magma objects.

EXAMPLES: The bar_call function is used by the sub, quo, and ideal methods of Magma elements. Here we illustrate directly using bar_call to create quotients:

chdir (dir)

Change the Magma interpreter's current working directory.

INPUT:

•dir -a string

```
sage: magma.chdir('/')  # optional - magma
sage: magma.eval('System("pwd")')  # optional - magma
'/'
```

clear (var)

Clear the variable named var and make it available to be used again.

INPUT:

```
•var - a string
```

EXAMPLES:

```
sage: magma = Magma()  # optional - magma
sage: magma.clear('foo')  # sets foo to 0 in magma; optional - magma
sage: magma.eval('foo')  # optional - magma
'0'
```

Because we cleared foo, it is set to be used as a variable name in the future:

```
sage: a = magma('10')  # optional - magma
sage: a.name()  # optional - magma
'foo'
```

The following tests that the whole variable clearing and freeing system is working correctly.

```
sage: magma = Magma()
                      # optional - magma
sage: a = magma('100')
                     # optional - magma
sage: a.name()
                       # optional - magma
'_sage_[1]'
sage: del a
                      # optional - magma
sage: b.name()
                      # optional - magma
'_sage_[1]'
sage: del b
                      # optional - magma
sage: magma('_sage_[1]') # optional - magma
0
```

console ()

Run a command line Magma session. This session is completely separate from this Magma interface.

EXAMPLES:

cputime (t=None)

Return the CPU time in seconds that has elapsed since this Magma session started. This is a floating point number, computed by Magma.

If t is given, then instead return the floating point time from when t seconds had elapsed. This is useful for computing elapsed times between two points in a running program.

INPUT:

•t - float (default: None); if not None, return cputime since t

OUTPUT:

•float - seconds

```
sage: type (magma.cputime())  # optional - magma
<type 'float'>
sage: magma.cputime()  # random, optional - magma
1.93999999999999
sage: t = magma.cputime()  # optional - magma
sage: magma.cputime(t)  # random, optional - magma
0.02
```

eval (x, strip=True, **kwds)

Evaluate the given block x of code in Magma and return the output as a string.

INPUT:

- •x string of code
- •strip -ignored

OUTPUT: string

EXAMPLES:

We evaluate a string that involves assigning to a variable and printing.

```
sage: magma.eval("a := 10;print 2+a;") # optional - magma
'12'
```

We evaluate a large input line (note that no weird output appears and that this works quickly).

```
sage: magma.eval("a := %s;"%(10^10000)) # optional - magma
''
```

Verify that trac ticket #9705 is fixed:

```
sage: nl=chr(10) # newline character
sage: magma.eval( # optional - magma
... "_<x>:=PolynomialRing(Rationals());"+nl+
... "repeat"+nl+
... " g:=3*b*x^4+18*c*x^3-6*b^2*x^2-6*b*c*x-b^3-9*c^2 where b:=Random([-10...
$\index*10]$) where c:=Random([-10..10]);"+nl+
... "until g ne 0 and Roots(g) ne [];"+nl+
... "print "success";")
'success'
```

Verify that trac ticket #11401 is fixed:

```
sage: nl=chr(10) # newline character
sage: magma.eval("a:=3;"+nl+"b:=5;") == nl # optional - magma
True
sage: magma.eval("[a,b];") # optional - magma
'[3,5]'
```

function_call (function, args=[], params={}, nvals=1)

Return result of evaluating a Magma function with given input, parameters, and asking for nvals as output.

INPUT:

- •function string, a Magma function name
- •args list of objects coercible into this magma interface
- •params Magma parameters, passed in after a colon

 ${\color{blue} \bullet} \texttt{nvals}\,$ - number of return values from the function to ask Magma for

OUTPUT: MagmaElement or tuple of nvals MagmaElement's

EXAMPLES:

Next, we illustrate multiple return values:

```
sage: magma.function_call('IsSquare', 100)  # optional - magma
true
sage: magma.function_call('IsSquare', 100, nvals=2)  # optional - magma
(true, 10)
sage: magma.function_call('IsSquare', 100, nvals=3)  # optional - magma
Traceback (most recent call last):
...
RuntimeError: Error evaluating Magma code...
Runtime error in :=: Expected to assign 3 value(s) but only computed 2______
→value(s)
```

get (var)

Get the value of the variable var.

INPUT:

•var - string; name of a variable defined in the Magma session

OUTPUT:

•string - string representation of the value of the variable.

EXAMPLES:

```
sage: magma.set('abc', '2 + 3/5') # optional - magma
sage: magma.get('abc') # optional - magma
'13/5'
```

get_verbose (type)

Get the verbosity level of a given algorithm class etc. in Magma.

INPUT:

•type - string (e.g. 'Groebner'), see Magma documentation

EXAMPLES:

```
sage: magma.set_verbose("Groebner", 2) # optional - magma
sage: magma.get_verbose("Groebner") # optional - magma
2
```

help(s)

Return Magma help on string s.

This returns what typing ?s would return in Magma.

INPUT:

•s - string

OUTPUT: string

EXAMPLES:

ideal(L)

Return the Magma ideal defined by L.

INPUT:

•L - a list of elements of a Sage multivariate polynomial ring.

OUTPUT: The magma ideal generated by the elements of L.

EXAMPLES:

```
sage: R.<x,y> = QQ[]
sage: magma.ideal([x^2, y^3*x])  # optional - magma
Ideal of Polynomial ring of rank 2 over Rational Field
Order: Graded Reverse Lexicographical
Variables: x, y
Homogeneous
Basis:
[
x^2,
x*y^3
]
```

load (filename)

Load the file with given filename using the 'load' command in the Magma shell.

Loading a file in Magma makes all the functions and procedures in the file available. The file should not contain any intrinsics (or you'll get errors). It also runs code in the file, which can produce output.

INPUT:

•filename - string

OUTPUT: output printed when loading the file

```
sage: magma('f(12)')  # optional - magma
144
```

objgens (value, gens)

Create a new object with given value and gens.

INPUT:

•value - something coercible to an element of this Magma interface

•gens - string; comma separated list of variable names

OUTPUT: new Magma element that is equal to value with given gens

EXAMPLES:

Because of how Magma works you can use this to change the variable names of the generators of an object:

set (var, value)

Set the variable var to the given value in the Magma interpreter.

INPUT:

- •var string; a variable name
- •value string; what to set var equal to

EXAMPLES:

```
sage: magma.set('abc', '2 + 3/5') # optional - magma
sage: magma('abc') # optional - magma
13/5
```

set_seed (seed=None)

Sets the seed for the Magma interpeter. The seed should be an integer.

EXAMPLES:

```
sage: m = Magma() # optional - magma
sage: m.set_seed(1) # optional - magma
1
sage: [m.Random(100) for i in range(5)] # optional - magma
[95, 20, 61, 59, 24]
```

set_verbose (type, level)

Set the verbosity level for a given algorithm, class, etc. in Magma.

INPUT:

```
•type - string (e.g. 'Groebner')
```

•level - integer = 0

EXAMPLES:

```
sage: magma.set_verbose("Groebner", 2) # optional - magma
sage: magma.get_verbose("Groebner") # optional - magma
2
```

version ()

Return the version of Magma that you have in your PATH on your computer.

OUTPUT:

- •numbers 3-tuple: major, minor, etc.
- •string version as a string

EXAMPLES:

```
sage: magma.version()  # random, optional - magma
((2, 14, 9), 'V2.14-9')
```

class sage.interfaces.magma. MagmaElement (parent, value, is_name=False, name=None)

 $Bases: \\ sage.interfaces.tab_completion.ExtraTabCompletion$

sage.interfaces.expect.ExpectElement

AssignNames (names)

EXAMPLES:

```
sage: G = magma.DirichletGroup(20) # optional - magma
sage: G.AssignNames(['a','b']) # optional - magma
sage: G.1 # optional - magma
a
```

```
sage: G.Elements() # optional - magma
[
1,
a,
b,
a*b
]
```

assign_names (names)

```
sage: G = magma.DirichletGroup(20) # optional - magma
sage: G.AssignNames(['a','b']) # optional - magma
sage: G.1 # optional - magma
a
```

```
sage: G.Elements() # optional - magma
[
1,
a,
b,
```

```
a*b
]
```

eval (*args)

Evaluate self at the inputs.

INPUT:

•*args - import arguments

OUTPUT: self(*args)

EXAMPLES:

```
sage: f = magma('Factorization')  # optional - magma
sage: f.evaluate(15)  # optional - magma
[ <3, 1>, <5, 1> ]
sage: f(15)  # optional - magma
[ <3, 1>, <5, 1> ]
sage: f = magma('GCD')  # optional - magma
sage: f.evaluate(15,20)  # optional - magma
```

evaluate (*args)

Evaluate self at the inputs.

INPUT:

•*args - import arguments

OUTPUT: self(*args)

EXAMPLES:

```
sage: f = magma('Factorization')  # optional - magma
sage: f.evaluate(15)  # optional - magma
[ <3, 1>, <5, 1> ]
sage: f(15)  # optional - magma
[ <3, 1>, <5, 1> ]
sage: f = magma('GCD')  # optional - magma
sage: f.evaluate(15,20)  # optional - magma
5
```

gen(n)

Return the n-th generator of this Magma element. Note that generators are 1-based in Magma rather than 0 based!

INPUT:

•n - a positive integer

OUTPUT: MagmaElement

```
sage: k.<a> = GF(9)
sage: magma(k).gen(1)  # optional -- magma
a
sage: R.<s,t,w> = k[]
sage: m = magma(R)  # optional -- magma
sage: m.gen(1)  # optional -- magma
s
```

```
sage: m.gen(2)  # optional -- magma
t
sage: m.gen(3)  # optional -- magma
w
sage: m.gen(0)  # optional -- magma
Traceback (most recent call last):
...
IndexError: index must be positive since Magma indexes are 1-based
sage: m.gen(4)  # optional -- magma
Traceback (most recent call last):
...
IndexError: list index out of range
```

gen_names ()

Return list of Magma variable names of the generators of self.

Note: As illustrated below, these are not the print names of the generators of the Magma object, but special variable names in the Magma session that reference the generators.

EXAMPLES:

```
sage: R.<x,zw> = QQ[]
sage: S = magma(R)  # optional - magma
sage: S.gen_names()  # optional - magma
('_sage_[...]', '_sage_[...]')
sage: magma(S.gen_names()[1])  # optional - magma
zw
```

gens ()

Return generators for self.

If self is named X is Magma, this function evaluates X.1, X.2, etc., in Magma until an error occurs. It then returns a Sage list of the resulting X.i. Note - I don't think there is a Magma command that returns the list of valid X.i. There are numerous ad hoc functions for various classes but nothing systematic. This function gets around that problem. Again, this is something that should probably be reported to the Magma group and fixed there.

AUTHORS:

•William Stein (2006-07-02)

EXAMPLES

get_magma_attribute (attrname)

Return value of a given Magma attribute. This is like selfattrname in Magma.

OUTPUT: MagmaElement

```
sage: V = magma("VectorSpace(RationalField(),10)") # optional - magma
sage: V.set_magma_attribute('M','"hello"') # optional - magma
sage: V.get_magma_attribute('M') # optional - magma
hello
sage: V.M # optional - magma
hello
```

ideal (gens)

Return the ideal of self with given list of generators.

INPUT:

•gens - object or list/tuple of generators

OUTPUT:

•magma element - a Magma ideal

EXAMPLES:

list attributes ()

Return the attributes of self, obtained by calling the ListAttributes function in Magma.

OUTPUT: list of strings

EXAMPLES: We observe that vector spaces in Magma have numerous funny and mysterious attributes.

```
sage: V = magma("VectorSpace(RationalField(),2)")  # optional - magma
sage: v = V.list_attributes(); v.sort()  # optional - magma
sage: print(v)  # optional - magma
['Coroots', 'Involution', ..., 'p', 'ssbasis', 'weights']
```

methods (any=False)

Return signatures of all Magma intrinsics that can take self as the first argument, as strings.

INPUT:

•any - (bool: default is False) if True, also include signatures with Any as first argument.

OUTPUT: list of strings

EXAMPLES:

quo (gens)

Return the quotient of self by the given object or list of generators.

INPUT

•gens - object or list/tuple of generators

OUTPUT:

```
•magma element - the quotient object
```

•magma element - mapping from self to the quotient object

EXAMPLES:

We illustrate quotienting out by an object instead of a list of generators:

We quotient a ZZ module out by a submodule.

```
sage: V = magma.RModule(ZZ,3); V # optional - magma
RModule(IntegerRing(), 3)
sage: W, phi = V.quo([[1,2,3]]) # optional - magma
sage: W # optional - magma
RModule(IntegerRing(), 2)
sage: phi # optional - magma
Mapping from: RModule(IntegerRing(), 3) to RModule(IntegerRing(), 2)
```

set_magma_attribute (attrname, value)

INPUT: attrname - string value - something coercible to a MagmaElement

EXAMPLES:

```
sage: V = magma("VectorSpace(RationalField(),2)") # optional - magma
sage: V.set_magma_attribute('M',10) # optional - magma
sage: V.get_magma_attribute('M') # optional - magma
10
sage: V.M # optional - magma
10
```

sub (gens)

Return the sub-object of self with given gens.

INPUT:

•gens - object or list/tuple of generators

```
sage: V = magma('VectorSpace(RationalField(),3)')  # optional - magma
sage: W = V.sub([[1,2,3], [1,1,2]]); W  # optional - magma
Vector space of degree 3, dimension 2 over Rational Field
Generators:
(1 2 3)
(1 1 2)
Echelonized basis:
(1 0 1)
(0 1 1)
```

```
class sage.interfaces.magma. MagmaFunction (parent, name)
     Bases: sage.interfaces.expect.ExpectFunction
class sage.interfaces.magma. MagmaFunctionElement (obj, name)
     Bases: sage.interfaces.expect.FunctionElement
class sage.interfaces.magma. MagmaGBDefaultContext ( magma=None)
     Context to force preservation of verbosity options for Magma's Groebner basis computation.
class sage.interfaces.magma. MagmaGBLogPrettyPrinter (verbosity=1, style='magma')
     A device which filters Magma Groebner basis computation logs.
     flush ()
         EXAMPLE:
         sage: from sage.interfaces.magma import MagmaGBLogPrettyPrinter
         sage: logs = MagmaGBLogPrettyPrinter()
         sage: logs.flush()
     write (s)
         EXAMPLE:
         sage: P. \langle x, y, z \rangle = GF(32003)[]
         sage: I = sage.rings.ideal.Katsura(P)
         sage: _ = I.groebner_basis('magma',prot=True) # indirect doctest, optional -...
         →magma
         Homogeneous weights search
         Total Faugere F4 time: ..., real time: ...
sage.interfaces.magma.extcode_dir (iface=None)
     Return directory that contains all the Magma extcode. This is put in a writable directory owned by the user,
     since when attached, Magma has to write sig and lck files.
     EXAMPLES:
     sage: sage.interfaces.magma.extcode_dir()
     '...dir_.../data/'
sage.interfaces.magma. is_{MagmaElement} ( x)
     Return True if x is of type MagmaElement, and False otherwise.
     INPUT:
        •x - any object
     OUTPUT: bool
     EXAMPLES:
     sage: from sage.interfaces.magma import is_MagmaElement
     sage: is_MagmaElement(2)
     False
     sage: is_MagmaElement(magma(2))
                                                            # optional - magma
```

```
sage.interfaces.magma. magma_console ()
```

Run a command line Magma session.

EXAMPLES:

True

sage.interfaces.magma. magma_gb_standard_options (func)

Decorator to force default options for Magma.

EXAMPLE:

```
sage: P.<a,b,c,d,e> = PolynomialRing(GF(127))
sage: J = sage.rings.ideal.Cyclic(P).homogenize()
sage: from sage.misc.sageinspect import sage_getsource
sage: "mself" in sage_getsource(J._groebner_basis_magma)
True
```

sage.interfaces.magma.magma_version ()

Return the version of Magma that you have in your PATH on your computer.

OUTPUT:

- •numbers 3-tuple: major, minor, etc.
- •string version as a string

EXAMPLES:

```
sage: magma_version()  # random, optional - magma
((2, 14, 9), 'V2.14-9')
```

sage.interfaces.magma.reduce_load_Magma ()

Used in unpickling a Magma interface.

This functions just returns the global default Magma interface.

```
sage: sage.interfaces.magma.reduce_load_Magma()
Magma
```

Sage Reference Manual: Interpreter Interfaces, Release 7.5	

INTERFACE TO THE FREE ONLINE MAGMA CALCULATOR

```
class sage.interfaces.magma_free. MagmaExpr
    Bases: str

class sage.interfaces.magma_free. MagmaFree
    Evaluate MAGMA code without requiring that MAGMA be installed on your computer by using the free online MAGMA calculator.

EXAMPLES: sage: magma_free("Factorization(9290348092384)") # optional - internet [ <2, 5>, <290323377887, 1>]

eval (x, **kwds)

sage.interfaces.magma_free.magma_free_eval (code, strip=True, columns=0)
Use the free online MAGMA calculator to evaluate the given input code and return the answer as a string.

LIMITATIONS: The code must evaluate in at most 20 seconds and there is a limitation on the amount of RAM.

EXAMPLES: sage: magma_free("Factorization(9290348092384)") # optional - internet [ <2, 5>, <290323377887, 1>]
```



CHAPTER

TWENTYTWO

INTERFACE TO MAPLE

AUTHORS:

- William Stein (2005): maple interface
- Gregg Musiker (2006-02-02): tutorial
- William Stein (2006-03-05): added tab completion, e.g., maple.[tab], and help, e.g, maple.sin?.

You must have the optional commercial Maple interpreter installed and available as the command maple in your PATH in order to use this interface. You do not have to install any optional Sage packages.

Type maple. [tab] for a list of all the functions available from your Maple install. Type maple. [tab]? for Maple's help about a given function. Type maple (...) to create a new Maple object, and maple.eval(...) to run a string using Maple (and get the result back as a string).

EXAMPLES:

```
sage: maple('3 * 5')  # optional - maple

15
sage: maple.eval('ifactor(2005)')  # optional - maple
'``(5) *``(401)'
sage: maple.ifactor(2005)  # optional - maple
'`(5) *``(401)
sage: maple.fsolve('x^2=cos(x)+4', 'x=0..5')  # optional - maple
1.914020619
sage: maple.factor('x^5 - y^5')  # optional - maple
(x-y) *(x^4+x^3*y+x^2*y^2+x*y^3+y^4)
```

If the string "error" (case insensitive) occurs in the output of anything from Maple, a RuntimeError exception is raised.

22.1 Tutorial

AUTHORS:

• Gregg Musiker (2006-02-02): initial version.

This tutorial is based on the Maple Tutorial for number theory from http://www.math.mun.ca/~drideout/m3370/numtheory.html.

There are several ways to use the Maple Interface in Sage. We will discuss two of those ways in this tutorial.

1. If you have a maple expression such as

```
factor( (x^5-1));
```

We can write that in sage as

```
sage: maple('factor(x^5-1)')  # optional - maple
(x-1)*(x^4+x^3+x^2+x+1)
```

Notice, there is no need to use a semicolon.

2. Since Sage is written in Python, we can also import maple commands and write our scripts in a Pythonic way. For example, factor() is a maple command, so we can also factor in Sage using

```
sage: maple('(x^5-1)').factor() # optional - maple
(x-1)*(x^4+x^3+x^2+x+1)
```

where expression.command() means the same thing as command(expression) in Maple. We will use this second type of syntax whenever possible, resorting to the first when needed.

```
sage: maple('(x^12-1)/(x-1)').simplify() # optional - maple
x^11+x^10+x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x+1
```

The normal command will always reduce a rational function to the lowest terms. The factor command will factor a polynomial with rational coefficients into irreducible factors over the ring of integers. So for example,

```
sage: maple('(x^12-1)').factor() # optional - maple (x-1)*(x+1)*(x^2+x+1)*(x^2-x+1)*(x^2-x+1)*(x^2-x+1)*
```

```
sage: maple('(x^28-1)').factor() # optional - maple (x-1)*(x^6+x^5+x^4+x^3+x^2+x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(x+1)*(
```

Another important feature of maple is its online help. We can access this through sage as well. After reading the description of the command, you can press q to immediately get back to your original prompt.

Incidentally you can always get into a maple console by the command

```
sage: maple.console()  # not tested
sage: !maple  # not tested
```

Note that the above two commands are slightly different, and the first is preferred.

For example, for help on the maple command fibonacci, we type

```
sage: maple.help('fibonacci') # not tested, since it uses a pager
```

We see there are two choices. Type

```
sage: maple.help('combinat, fibonacci') # not tested, since it uses a pager
```

We now see how the Maple command fibonacci works under the combinatorics package. Try typing in

```
sage: maple.fibonacci(10) # optional - maple
fibonacci(10)
```

You will get fibonacci(10) as output since Maple has not loaded the combinatorics package yet. To rectify this type

```
sage: maple('combinat[fibonacci]')(10) # optional - maple
55
```

instead.

If you want to load the combinatorics package for future calculations, in Sage this can be done as

```
sage: maple.with_package('combinat') # optional - maple
```

or

```
sage: maple.load('combinat') # optional - maple
```

Now if we type maple.fibonacci(10), we get the correct output:

```
sage: maple.fibonacci(10) # optional - maple
55
```

Some common maple packages include combinat, linalg, and numtheory. To produce the first 19 Fibonacci numbers, use the sequence command.

```
sage: maple('seq(fibonacci(i), i=1..19)') # optional - maple
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181
```

Two other useful Maple commands are ifactor and isprime. For example

```
sage: maple.isprime(maple.fibonacci(27)) # optional - maple
false
sage: maple.ifactor(maple.fibonacci(27)) # optional - maple
``(2)*``(17)*``(53)*``(109)
```

Note that the isprime function that is included with Sage (which uses PARI) is better than the Maple one (it is faster and gives a provably correct answer, whereas Maple is sometimes wrong).

Let's say we want to write a maple program now that squares a number if it is positive and cubes it if it is negative. In maple, that would look like

```
mysqcu := proc(x)
if x > 0 then x^2;
else x^3; fi;
end;
```

In Sage, we write

22.1. Tutorial 149

More complicated programs should be put in a separate file and loaded.

Interface to the Maple interpreter.

Type maple.[tab] for a list of all the functions available from your Maple install. Type maple.[tab]? for Maple's help about a given function. Type maple(...) to create a new Maple object, and maple.eval(...) to run a string using Maple (and get the result back as a string).

clear (var)

Clear the variable named var.

To clear a Maple variable, you must assign 'itself' to itself. In Maple 'expr' prevents expr to be evaluated.

EXAMPLES:

```
sage: maple.set('xx', '2') # optional - maple
sage: maple.get('xx') # optional - maple
'2'
sage: maple.clear('xx') # optional - maple
sage: maple.get('xx') # optional - maple
'xx'
```

completions (s)

Return all commands that complete the command starting with the string s. This is like typing s[Ctrl-T] in the maple interpreter.

EXAMPLES:

```
sage: c = maple.completions('di') # optional - maple
sage: 'divide' in c # optional - maple
True
```

console ()

Spawn a new Maple command-line session.

EXAMPLES:

```
sage: maple.console() # not tested
    |^/| Maple 11 (IBM INTEL LINUX)
._|\| |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2007
\ MAPLE / All rights reserved. Maple is a trademark of
<____ > Waterloo Maple Inc.
    | Type ? for help.
>
```

cputime (t=None)

Returns the amount of CPU time that the Maple session has used. If t is not None, then it returns the difference between the current CPU time and t.

```
2*x
sage: maple.cputime(t)  # random; optional - maple
0.0
```

expect ()

Returns the pexpect object for this Maple session.

EXAMPLES:

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: maple.set('xx', '2') # optional - maple
sage: maple.get('xx') # optional - maple
'2'
```

help (str)

Display Maple help about str.

This is the same as typing "?str" in the Maple console.

INPUT:

•str - a string to search for in the maple help system

EXAMPLES:

```
sage: maple.help('digamma') #not tested
Psi - the Digamma and Polygamma functions
...
```

load (package)

Make a package of Maple procedures available in the interpreter.

INPUT:

package - string

EXAMPLES: Some functions are unknown to Maple until you use with to include the appropriate package.

```
sage: maple.quit() # reset maple; optional -- maple
                                           # optional - maple
sage: maple('partition(10)')
partition(10)
sage: maple('bell(10)')
                                            # optional - maple
bell(10)
sage: maple.with_package('combinat')
                                           # optional - maple
sage: maple('partition(10)')
                                            # optional - maple
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1,
→1, 2, 2], [1, 1, 1, 1, 2, 2, 2], [1, 1, 2, 2, 2, 2], [2, 2, 2, 2, 2], [1,...
→1, 1, 1, 1, 1, 1, 3], [1, 1, 1, 1, 1, 2, 3], [1, 1, 1, 2, 2, 3], [1, 2, 2, __
\rightarrow 2, 3], [1, 1, 1, 1, 3, 3], [1, 1, 2, 3, 3], [2, 2, 3, 3], [1, 3, 3, 3], [1, \square
 \rightarrow1, 1, 1, 1, 1, 4], [1, 1, 1, 1, 2, 4], [1, 1, 2, 2, 4], [2, 2, 2, 4], [1, \_
→1, 1, 3, 4], [1, 2, 3, 4], [3, 3, 4], [1, 1, 4, 4], [2, 4, 4], [1, 1, 1, 1, 1, ...
```

```
sage: maple('bell(10)')  # optional - maple
115975
sage: maple('fibonacci(10)')  # optional - maple
55
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: maple.set('xx', '2') # optional - maple
sage: maple.get('xx') # optional - maple
'2'
```

source (s)

Display the Maple source (if possible) about s. This is the same as returning the output produced by the following Maple commands:

interface(verboseproc=2): print(s)

INPUT:

•s - a string representing the function whose source code you want

EXAMPLES:

```
sage: maple.source('curry') #not tested
p -> subs('_X' = args[2 .. nargs], () -> p(_X, args))
```

with_package (package)

Make a package of Maple procedures available in the interpreter.

INPUT:

package - string

EXAMPLES: Some functions are unknown to Maple until you use with to include the appropriate package.

```
sage: maple.quit() # reset maple; optional -- maple
sage: maple('partition(10)')
                                        # optional - maple
partition(10)
sage: maple('bell(10)')
                                        # optional - maple
bell(10)
sage: maple.with_package('combinat')
                                        # optional - maple
sage: maple('partition(10)')
                                        # optional - maple
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 2], [1, 1, 1, 1, 1, 1,
→1, 2, 2], [1, 1, 1, 1, 2, 2, 2], [1, 1, 2, 2, 2, 2], [2, 2, 2, 2, 2], [1, __
\rightarrow 2, 3], [1, 1, 1, 1, 3, 3], [1, 1, 2, 3, 3], [2, 2, 3, 3], [1, 3, 3, 3], [1, \square
→1, 1, 1, 1, 1, 4], [1, 1, 1, 1, 2, 4], [1, 1, 2, 2, 4], [2, 2, 2, 4], [1,...
→1, 1, 3, 4], [1, 2, 3, 4], [3, 3, 4], [1, 1, 4, 4], [2, 4, 4], [1, 1, 1, 1, 1, ...
→1, 5], [1, 1, 1, 2, 5], [1, 2, 2, 5], [1, 1, 3, 5], [2, 3, 5], [1, 4, 5], __
→ [5, 5], [1, 1, 1, 1, 6], [1, 1, 2, 6], [2, 2, 6], [1, 3, 6], [4, 6], [1, 1, _
\rightarrow 1, 7], [1, 2, 7], [3, 7], [1, 1, 8], [2, 8], [1, 9], [10]]
sage: maple('bell(10)')
                                        # optional - maple
115975
sage: maple('fibonacci(10)')
                                         # optional - maple
55
```

EXAMPLES:

sage.interfaces.maple. reduce_load_Maple ()

Returns the maple object created in sage.interfaces.maple.

EXAMPLES:

```
sage: from sage.interfaces.maple import reduce_load_Maple
sage: reduce_load_Maple()
Maple
```

22.1. Tutorial 153

CHAPTER

TWENTYTHREE

INTERFACE TO MATHEMATICA

The Mathematica interface will only work if Mathematica is installed on your computer with a command line interface that runs when you give the math command. The interface lets you send certain Sage objects to Mathematica, run Mathematica functions, import certain Mathematica expressions to Sage, or any combination of the above.

To send a Sage object <code>sobj</code> to Mathematica, call <code>mathematica(sobj)</code>. This exports the Sage object to Mathematica and returns a new Sage object wrapping the Mathematica expression/variable, so that you can use the Mathematica variable from within Sage. You can then call Mathematica functions on the new object; for example:

```
sage: mobj = mathematica(x^2-1)  # optional - mathematica
sage: mobj.Factor()  # optional - mathematica
(-1 + x)*(1 + x)
```

In the above example the factorization is done using Mathematica's Factor[] function.

To see Mathematica's output you can simply print the Mathematica wrapper object. However if you want to import Mathematica's output back to Sage, call the Mathematica wrapper object's sage () method. This method returns a native Sage object:

```
sage: mobj = mathematica(x^2-1)  # optional - mathematica
sage: mobj2 = mobj.Factor(); mobj2  # optional - mathematica
(-1 + x) * (1 + x)
sage: mobj2.parent()  # optional - mathematica
Mathematica
sage: sobj = mobj2.sage(); sobj  # optional - mathematica
(x + 1) * (x - 1)
sage: sobj.parent()  # optional - mathematica
Symbolic Ring
```

If you want to run a Mathematica function and don't already have the input in the form of a Sage object, then it might be simpler to input a string to mathematica (expr). This string will be evaluated as if you had typed it into Mathematica:

```
sage: mathematica('Factor[x^2-1]')  # optional - mathematica
(-1 + x) * (1 + x)
sage: mathematica('Range[3]')  # optional - mathematica
\{1, 2, 3\}
```

If you don't want Sage to go to the trouble of creating a wrapper for the Mathematica expression, then you can call mathematica.eval(expr), which returns the result as a Mathematica AsciiArtString formatted string. If you want the result to be a string formatted like Mathematica's InputForm, call repr(mobj) on the wrapper object mobj. If you want a string formatted in Sage style, call mobj._sage_repr():

```
sage: mathematica.eval('x^2 - 1') # optional - mathematica
2
```

```
-1 + x

sage: repr(mathematica('Range[3]'))  # optional - mathematica
'{1, 2, 3}'

sage: mathematica('Range[3]')._sage_repr()  # optional - mathematica
'[1, 2, 3]'
```

Finally, if you just want to use a Mathematica command line from within Sage, the function mathematica_console() dumps you into an interactive command-line Mathematica session. This is an enhanced version of the usual Mathematica command-line, in that it provides readline editing and history (the usual one doesn't!)

23.1 Tutorial

We follow some of the tutorial from http://library.wolfram.com/conferences/devconf99/withoff/Basic1.html/.

For any of this to work you must buy and install the Mathematica program, and it must be available as the command math in your PATH.

23.1.1 Syntax

Now make 1 and add it to itself. The result is a Mathematica object.

```
sage: m = mathematica
sage: a = m(1) + m(1); a  # optional - mathematica
2
sage: a.parent()  # optional - mathematica
Mathematica
sage: m('1+1')  # optional - mathematica
2
sage: m(3)**m(50)  # optional - mathematica
```

The following is equivalent to Plus [2, 3] in Mathematica:

```
sage: m = mathematica
sage: m(2).Plus(m(3)) # optional - mathematica
5
```

We can also compute 7(2+3).

```
sage: m(7).Times(m(2).Plus(m(3))) # optional - mathematica
35
sage: m('7(2+3)') # optional - mathematica
35
```

23.1.2 Some typical input

We solve an equation and a system of two equations:

23.1.3 Assignments and definitions

If you assign the mathematica 5 to a variable c in Sage, this does not affect the c in Mathematica.

The Sage interfaces changes Sage lists into Mathematica lists:

```
sage: m = mathematica
sage: eq1 = m('x^2 - 3y == 3')  # optional - mathematica
sage: eq2 = m('2x - y == 1')  # optional - mathematica
sage: v = m([eq1, eq2]); v  # optional - mathematica
{x^2 - 3*y == 3, 2*x - y == 1}
sage: v.Solve(['x', 'y'])  # optional - mathematica
{{x -> 0, y -> -1}, {x -> 6, y -> 11}}
```

23.1.4 Function definitions

Define mathematica functions by simply sending the definition to the interpreter.

```
sage: m = mathematica
sage: _ = mathematica('f[p_] = p^2');  # optional - mathematica
sage: m('f[9]')  # optional - mathematica
81
```

23.1.5 Numerical Calculations

We find the x such that $e^x - 3x = 0$.

```
sage: e = mathematica('Exp[x] - 3x == 0') # optional - mathematica
sage: e.FindRoot(['x', 2]) # optional - mathematica
\{x \rightarrow 1.512134551657842\}
```

Note that this agrees with what the PARI interpreter gp produces:

```
sage: gp('solve(x=1,2,exp(x)-3*x)')
1.512134551657842473896739678 # 32-bit
1.5121345516578424738967396780720387046 # 64-bit
```

Next we find the minimum of a polynomial using the two different ways of accessing Mathematica:

23.1. Tutorial 157

23.1.6 Polynomial and Integer Factorization

We factor a polynomial of degree 200 over the integers.

```
sage: R.<x> = PolynomialRing(ZZ)
sage: f = (x**100+17*x+5)*(x**100-5*x+20)
sage: f
x^200 + 12*x^101 + 25*x^100 - 85*x^2 + 315*x + 100
sage: g = mathematica(str(f))
                                         # optional - mathematica
                                          # optional - mathematica
sage: print(g)
                           2 100
                                            101
         100 + 315 x - 85 x + 25 x + 12 x + x
sage: q
                                          # optional - mathematica
100 + 315 \times x - 85 \times x^2 + 25 \times x^100 + 12 \times x^101 + x^200
sage: print(g.Factor())
                                         # optional - mathematica
                      100
                                        100
         (20 - 5 x + x) (5 + 17 x + x)
```

We can also factor a multivariate polynomial:

We factor an integer:

```
sage: n = mathematica(2434500)
                                            # optional - mathematica
                                            # optional - mathematica
sage: n.FactorInteger()
\{\{2, 2\}, \{3, 2\}, \{5, 3\}, \{541, 1\}\}
sage: n = mathematica(2434500)
                                            # optional - mathematica
                                            # optional - mathematica
sage: F = n.FactorInteger(); F
\{\{2, 2\}, \{3, 2\}, \{5, 3\}, \{541, 1\}\}
sage: F[1]
                                            # optional - mathematica
{2, 2}
sage: F[4]
                                            # optional - mathematica
{541, 1}
```

Mathematica's ECM package is no longer available.

23.2 Long Input

The Mathematica interface reads in even very long input (using files) in a robust manner.

```
sage: t = '"%s"'%10^10000 # ten thousand character string.
sage: a = mathematica(t) # optional - mathematica
sage: a = mathematica.eval(t) # optional - mathematica
```

23.3 Loading and saving

Mathematica has an excellent InputForm function, which makes saving and loading Mathematica objects possible. The first examples test saving and loading to strings.

```
sage: x = mathematica(pi/2)
                                # optional - mathematica
sage: print(x)
                                # optional - mathematica
        Ρi
         2
sage: loads(dumps(x)) == x
                           # optional - mathematica
True
sage: n = x.N(50)
                                # optional - mathematica
                               # optional - mathematica
sage: print(n)
             1.5707963267948966192313216916397514420985846996876
sage: loads(dumps(n)) == n # optional - mathematica
True
```

23.4 Complicated translations

The mobj.sage() method tries to convert a Mathematica object to a Sage object. In many cases, it will just work. In particular, it should be able to convert expressions entirely consisting of:

- numbers, i.e. integers, floats, complex numbers;
- functions and named constants also present in Sage, where:
 - Sage knows how to translate the function or constant's name from Mathematica's, or
 - the Sage name for the function or constant is trivially related to Mathematica's;
- symbolic variables whose names don't pathologically overlap with objects already defined in Sage.

This method will not work when Mathematica's output includes:

- strings;
- functions unknown to Sage;
- Mathematica functions with different parameters/parameter order to the Sage equivalent.

If you want to convert more complicated Mathematica expressions, you can instead call mobj._sage_() and supply a translation dictionary:

```
sage: m = mathematica('NewFn[x]') # optional - mathematica
sage: m._sage_(locals={'NewFn': sin}) # optional - mathematica
sin(x)
```

For more details, see the documentation for ._sage_() .

OTHER Examples:

```
sage: def math_bessel_K(nu,x):
...    return mathematica(nu).BesselK(x).N(20)
...
sage: math_bessel_K(2,I)  # optional - mathematica
-2.5928861754911969782 + 0.1804899720669620266 I
```

```
sage: slist = [[1, 2], 3., 4 + I]
sage: mlist = mathematica(slist); mlist
                                             # optional - mathematica
\{\{1, 2\}, 3., 4 + I\}
sage: slist2 = list(mlist); slist2
                                             # optional - mathematica
[\{1, 2\}, 3., 4 + I]
sage: slist2[0]
                                             # optional - mathematica
{1, 2}
sage: slist2[0].parent()
                                             # optional - mathematica
Mathematica
sage: slist3 = mlist.sage(); slist3
                                             # optional - mathematica
[[1, 2], 3.0, I + 4]
```

```
sage: mathematica('10.^80') # optional - mathematica
1.*^80
sage: mathematica('10.^80').sage() # optional - mathematica
1e+80
```

AUTHORS:

- William Stein (2005): first version
- Doug Cutrell (2006-03-01): Instructions for use under Cygwin/Windows.
- Felix Lawrence (2009-08-21): Added support for importing Mathematica lists and floats with exponents.

class sage.interfaces.mathematica. Mathematica (maxread=None,

```
script_subdirectory=None, logfile=None,
server=None, server tmpdir=None)
```

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect

Interface to the Mathematica interpreter.

```
chdir ( dir)
```

Change Mathematica's current working directory.

EXAMPLES:

```
sage: mathematica.chdir('/')  # optional - mathematica
sage: mathematica('Directory[]')  # optional - mathematica
"/"
```

```
console ( readline=True)
eval ( code, strip=True, **kwds)
get ( var, ascii_art=False)
    Get the value of the variable var.
AUTHORS:
```

•William Stein

•Kiran Kedlaya (2006-02-04): suggested using InputForm

help (cmd)

```
set (var, value)
```

Set the variable var to the given value.

```
Bases: sage.interfaces.expect.ExpectElement
```

N (precision=None)

Numerical approximation by calling Mathematica's N[]

Calling Mathematica's N[] function, with optional precision in decimal digits. Unlike Sage's n(), N() can be applied to symbolic Mathematica objects.

A workaround for trac ticket #18888 backtick issue, stripped away by get(), is included.

Note: The base class way up the hierarchy defines an N (modeled after Mathematica's) which overwrites the Mathematica one, and doesn't work at all. We restore it here.

EXAMPLES:

n (*args, **kwargs)

Numerical approximation by converting to Sage object first

Convert the object into a Sage object and return its numerical approximation. See documentation of the function sage.misc.functional.n() for details.

EXAMPLES:

```
sage: mathematica('Pi').n(10) # optional -- mathematica
3.1
sage: mathematica('Pi').n() # optional -- mathematica
3.14159265358979
sage: mathematica('Pi').n(digits=10) # optional -- mathematica
3.141592654
```

save_image (filename, ImageSize=600)

Save a mathematica graphics

INPUT:

- •filename string. The filename to save as. The extension determines the image file format.
- •ImageSize integer. The size of the resulting image.

```
sage: P = mathematica('Plot[Sin[x], {x,-2Pi,4Pi}]') # optional - mathematica
sage: filename = tmp_filename() # optional - mathematica
sage: P.save_image(filename, ImageSize=800) # optional -

→mathematica
```

```
show ( ImageSize=600)
```

Show a mathematica expression immediately.

This method attempts to display the graphics immediately, without waiting for the currently running code (if any) to return to the command line. Be careful, calling it from within a loop will potentially launch a large number of external viewer programs.

INPUT:

•ImageSize - integer. The size of the resulting image.

OUTPUT:

This method does not return anything. Use save () if you want to save the figure as an image.

```
sage: P = mathematica('Plot[Sin[x], {x, -2Pi, 4Pi}]')
                                                       # optional - mathematica
sage: show(P)
                                                       # optional - mathematica
sage: P.show(ImageSize=800)
                                                       # optional - mathematica
sage: Q = mathematica('Sin[x Cos[y]]/Sqrt[1-x^2]') # optional - mathematica
                                                       # optional - mathematica
sage: show(Q)
<html><script type="math/tex">\frac{\sin (x \cos (y))}{\sqrt{1-x^2}}</script>
\hookrightarrow</html>
```

```
str()
class sage.interfaces.mathematica. MathematicaFunction (parent, name)
    Bases: sage.interfaces.expect.ExpectFunction
class sage.interfaces.mathematica. MathematicaFunctionElement (obj, name)
    Bases: sage.interfaces.expect.FunctionElement
sage.interfaces.mathematica. clean_output (s)
sage.interfaces.mathematica.mathematica_console (readline=True)
sage.interfaces.mathematica. reduce_load(X)
```

CHAPTER

TWENTYFOUR

INTERFACE TO MATLAB

According to their website, MATLAB is "a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran."

The commands in this section only work if you have the "matlab" interpreter installed and available in your PATH. It's not necessary to install any special Sage packages.

EXAMPLES:

```
sage: matlab.eval('2+2') # optional - matlab
'\nans =\n\n 4\n'
```

```
sage: a = matlab(10)  # optional - matlab
sage: a**10  # optional - matlab
1.0000e+10
```

AUTHORS:

• William Stein (2006-10-11)

24.1 Tutorial

```
sage: matlab('4+10')
                                        # optional - matlab
sage: matlab('date')
                                        # optional - matlab; random output
18-Oct-2006
sage: matlab('5*10 + 6')
                                        # optional - matlab
sage: matlab('(6+6)/3')
                                        # optional - matlab
sage: matlab('9')^2
                                        # optional - matlab
sage: a = matlab(10); b = matlab(20); c = matlab(30) # optional - matlab
sage: avg = (a+b+c)/3; avg
                                        # optional - matlab
                                        # optional - matlab
sage: parent(avg)
Matlab
```

```
sage: my_scalar = matlab('3.1415') # optional - matlab
sage: my_scalar # optional - matlab
3.1415
```

```
sage: my_vector1 = matlab('[1,5,7]')  # optional - matlab
sage: my_vector1  # optional - matlab
1    5    7
sage: my_vector2 = matlab('[1;5;7]')  # optional - matlab
sage: my_vector2  # optional - matlab

1    5
7
sage: my_vector1 * my_vector2  # optional - matlab
75
```

```
sage: row_vector1 = matlab('[1 2 3]')  # optional - matlab
sage: row_vector2 = matlab('[3 2 1]')  # optional - matlab
sage: matrix_from_row_vec = matlab('[%s; %s]'%(row_vector1.name(), row_vector2.

→name()))  # optional - matlab
sage: matrix_from_row_vec  # optional - matlab

1  2  3
3  2  1
```

```
sage: tm = matlab('0.5:2:10')  # optional - matlab
sage: tm
0.5000  2.5000  4.5000  6.5000  8.5000
```

```
sage: my_vector1 = matlab('[1,5,7]')  # optional - matlab
sage: my_vector1(1)  # optional - matlab
1
sage: my_vector1(2)  # optional - matlab
5
sage: my_vector1(3)  # optional - matlab
7
```

Matrix indexing works as follows:

Setting using parenthesis cannot work (because of how the Python language works). Use square brackets or the set function:

```
sage: my_matrix = matlab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]')
                                                                        # optional -_
sage: my_matrix.set(2,3, 1999)
                                                        # optional - matlab
                                                        # optional - matlab
sage: my_matrix
          8
                     12
                                 19
          7
                      3
                                1999
          12
                      4
                                23
          8
                      1
                                  1
```

Bases: sage.interfaces.expect.Expect

Interface to the Matlab interpreter.

EXAMPLES:

```
sage: a = matlab('[ 1, 1, 2; 3, 5, 8; 13, 21, 33 ]')  # optional - matlab
sage: b = matlab('[ 1; 3; 13]')  # optional - matlab
sage: c = a * b  # optional - matlab
sage: print(c)  # optional - matlab

122
505
```

chdir (directory)

Change MATLAB's current working directory.

EXAMPLES:

```
sage: matlab.chdir('/')  # optional - matlab
sage: matlab.pwd()  # optional - matlab
/
```

console ()

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: s = matlab.eval('a = 2') # optional - matlab
sage: matlab.get('a') # optional - matlab
' 2'
```

sage2matlab_matrix_string (A)

Return an matlab matrix from a Sage matrix.

INPUT: A Sage matrix with entries in the rationals or reals.

OUTPUT: A string that evaluates to an Matlab matrix.

EXAMPLES:

24.1. Tutorial 165

```
sage: M33 = MatrixSpace(QQ,3,3)
sage: A = M33([1,2,3,4,5,6,7,8,0])
sage: matlab.sage2matlab_matrix_string(A) # optional - matlab
'[1, 2, 3; 4, 5, 6; 7, 8, 0]'
```

AUTHOR:

•David Joyner and William Stein

```
set (var, value)
```

Set the variable var to the given value.

$strip_answer$ (s)

Returns the string s with Matlab's answer prompt removed.

EXAMPLES:

```
sage: s = '\nans =\n\n 2\n'
sage: matlab.strip_answer(s)
' 2'
```

version ()

```
whos ()
```

class sage.interfaces.matlab. MatlabElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.expect.ExpectElement

```
set (i, j, x)
```

sage.interfaces.matlab. matlab_console ()

This requires that the optional matlab program be installed and in your PATH, but no optional Sage packages need be installed.

EXAMPLES:

ans =

5

quit

Typing quit exits the matlab console and returns you to Sage. matlab, like Sage, remembers its history from one session to another.

```
sage.interfaces.matlab. {\tt matlab\_version} ()
```

Return the version of Matlab installed.

```
sage: matlab_version() # random; optional - matlab
'7.2.0.283 (R2006a)'
```

```
sage.interfaces.matlab. reduce_load_Matlab ()
```

CHAPTER

TWENTYFIVE

PEXPECT INTERFACE TO MAXIMA

Maxima is a free GPL'd general purpose computer algebra system whose development started in 1968 at MIT. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Maxima has implementations of many functions relating to the invariant theory of the symmetric group S_n . (However, the commands for group invariants, and the corresponding Maxima documentation, are in French.) For many links to Maxima documentation see http://maxima.sourceforge.net/documentation.html.

AUTHORS:

- William Stein (2005-12): Initial version
- David Joyner: Improved documentation
- William Stein (2006-01-08): Fixed bug in parsing
- William Stein (2006-02-22): comparisons (following suggestion of David Joyner)
- William Stein (2006-02-24): *greatly* improved robustness by adding sequence numbers to IO bracketing in _eval_line
- Robert Bradshaw, Nils Bruin, Jean-Pierre Flori (2010,2011): Binary library interface

This is the interface used by the maxima object:

```
sage: type(maxima)
<class 'sage.interfaces.maxima.Maxima'>
```

If the string "error" (case insensitive) occurs in the output of anything from Maxima, a RuntimeError exception is raised.

EXAMPLES: We evaluate a very simple expression in Maxima.

```
sage: maxima('3 * 5')
15
```

We factor $x^5 - y^5$ in Maxima in several different ways. The first way yields a Maxima object.

```
sage: F = maxima.factor('x^5 - y^5')
sage: F
- (y-x) * (y^4+x*y^3+x^2*y^2+x^3*y+x^4)
sage: type(F)
<class 'sage.interfaces.maxima.MaximaElement'>
```

Note that Maxima objects can also be displayed using "ASCII art"; to see a normal linear representation of any Maxima object x. Just use the print command: use str(x).

You can always use repr(x) to obtain the linear representation of an object. This can be useful for moving maxima data to other systems.

```
sage: repr(F)
'-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)'
sage: F.str()
'-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)'
```

The maxima .eval command evaluates an expression in maxima and returns the result as a *string* not a maxima object.

```
sage: print(maxima.eval('factor(x^5 - y^5)'))
-(y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)
```

We can create the polynomial f as a Maxima polynomial, then call the factor method on it. Notice that the notation f.factor() is consistent with how the rest of Sage works.

```
sage: f = maxima('x^5 - y^5')
sage: f^2
(x^5-y^5)^2
sage: f.factor()
- (y-x)*(y^4+x*y^3+x^2*y^2+x^3*y+x^4)
```

Control-C interruption works well with the maxima interface, because of the excellent implementation of maxima. For example, try the following sum but with a much bigger range, and hit control-C.

```
sage: maxima('sum(1/x^2, x, 1, 10)')
1968329/1270080
```

25.1 Tutorial

We follow the tutorial at http://maxima.sourceforge.net/docs/intromax/intromax.html.

```
sage: maxima('1/100 + 1/101')
201/10100
```

```
sage: a = maxima('(1 + sqrt(2))^5'); a
(sqrt(2)+1)^5
sage: a.expand()
29*sqrt(2)+41
```

```
sage: a = maxima('(1 + sqrt(2))^5')
sage: float(a)
82.01219330881975
sage: a.numer()
82.01219330881975
```

```
sage: maxima.eval('fpprec : 100')
'100'
```

```
sage: a.bfloat()
8.

→201219330881975641524897300208124427852048438593149412212371240173124187540110412666123849550160566
```

```
sage: maxima('100!')
9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828
```

```
sage: f = maxima('(x + 3*y + x^2*y)^3')
sage: f.expand()
x^6*y^3+9*x^4*y^3+27*x^2*y^3+27*y^3+3*x^5*y^2+18*x^3*y^2+27*x*y^2+3*x^4*y+9*x^2*y+x^3
sage: f.subst('x=5/z')
(5/z+25*y/z^2+3*y)^3
sage: g = f.subst('x=5/z')
sage: h = g.ratsimp(); h
(27*y^3*z^6+135*y^2*z^5+(675*y^3+225*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+125)*z^3+(5625*y^3+1875*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+(2250*y^2+185*y)*z^4+
```

```
sage: eqn = maxima(['a+b*c=1', 'b-a*c=0', 'a+b=5'])
sage: s = eqn.solve('[a,b,c]'); s
[[a=(25*sqrt(79)*%i+25)/(6*sqrt(79)*%i-34),b=(5*sqrt(79)*%i+5)/(sqrt(79)*
→%i+11),c=(sqrt(79)*%i+1)/10],[a=(25*sqrt(79)*%i-25)/(6*sqrt(79)*
→%i+34),b=(5*sqrt(79)*%i-5)/(sqrt(79)*%i-11),c=-(sqrt(79)*%i-1)/10]]
```

Here is an example of solving an algebraic equation:

```
sage: maxima('x^2+y^2=1').solve('y')
[y=-sqrt(1-x^2),y=sqrt(1-x^2)]
sage: maxima('x^2 + y^2 = (x^2 - y^2)/sqrt(x^2 + y^2)').solve('y')
[y=-sqrt((-y^2-x^2)*sqrt(y^2+x^2)+x^2),y=sqrt((-y^2-x^2)*sqrt(y^2+x^2)+x^2)]
```

You can even nicely typeset the solution in latex:

To have the above appear onscreen via xdvi, type view(s). (TODO: For OS X should create pdf output and use preview instead?)

```
sage: e = maxima('sin(u + v) * cos(u)^3'); e
cos(u)^3*sin(v+u)
sage: f = e.trigexpand(); f
cos(u)^3*(cos(u)*sin(v)+sin(u)*cos(v))
sage: f.trigreduce()
(sin(v+4*u)+sin(v-2*u))/8+(3*sin(v+2*u)+3*sin(v))/8
sage: w = maxima('3 + k*%i')
sage: f = w^2 + maxima('%e')^w
sage: f.realpart()
%e^3*cos(k)-k^2+9
```

```
sage: f = maxima('x^3 * %e^(k*x) * sin(w*x)'); f x^3*%e^(k*x)*sin(w*x)
```

25.1. Tutorial 169

```
sage: f.diff('x')
k*x^3*%e^(k*x)*sin(w*x)+3*x^2*%e^(k*x)*sin(w*x)+w*x^3*%e^(k*x)*cos(w*x)
sage: f.integrate('x')
(((k*w^6+3*k^3*w^4+3*k^5*w^2+k^7)*x^3+(3*w^6+3*k^2*w^4-3*k^4*w^2-3*k^6)*x^2+(-18*k*w^4+12*k^3*w^2+6*k^5)*x-6*w^4+36*k^2*w^2-6*k^4)*%e^(k*x)*sin(w*x)+((-w^7-3*k^2*w^5-4-12*k^3*w^3-k^6*w)*x^3+(6*k*w^5+12*k^3*w^3+6*k^5*w)*x^2+(6*w^5-12*k^2*w^3-18*k^4*w)*x-44*k*w^3+24*k^3*w)*%e^(k*x)*cos(w*x))/(w^8+4*k^2*w^6+6*k^4*w^4+4*k^6*w^2+k^8)
```

```
sage: f = maxima('1/x^2')
sage: f.integrate('x', 1, 'inf')
1
sage: g = maxima('f/sinh(k*x)^4')
sage: g.taylor('x', 0, 3)
f/(k^4*x^4)-2*f/(3*k^2*x^2)+11*f/45-62*k^2*f*x^2/945
```

```
sage: maxima.taylor('asin(x)','x',0, 10)
x+x^3/6+3*x^5/40+5*x^7/112+35*x^9/1152
```

25.2 Examples involving matrices

We illustrate computing with the matrix whose i, j entry is i/j, for $i, j = 1, \dots, 4$.

```
sage: f = maxima.eval('f[i,j] := i/j')
sage: A = maxima('genmatrix(f,4,4)'); A
matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
sage: A.determinant()
0
sage: A.echelon()
matrix([1,1/2,1/3,1/4],[0,0,0,0],[0,0,0,0],[0,0,0,0])
sage: A.eigenvalues()
[[0,4],[3,1]]
sage: A.eigenvectors()
[[0,4],[3,1]],[[[1,0,0,-4],[0,1,0,-2],[0,0,1,-4/3]],[[1,2,3,4]]]]
```

We can also compute the echelon form in Sage:

```
sage: B = matrix(QQ, A)
sage: B.echelon_form()
[ 1 1/2 1/3 1/4]
[ 0  0  0  0]
[ 0  0  0  0]
[ 0  0  0  0]
[ 0  0  0  0]
sage: B.charpoly('x').factor()
(x - 4) * x^3
```

25.3 Laplace Transforms

We illustrate Laplace transforms:

```
sage: _ = maxima.eval("f(t) := t*sin(t)")
sage: maxima("laplace(f(t),t,s)")
2*s/(s^2+1)^2
```

```
sage: maxima("laplace(delta(t-3),t,s)") #Dirac delta function
%e^-(3*s)
```

```
sage: _ = maxima.eval("f(t) := exp(t)*sin(t)")
sage: maxima("laplace(f(t),t,s)")
1/(s^2-2*s+2)
```

```
sage: maxima("laplace(diff(x(t),t),t,s)")
s*'laplace(x(t),t,s)-x(0)
```

```
sage: maxima("laplace(diff(x(t),t,2),t,s)")
-%at('diff(x(t),t,1),t=0)+s^2*'laplace(x(t),t,s)-x(0)*s
```

It is difficult to read some of these without the 2d representation:

Even better, use view (maxima ("laplace (diff (x(t),t,2),t,s)")) to see a typeset version.

25.4 Continued Fractions

A continued fraction $a + 1/(b + 1/(c + \cdots))$ is represented in maxima by the list $[a, b, c, \ldots]$.

```
sage: maxima("cf((1 + sqrt(5))/2)")
[1,1,1,1,2]
sage: maxima("cf ((1 + sqrt(341))/2)")
[9,1,2,1,2,1,17,1,2,1,2,1,17,1,2]
```

25.5 Special examples

In this section we illustrate calculations that would be awkward to do (as far as I know) in non-symbolic computer algebra systems like MAGMA or GAP.

We compute the gcd of $2x^{n+4} - x^{n+2}$ and $4x^{n+1} + 3x^n$ for arbitrary n.

```
sage: f = maxima('2*x^(n+4) - x^(n+2)')
sage: g = maxima('4*x^(n+1) + 3*x^n')
sage: f.gcd(g)
x^n
```

You can plot 3d graphs (via gnuplot):

```
sage: maxima('plot3d(x^2-y^2, [x,-2,2], [y,-2,2], [grid,12,12])') # not tested
[displays a 3 dimensional graph]
```

You can formally evaluate sums (note the nusum command):

We formally compute the limit as $n \to \infty$ of 2S/n as follows:

```
sage: T = S*maxima('2/n')
sage: T.tlimit('n','inf')
%e^3-%e
```

25.6 Miscellaneous

Obtaining digits of π :

Defining functions in maxima:

```
sage: maxima.eval('fun[a] := a^2')
'fun[a]:=a^2'
sage: maxima('fun[10]')
100
```

25.7 Interactivity

Unfortunately maxima doesn't seem to have a non-interactive mode, which is needed for the Sage interface. If any Sage call leads to maxima interactively answering questions, then the questions can't be answered and the maxima session may hang. See the discussion at http://www.ma.utexas.edu/pipermail/maxima/2005/011061.html for some ideas about how to fix this problem. An example that illustrates this problem is maxima.eval('integrate(exp(a*x),x,0,inf)').

25.8 Latex Output

To TeX a maxima object do this:

```
sage: latex(maxima('sin(u) + sinh(v^2)'))
\sinh v^2+\sin u
```

Here's another example:

```
sage: g = maxima('exp(3*%i*x)/(6*%i) + exp(%i*x)/(2*%i) + c')
sage: latex(g)
-{\{i\setminus,e^{3\setminus,i\setminus,x}\}\setminus e^{i\setminus,x}}\setminus e^{i\setminus,x}}
```

25.9 Long Input

The MAXIMA interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

Note: Using maxima.eval for long input is much less robust, and is not recommended.

```
sage: t = '"%s"'%10^10000 # ten thousand character string.
sage: a = maxima(t)
```

TESTS:

This working tests that a subtle bug has been fixed:

```
sage: f = maxima.function('x','gamma(x)')
sage: g = f(1/7)
sage: g
gamma(1/7)
sage: del f
sage: maxima(sin(x))
sin(_SAGE_VAR_x)
```

This tests to make sure we handle the case where Maxima asks if an expression is positive or zero.

```
sage: var('Ax,Bx,By')
(Ax, Bx, By)
sage: t = -Ax*sin(sqrt(Ax^2)/2)/(sqrt(Ax^2)*sqrt(By^2 + Bx^2))
sage: t.limit(Ax=0, dir='+')
0
```

A long complicated input expression:

Test that Maxima gracefully handles this syntax error (trac ticket #17667):

25.8. Latex Output 173

```
sage: maxima.eval("1 == 1;")
Traceback (most recent call last):
...
TypeError: ...incorrect syntax: = is not a prefix operator...
```

Bases: sage.interfaces.maxima_abstract.MaximaAbstract sage.interfaces.expect.Expect

Interface to the Maxima interpreter.

EXAMPLES:

```
sage: m = Maxima()
sage: m == maxima
False
```

clear (var)

Clear the variable named var.

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
sage: maxima.clear('xxxxx')
sage: maxima.get('xxxxx')
```

get (var)

Get the string value of the variable var.

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
```

lisp (cmd)

Send a lisp command to Maxima.

Note: The output of this command is very raw - not pretty.

EXAMPLES:

```
sage: maxima.lisp("(+ 2 17)")  # random formatted output
:lisp (+ 2 17)
19
(
```

set (var, value)

Set the variable var to the given value.

INPUT:

```
•var - string
```

EXAMPLES:

```
sage: maxima.set('xxxxx', '2')
sage: maxima.get('xxxxx')
'2'
```

set seed (seed=None)

http://maxima.sourceforge.net/docs/manual/maxima_10.html make_random_state (n) returns a new random state object created from an integer seed value equal to n modulo 2^32. n may be negative.

EXAMPLES:

```
sage: m = Maxima()
sage: m.set_seed(1)
1
sage: [m.random(100) for i in range(5)]
[45, 39, 24, 68, 63]
```

Element of Maxima through Pexpect interface.

EXAMPLES:

Elements of this class should not be created directly. The targeted parent should be used instead:

```
sage: maxima(3)
3
sage: maxima(cos(x)+e^234)
cos(_SAGE_VAR_x)+%e^234
```

display2d (onscreen=True)

Return the 2d string representation of this Maxima object.

EXAMPLES:

class sage.interfaces.maxima. MaximaElementFunction (parent, name, defn, args, latex)

Bases: sage.interfaces.maxima.MaximaElement, sage.interfaces.maxima_abstract.MaximaAbstra

Maxima user-defined functions.

EXAMPLES:

Elements of this class should not be created directly. The method function of the targeted parent should be used instead:

```
sage: maxima.function('x,y','h(x)*y')
h(x)*y
```

25.9. Long Input 175

EXAMPLES:

```
sage: from sage.interfaces.maxima import is_MaximaElement
sage: m = maxima(1)
sage: is_MaximaElement(m)
True
sage: is_MaximaElement(1)
False
```

sage.interfaces.maxima. $reduce_load_Maxima$ ()

Unpickle a Maxima Pexpect interface.

EXAMPLES:

```
sage: from sage.interfaces.maxima import reduce_load_Maxima
sage: reduce_load_Maxima()
Maxima
```

sage.interfaces.maxima. reduce_load_Maxima_function (parent, defn, args, latex)
Unpickle a Maxima function.

```
sage: from sage.interfaces.maxima import reduce_load_Maxima_function
sage: f = maxima.function('x,y','sin(x+y)')
sage: _,args = f.__reduce__()
sage: g = reduce_load_Maxima_function(*args)
sage: g == f
True
```

ABSTRACT INTERFACE TO MAXIMA

Maxima is a free GPL'd general purpose computer algebra system whose development started in 1968 at MIT. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Maxima has implementations of many functions relating to the invariant theory of the symmetric group S_n . (However, the commands for group invariants, and the corresponding Maxima documentation, are in French.) For many links to Maxima documentation see http://maxima.sourceforge.net/docs.shtml/.

AUTHORS:

- William Stein (2005-12): Initial version
- David Joyner: Improved documentation
- William Stein (2006-01-08): Fixed bug in parsing
- William Stein (2006-02-22): comparisons (following suggestion of David Joyner)
- William Stein (2006-02-24): *greatly* improved robustness by adding sequence numbers to IO bracketing in _eval_line
- Robert Bradshaw, Nils Bruin, Jean-Pierre Flori (2010,2011): Binary library interface

This is an abstract class implementing the functions shared between the Pexpect and library interfaces to Maxima.

Abstract interface to Maxima.

INPUT:

```
•name - string
```

OUTPUT: the interface

EXAMPLES:

This class should not be instantiated directly, but through its subclasses Maxima (Pexpect interface) or MaximaLib (library interface):

```
sage: m = Maxima()
sage: from sage.interfaces.maxima_abstract import MaximaAbstract
sage: isinstance(m, MaximaAbstract)
True
```

chdir (dir)

Change Maxima's current working directory.

INPUT:

```
•dir - string
```

OUTPUT: none

EXAMPLES:

```
sage: maxima.chdir('/')
```

completions (s, verbose=True)

Return all commands that complete the command starting with the string s. This is like typing s[tab] in the Maxima interpreter.

INPUT:

```
•s - string
```

•verbose - boolean (default: True)

OUTPUT: array of strings

EXAMPLES:

```
sage: sorted(maxima.completions('gc', verbose=False))
['gcd', 'gcdex', 'gcfactor', 'gctime']
```

console ()

Start the interactive Maxima console. This is a completely separate maxima session from this interface. To interact with this session, you should instead use maxima.interact().

INPUT: none

OUTPUT: none

EXAMPLES:

```
sage: maxima.console() # not tested (since we can't)
Maxima 5.34.1 http://maxima.sourceforge.net
Using Lisp ECL 13.5.1
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

```
sage: maxima.interact() # this is not tested either
  --> Switching to Maxima <--
maxima: 2+2
4
maxima:
  --> Exiting back to Sage <--</pre>
```

cputime (t=None)

Returns the amount of CPU time that this Maxima session has used.

INPUT:

•t - float (default: None); If $var\{t\}$ is not None, then it returns the difference between the current CPU time and $var\{t\}$.

OUTPUT: float

EXAMPLES: sage: t = maxima.cputime() sage: $_ = maxima.de_solve('diff(y,x,2) + 3*x = y', ['x','y'], [1,1,1])$ sage: maxima.cputime(t) # output random 0.568913

de_solve (de, vars, ics=None)

Solves a 1st or 2nd order ordinary differential equation (ODE) in two variables, possibly with initial conditions.

INPUT:

- •de a string representing the ODE
- •vars a list of strings representing the two variables.
- •ics a triple of numbers [a,b1,b2] representing y(a)=b1, y'(a)=b2

EXAMPLES:

```
sage: maxima.de_solve('diff(y,x,2) + 3*x = y', ['x','y'], [1,1,1])
y=3*x-2*%e^(x-1)
sage: maxima.de_solve('diff(y,x,2) + 3*x = y', ['x','y'])
y=%k1*%e^x+%k2*%e^-x+3*x
sage: maxima.de_solve('diff(y,x) + 3*x = y', ['x','y'])
y=(%c-3*(-x-1)*%e^-x)*%e^x
sage: maxima.de_solve('diff(y,x) + 3*x = y', ['x','y'], [1,1])
y=-%e^-1*(5*%e^x-3*%e*x-3*%e)
```

de_solve_laplace (de, vars, ics=None)

Solves an ordinary differential equation (ODE) using Laplace transforms.

INPUT:

```
•de - a string representing the ODE (e.g., de = "diff(f(x),x,2)=diff(f(x),x)+sin(x)")
```

•vars - a list of strings representing the variables (e.g., vars = ["x","f"])

•ics - a list of numbers representing initial conditions, with symbols allowed which are represented by strings (eg, f(0)=1, f'(0)=2 is ics = [0,1,2])

EXAMPLES:

```
sage: maxima.clear('x'); maxima.clear('f')
sage: maxima.de_solve_laplace("diff(f(x),x,2) = 2*diff(f(x),x)-f(x)", ["x","f

\(\to \''), [0,1,2]\)
f(x)=x*%e^x+%e^x
```

Note: The second equation sets the values of f(0) and f'(0) in Maxima, so subsequent ODEs involving these variables will have these initial conditions automatically imposed.

demo(s)

Run Maxima's demo for s.

```
INPUT:
```

•s - string

OUTPUT: none

EXAMPLES:

```
sage: maxima.demo('cf') # not tested
read and interpret file: .../share/maxima/5.34.1/demo/cf.dem

At the '_' prompt, type ';' and <enter> to get next demonstration.
frac1:cf([1,2,3,4])
...
```

describe(s)

Return Maxima's help for s.

INPUT:

•s - string

OUTPUT:

Maxima's help for s

EXAMPLES:

```
sage: maxima.help('gcd')
-- Function: gcd (<p_1>, <p_2>, <x_1>, ...)
...
```

example (s)

Return Maxima's examples for s.

INPUT:

•s - string

OUTPUT:

Maxima's examples for s

EXAMPLES:

function (args, defn, rep=None, latex=None)

Return the Maxima function with given arguments and definition.

INPUT:

```
•args - a string with variable names separated by commas
```

•defn - a string (or Maxima expression) that defines a function of the arguments in Maxima.

•rep - an optional string; if given, this is how the function will print.

OUTPUT: Maxima function

EXAMPLES:

```
sage: f = maxima.function('x', 'sin(x)')
sage: f(3.2)  # abs tol 2e-16
-0.058374143427579909
sage: f = maxima.function('x,y', 'sin(x)+cos(y)')
sage: f(2, 3.5)  # abs tol 2e-16
sin(2)-0.9364566872907963
sage: f
sin(x)+cos(y)
```

```
sage: g = f.integrate('z')
sage: g
(cos(y)+sin(x))*z
sage: g(1,2,3)
3*(cos(2)+sin(1))
```

The function definition can be a Maxima object:

```
sage: an_expr = maxima('sin(x)*gamma(x)')
sage: t = maxima.function('x', an_expr)
sage: t
gamma(x)*sin(x)
sage: t(2)
sin(2)
sage: float(t(2))
0.9092974268256817
sage: loads(t.dumps())
gamma(x)*sin(x)
```

help(s)

Return Maxima's help for s.

INPUT:

•s - string

OUTPUT:

Maxima's help for s

EXAMPLES:

```
sage: maxima.help('gcd')
-- Function: gcd (<p_1>, <p_2>, <x_1>, ...)
...
```

```
plot2d ( *args)
```

Plot a 2d graph using Maxima / gnuplot.

maxima.plot2d(f, '[var, min, max]', options)

INPUT:

•f - a string representing a function (such as f="sin(x)") [var, xmin, xmax]

•options - an optional string representing plot2d options in gnuplot format

EXAMPLES:

```
sage: maxima.plot2d('sin(x)','[x,-5,5]') # not tested
sage: opts = '[gnuplot_term, ps], [gnuplot_out_file, "sin-plot.eps"]'
sage: maxima.plot2d('sin(x)','[x,-5,5]',opts) # not tested
```

The eps file is saved in the current directory.

```
plot2d_parametric ( r, var, trange, nticks=50, options=None)
```

Plot r = [x(t), y(t)] for t = tmin...tmax using gnuplot with options.

INPUT:

•r - a string representing a function (such as r="[x(t),y(t)]")

•var - a string representing the variable (such as var = "t")

•trange - [tmin, tmax] are numbers with tmintmax

•nticks - int (default: 50)

•options - an optional string representing plot2d options in gnuplot format

EXAMPLES:

The eps file is saved to the current working directory.

Here is another fun plot:

plot3d (*args)

Plot a 3d graph using Maxima / gnuplot.

maxima.plot3d(f, '[x, xmin, xmax]', '[y, ymin, ymax]', '[grid, nx, ny]', options)

INPUT:

- •f a string representing a function (such as f="sin(x)") [var, min, max]
- •args should be of the form '[x, xmin, xmax]', '[y, ymin, ymax]', '[grid, nx, ny]', options

```
sage: maxima.plot3d('1 + x^3 - y^2', '[x,-2,2]', '[y,-2,2]', '[grid,12,12]') _
    # not tested
sage: maxima.plot3d('sin(x)*cos(y)', '[x,-2,2]', '[y,-2,2]', '[grid,30,30]') _
    # not tested
sage: opts = '[gnuplot_term, ps], [gnuplot_out_file, "sin-plot.eps"]'
sage: maxima.plot3d('sin(x+y)', '[x,-5,5]', '[y,-1,1]', opts) # not tested
```

The eps file is saved in the current working directory.

```
plot3d_parametric ( r, vars, urange, vrange, options=None)
```

Plot a 3d parametric graph with r=(x,y,z), x=x(u,v), y=y(u,v), z=z(u,v), for u=umin...umax, v=vmin...vmax using gnuplot with options.

INPUT:

- •x, y, z a string representing a function (such as x="u2+v2", ...) vars is a list or two strings representing variables (such as vars = ["u","v"])
- •urange [umin, umax]
- •vrange [vmin, vmax] are lists of numbers with umin umax, vmin vmax
- •options optional string representing plot2d options in gnuplot format

OUTPUT: displays a plot on screen or saves to a file

EXAMPLES:

The eps file is saved in the current working directory.

Here is a torus:

Here is a Möbius strip:

plot_list (ptsx, ptsy, options=None)

Plots a curve determined by a sequence of points.

INPUT:

```
•ptsx - [x1,...,xn], where the xi and yi are real,
```

```
•ptsy - [y1,...,yn]
```

options - a string representing maxima plot2d options.

The points are (x1,y1), (x2,y2), etc.

This function requires maxima 5.9.2 or newer.

Note: More that 150 points can sometimes lead to the program hanging. Why?

plot_multilist (pts_list, options=None)

Plots a list of list of points pts_list=[pts1,pts2,...,ptsn], where each ptsi is of the form [[x1,y1],...,[xn,yn]] x's must be integers and y's reals options is a string representing maxima plot2d options.

INPUT:

•pts_1st - list of points; each point must be of the form [x,y] where x is an integer and y is a real

•var - string; representing Maxima's plot2d options

Requires maxima 5.9.2 at least.

Note: More that 150 points can sometimes lead to the program hanging.

EXAMPLES:

```
sage: xx = [i/10.0 \text{ for } i \text{ in } range (-10,10)]
sage: yy = [i/10.0 \text{ for } i \text{ in } range (-10,10)]
sage: x0 = [0 \text{ for } i \text{ in } range (-10,10)]
sage: y0 = [0 \text{ for } i \text{ in } range (-10,10)]
sage: zeta_ptsx1 = [ (pari(1/2+i*I/10).zeta().real()).precision(1) for i in_
→range (10)]
sage: zeta_ptsy1 = [(pari(1/2+i*I/10).zeta().imag()).precision(1) for i in.
\rightarrowrange (10)]
sage: maxima.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]])
→# not tested
sage: zeta_ptsx1 = [ (pari(1/2+i*I/10).zeta().real()).precision(1) for i in,
\rightarrowrange (10,150)]
sage: zeta_ptsy1 = [ (pari(1/2+i*I/10).zeta().imag()).precision(1) for i in,
\rightarrowrange (10,150)]
sage: maxima.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]])
⇒not tested
sage: opts='[gnuplot_preamble, "set nokey"]'
sage: maxima.plot_multilist([[zeta_ptsx1,zeta_ptsy1],[xx,y0],[x0,yy]],opts)
→ # not tested
```

solve_linear (eqns, vars)

Wraps maxima's linsolve.

INPUT:

- •eqns a list of m strings; each representing a linear question in m = n variables
- •vars a list of n strings; each representing a variable

```
sage: eqns = ["x + z = y", "2*a*x - y = 2*a^2", "y - 2*z = 2"]
sage: vars = ["x", "y", "z"]
```

```
sage: maxima.solve_linear(eqns, vars)
[x=a+1, y=2*a, z=a-1]
```

unit_quadratic_integer (n)

Finds a unit of the ring of integers of the quadratic number field $\mathbf{Q}(\sqrt{n})$, n>1, using the qunit maxima command.

INPUT:

•n - an integer

EXAMPLES:

```
sage: u = maxima.unit_quadratic_integer(101); u
a + 10
sage: u.parent()
Number Field in a with defining polynomial x^2 - 101
sage: u = maxima.unit_quadratic_integer(13)
sage: u
5*a + 18
sage: u.parent()
Number Field in a with defining polynomial x^2 - 13
```

version ()

Return the version of Maxima that Sage includes.

INPUT: none **OUTPUT**: none

EXAMPLES:

```
sage: maxima.version()
```

```
'5.35.1'
```

class sage.interfaces.maxima_abstract. MaximaAbstractElement (parent,

```
is_name=False,
                                                          name=None)
                  sage.interfaces.tab_completion.ExtraTabCompletion
sage.interfaces.interface.InterfaceElement
```

Element of Maxima through an abstract interface.

EXAMPLES:

Elements of this class should not be created directly. The targeted parent of a concrete inherited class should be used instead:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: xp = maxima(x)
sage: type(xp)
<class 'sage.interfaces.maxima.MaximaElement'>
sage: xl = maxima_lib(x)
sage: type(x1)
<class 'sage.interfaces.maxima_lib.MaximaLibElement'>
```

bool ()

Convert self into a boolean.

INPUT: none

value,

OUTPUT: boolean

EXAMPLES:

```
sage: maxima(0).bool()
False
sage: maxima(1).bool()
True
```

comma (args)

Form the expression that would be written 'self, args' in Maxima.

INPUT:

```
•args - string
```

OUTPUT: Maxima object

EXAMPLES:

```
sage: maxima('sqrt(2) + I').comma('numer')
I+1.41421356237309...
sage: maxima('sqrt(2) + I*a').comma('a=5')
5*I+sqrt(2)
```

derivative (var='x', n=1)

Return the n-th derivative of self.

INPUT:

```
•var - variable (default: 'x')
```

•n - integer (default: 1)

OUTPUT: n-th derivative of self with respect to the variable var

EXAMPLES:

```
sage: f = maxima('x^2')
sage: f.diff()
2*x
sage: f.diff('x')
2*x
sage: f.diff('x', 2)
2
sage: maxima('sin(x^2)').diff('x',4)
16*x^4*sin(x^2)-12*sin(x^2)-48*x^2*cos(x^2)
```

```
sage: f = maxima('x^2 + 17*y^2')
sage: f.diff('x')
34*y*'diff(y,x,1)+2*x
sage: f.diff('y')
34*y
```

diff(var='x', n=1)

Return the n-th derivative of self.

INPUT:

```
•var - variable (default: 'x')
```

•n - integer (default: 1)

OUTPUT: n-th derivative of self with respect to the variable var

EXAMPLES:

```
sage: f = maxima('x^2')
sage: f.diff()
2*x
sage: f.diff('x')
2*x
sage: f.diff('x', 2)
2
sage: maxima('sin(x^2)').diff('x', 4)
16*x^4*sin(x^2)-12*sin(x^2)-48*x^2*cos(x^2)
```

```
sage: f = maxima('x^2 + 17*y^2')
sage: f.diff('x')
34*y*'diff(y,x,1)+2*x
sage: f.diff('y')
34*y
```

dot (other)

Implements the notation self . other.

INPUT:

•other - matrix; argument to dot.

OUTPUT: Maxima matrix

EXAMPLES:

```
sage: A = maxima('matrix ([a1], [a2])')
sage: B = maxima('matrix ([b1, b2])')
sage: A.dot(B)
matrix([a1*b1,a1*b2], [a2*b1,a2*b2])
```

imag()

Return the imaginary part of this Maxima element.

INPUT: none

OUTPUT: Maxima real

EXAMPLES:

```
sage: maxima('2 + (2/3)*%i').imag()
2/3
```

integral (var='x', min=None, max=None)

Return the integral of self with respect to the variable x.

INPUT:

```
•var - variable
```

•min - default: None

•max - default: None

OUTPUT:

•the definite integral if xmin is not None

•an indefinite integral otherwise

EXAMPLES:

```
sage: maxima('x^2+1').integral()
x^3/3+x
sage: maxima('x^2+ 1 + y^2').integral('y')
y^3/3+x^2*y+y
sage: maxima('x / (x^2+1)').integral()
log(x^2+1)/2
sage: maxima('1/(x^2+1)').integral()
atan(x)
sage: maxima('1/(x^2+1)').integral('x', 0, infinity)
%pi/2
sage: maxima('x/(x^2+1)').integral('x', -1, 1)
0
```

```
sage: f = maxima('exp(x^2)').integral('x',0,1); f
-sqrt(%pi)*%i*erf(%i)/2
sage: f.numer()
1.46265174590718...
```

integrate (var='x', min=None, max=None)

Return the integral of self with respect to the variable x.

INPUT:

- •var variable
- •min default: None
- •max default: None

OUTPUT:

- •the definite integral if xmin is not None
- •an indefinite integral otherwise

EXAMPLES:

```
sage: maxima('x^2+1').integral()
x^3/3+x
sage: maxima('x^2+ 1 + y^2').integral('y')
y^3/3+x^2*y+y
sage: maxima('x / (x^2+1)').integral()
log(x^2+1)/2
sage: maxima('1/(x^2+1)').integral()
atan(x)
sage: maxima('1/(x^2+1)').integral('x', 0, infinity)
%pi/2
sage: maxima('x/(x^2+1)').integral('x', -1, 1)
0
```

```
sage: f = maxima('exp(x^2)').integral('x',0,1); f
-sqrt(%pi)*%i*erf(%i)/2
sage: f.numer()
1.46265174590718...
```

nintegral (var='x', a=0, b=1, $desired_relative_error='1e-8'$, $maximum_num_subintervals=200$) Return a numerical approximation to the integral of self from a to b.

INPUT:

- •var variable to integrate with respect to
- •a lower endpoint of integration
- •b upper endpoint of integration
- •desired_relative_error (default: '1e-8') the desired relative error
- •maximum num subintervals (default: 200) maxima number of subintervals

OUTPUT:

- •approximation to the integral
- •estimated absolute error of the approximation
- •the number of integrand evaluations
- •an error code:
 - -0 no problems were encountered
 - -1 too many subintervals were done
 - -2 excessive roundoff error
 - -3 extremely bad integrand behavior
 - -4 failed to converge
 - -5 integral is probably divergent or slowly convergent
 - -6 the input is invalid

EXAMPLES:

```
sage: maxima('exp(-sqrt(x))').nintegral('x',0,1)
(0.5284822353142306, 0.41633141378838...e-10, 231, 0)
```

Note that GP also does numerical integration, and can do so to very high precision very quickly:

numer ()

Return numerical approximation to self as a Maxima object.

INPUT: none

OUTPUT: Maxima object

```
sage: a = maxima('sqrt(2)').numer(); a
1.41421356237309...
sage: type(a)
<class 'sage.interfaces.maxima.MaximaElement'>
```

partial_fraction_decomposition (var='x')

Return the partial fraction decomposition of self with respect to the variable var.

INPUT:

```
•var - string
```

OUTPUT: Maxima object

EXAMPLES:

real ()

Return the real part of this Maxima element.

INPUT: none

OUTPUT: Maxima real

EXAMPLES:

```
sage: maxima('2 + (2/3)*%i').real()
2
```

str ()

Return string representation of this Maxima object.

INPUT: none

OUTPUT: string

EXAMPLES:

```
sage: maxima('sqrt(2) + 1/3').str()
'sqrt(2)+1/3'
```

subst (val)

Substitute a value or several values into this Maxima object.

INPUT:

•val - string representing substitution(s) to perform

OUTPUT: Maxima object

```
sage: maxima('a^2 + 3*a + b').subst('b=2')
a^2+3*a+2
sage: maxima('a^2 + 3*a + b').subst('a=17')
b+340
sage: maxima('a^2 + 3*a + b').subst('a=17, b=2')
342
```

Bases: sage.interfaces.maxima_abstract.MaximaAbstractElement

Create a Maxima function with the parent parent, name name, definition defn, arguments args and latex representation latex.

INPUT:

- •parent an instance of a concrete Maxima interface
- •name string
- •defn string
- •args string; comma separated names of arguments
- •latex string

OUTPUT: Maxima function

EXAMPLES:

```
sage: maxima.function('x,y','e^cos(x)')
e^cos(x)
```

arguments (split=True)

Returns the arguments of this Maxima function.

INPUT:

•split - boolean; if True return a tuple of strings, otherwise return a string of comma-separated arguments

OUTPUT:

- •a string if split is False
- •a list of strings if split is True

EXAMPLES:

```
sage: f = maxima.function('x,y','sin(x+y)')
sage: f.arguments()
['x', 'y']
sage: f.arguments(split=False)
'x,y'
sage: f = maxima.function('', 'sin(x)')
sage: f.arguments()
[]
```

definition ()

Returns the definition of this Maxima function as a string.

INPUT: none

OUTPUT: string

```
sage: f = maxima.function('x,y','sin(x+y)')
sage: f.definition()
'sin(x+y)'
```

```
integral (var)
```

Returns the integral of self with respect to the variable var.

INPUT:

•var - a variable

OUTPUT: Maxima function

Note that integrate is an alias of integral.

EXAMPLES:

```
sage: x,y = var('x,y')
sage: f = maxima.function('x','sin(x)')
sage: f.integral(x)
-cos(x)
sage: f.integral(y)
sin(x)*y
```

integrate (var)

Returns the integral of self with respect to the variable var.

INPUT:

•var - a variable

OUTPUT: Maxima function

Note that integrate is an alias of integral.

EXAMPLES:

```
sage: x,y = var('x,y')
sage: f = maxima.function('x','sin(x)')
sage: f.integral(x)
-cos(x)
sage: f.integral(y)
sin(x)*y
```

```
{\bf class} \; {\tt sage.interfaces.maxima\_abstract.} \; {\tt MaximaAbstractFunction} \; ( \; {\it parent, name})
```

 $Bases: \ sage.interfaces.interface.InterfaceFunction\\$

 ${f class}$ sage.interfaces.maxima_abstract. ${f MaximaAbstractFunctionElement}$ (obj,

Bases: sage.interfaces.interface.InterfaceFunctionElement

sage.interfaces.maxima_abstract. maxima_console ()

Spawn a new Maxima command-line session.

EXAMPLES:

```
sage: from sage.interfaces.maxima_abstract import maxima_console
sage: maxima_console() # not tested
Maxima 5.34.1 http://maxima.sourceforge.net
...
```

```
{\tt sage.interfaces.maxima\_abstract.} \ {\tt maxima\_version} \ (\ )
```

Return Maxima version.

Currently this calls a new copy of Maxima.

name)

EXAMPLES:

```
sage: from sage.interfaces.maxima_abstract import maxima_version
sage: maxima_version()
'5.35.1'
```

Unpickle a Maxima function.

Sage Reference Manual: Interpreter Interfaces, Release 7.5		

CHAPTER

TWENTYSEVEN

LIBRARY INTERFACE TO MAXIMA

Maxima is a free GPL'd general purpose computer algebra system whose development started in 1968 at MIT. It contains symbolic manipulation algorithms, as well as implementations of special functions, including elliptic functions and generalized hypergeometric functions. Moreover, Maxima has implementations of many functions relating to the invariant theory of the symmetric group S_n . (However, the commands for group invariants, and the corresponding Maxima documentation, are in French.) For many links to Maxima documentation, see http://maxima.sourceforge.net/documentation.html.

AUTHORS:

- William Stein (2005-12): Initial version
- David Joyner: Improved documentation
- William Stein (2006-01-08): Fixed bug in parsing
- William Stein (2006-02-22): comparisons (following suggestion of David Joyner)
- William Stein (2006-02-24): *greatly* improved robustness by adding sequence numbers to IO bracketing in _eval_line
- Robert Bradshaw, Nils Bruin, Jean-Pierre Flori (2010,2011): Binary library interface

For this interface, Maxima is loaded into ECL which is itself loaded as a C library in Sage. Translations between Sage and Maxima objects (which are nothing but wrappers to ECL objects) is made as much as possible directly, but falls back to the string based conversion used by the classical Maxima Pexpect interface in case no new implementation has been made.

This interface is the one used for calculus by Sage and is accessible as maxima_calculus:

```
sage: maxima_calculus
Maxima_lib
```

Only one instance of this interface can be instantiated, so the user should not try to instantiate another one, which is anyway set to raise an error:

```
sage: from sage.interfaces.maxima_lib import MaximaLib
sage: MaximaLib()
Traceback (most recent call last):
...
RuntimeError: Maxima interface in library mode can only be instantiated once
```

Changed besselexpand to true in init_code – automatically simplify bessel functions to trig functions when appropriate when true. Examples:

For some infinite sums, a closed expression can be found. By default, "maxima" is used for that:

```
sage: x,n,k = var("x","n","k")
sage: sum((-x)^n/(factorial(n)*factorial(n+3/2)),n,0,00)
-1/2*(2*x*cos(2*sqrt(x)) - sqrt(x)*sin(2*sqrt(x)))/(sqrt(pi)*x^2)
```

Maxima has some flags that affect how the result gets simplified (By default, besselexpand was set to false in Maxima):

```
sage: maxima_calculus("besselexpand:false")
false
sage: x,n,k = var("x","n","k")
sage: sum((-x)^n/(factorial(n)*factorial(n+3/2)),n,0,00)
bessel_J(3/2, 2*sqrt(x))/x^(3/4)
sage: maxima_calculus("besselexpand:true")
true
```

class sage.interfaces.maxima_lib. MaximaLib

Bases: sage.interfaces.maxima_abstract.MaximaAbstract

Interface to Maxima as a Library.

INPUT: none

OUTPUT: Maxima interface as a Library

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import MaximaLib, maxima_lib
sage: isinstance(maxima_lib,MaximaLib)
True
```

Only one such interface can be instantiated:

```
sage: MaximaLib()
Traceback (most recent call last):
...
RuntimeError: Maxima interface in library mode can only
be instantiated once
```

clear (var)

Clear the variable named var.

INPUT:

•var - string

OUTPUT: none

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
sage: maxima_lib.clear('xxxxx')
sage: maxima_lib.get('xxxxx')
```

eval (line, locals=None, reformat=True, **kwds)

Evaluate the line in Maxima.

INPUT:

- •line string; text to evaluate
- •locals None (ignored); this is used for compatibility with the Sage notebook's generic system interface.
- •reformat boolean; whether to strip output or not
- •**kwds All other arguments are currently ignored.

OUTPUT: string representing Maxima output

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib._eval_line('1+1')
'2'
sage: maxima_lib._eval_line('1+1;')
'2'
sage: maxima_lib._eval_line('1+1$')
''
sage: maxima_lib._eval_line('randvar : cos(x)+sin(y)$')
''
sage: maxima_lib._eval_line('randvar')
'sin(y)+cos(x)'
```

get (var)

Get the string value of the variable var.

INPUT:

•var - string

OUTPUT: string

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
```

lisp (cmd)

Send a lisp command to maxima.

INPUT:

•cmd - string

OUTPUT: ECL object

Note: The output of this command is very raw - not pretty.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.lisp("(+ 2 17)")
<ECL: 19>
```

set (var, value)

Set the variable var to the given value.

INPUT:

```
var - stringvalue - stringOUTPUT: noneEXAMPLES:
```

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib.set('xxxxx', '2')
sage: maxima_lib.get('xxxxx')
'2'
```

sr_integral (*args)

Helper function to wrap calculus use of Maxima's integration.

TESTS:

```
sage: a, b=var('a, b')
sage: integrate (1/(x^3 * (a+b*x)^(1/3)), x)
Traceback (most recent call last):
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(a>0)', see
`assume?` for more details)
Is a positive or negative?
sage: assume(a>0)
sage: integrate (1/(x^3 * (a+b*x)^(1/3)), x)
2/9*sqrt(3)*b^2*arctan(1/3*sqrt(3)*(2*(b*x + a)^(1/3) + a^(1/3))/a^(1/3))/a^
\rightarrow (7/3) - 1/9*b^2*log((b*x + a)^(2/3) + (b*x + a)^(1/3)*a^(1/3) + a^(2/3))/a^
\rightarrow (7/3) + 2/9*b^2*log((b*x + a)^(1/3) - a^(1/3))/a^(7/3) + 1/6*(4*(b*x + a)^
(5/3)*b^2 - 7*(b*x + a)^(2/3)*a*b^2/(b*x + a)^2*a^2 - 2*(b*x + a)*a^3 + a^3
sage: var('x, n')
(x, n)
sage: integral(x^n,x)
Traceback (most recent call last):
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(n>0)',
see `assume?` for more details)
Is n equal to -1?
sage: assume (n+1>0)
sage: integral(x^n,x)
x^{(n + 1)}/(n + 1)
sage: forget()
sage: assumptions() # Check the assumptions really were forgotten
```

Make sure the abs_integrate package is being used, trac ticket #11483. The following are examples from the Maxima abs_integrate documentation:

```
sage: integrate(abs(x), x)
1/2*x*abs(x)
```

```
sage: integrate(sgn(x) - sgn(1-x), x)
abs(x - 1) + abs(x)
```

```
sage: integrate(1 / (1 + abs(x-5)), x, -5, 6) log(11) + log(2)
```

```
sage: integrate(1/(1 + abs(x)), x) 1/2*(\log(x + 1) + \log(-x + 1))*sgn(x) + 1/2*log(x + 1) - 1/2*log(-x + 1)
```

```
sage: integrate(cos(x + abs(x)), x)
-1/2*x*sgn(x) + 1/4*(sgn(x) + 1)*sin(2*x) + 1/2*x
```

The last example relies on the following simplification:

```
sage: maxima("realpart(signum(x))")
signum(x)
```

An example from sage-support thread e641001f8b8d1129:

```
sage: f = e^(-x^2/2)/sqrt(2*pi) * sgn(x-1)
sage: integrate(f, x, -Infinity, Infinity)
-erf(1/2*sqrt(2))
```

From trac ticket #8624:

```
sage: integral(abs(cos(x))*sin(x),(x,pi/2,pi))
1/2
```

```
sage: integrate(sqrt(x + sqrt(x)), x).canonicalize_radical() 1/12*((8*x - 3)*x^{(1/4)} + 2*x^{(3/4)})*sqrt(sqrt(x) + 1) + 1/8*log(sqrt(sqrt(x) + 1) + x^{(1/4)}) - 1/8*log(sqrt(sqrt(x) + 1) - x^{(1/4)})
```

And trac ticket #11594:

```
sage: integrate(abs(x^2 - 1), x, -2, 2)
```

This definite integral returned zero (incorrectly) in at least Maxima 5.23. The correct answer is now given (trac ticket #11591):

```
sage: f = (x^2) * exp(x) / (1 + exp(x))^2
sage: integrate(f, (x, -infinity, infinity))
1/3 * pi^2
```

Sometimes one needs different simplification settings, such as radexpand, to compute an integral (see trac ticket #10955):

```
sage: f = sqrt(x + 1/x^2)
sage: maxima = sage.calculus.calculus.maxima
sage: maxima('radexpand')
true
sage: integrate(f, x)
integrate(sqrt(x + 1/x^2), x)
sage: maxima('radexpand: all')
all
sage: g = integrate(f, x); g
2/3*sqrt(x^3 + 1) - 1/3*log(sqrt(x^3 + 1) + 1) + 1/3*log(sqrt(x^3 + 1) - 1)
sage: (f - g.diff(x)).canonicalize_radical()
0
```

```
sage: maxima('radexpand: true')
true
```

The following integral was computed incorrectly in versions of Maxima before 5.27 (see trac ticket #12947):

```
sage: a = integrate(x*cos(x^3),(x,0,1/2)).n()
sage: a.real()
0.124756040961038
sage: a.imag().abs() < 3e-17
True</pre>
```

sr_limit (expr, v, a, dir=None)

Helper function to wrap calculus use of Maxima's limits.

TESTS:

```
sage: f = (1+1/x)^x
sage: limit(f, x = oo)
sage: limit(f, x = 5)
7776/3125
sage: limit(f, x = 1.2)
2.06961575467...
sage: var('a')
sage: limit (x^a, x=0)
Traceback (most recent call last):
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation
*may* help (example of legal syntax is 'assume(a>0)', see `assume?`
for more details)
Is a positive, negative or zero?
sage: assume(a>0)
sage: limit (x^a, x=0)
Traceback (most recent call last):
ValueError: Computation failed ...
Is a an integer?
sage: assume(a,'integer')
sage: assume(a,'even') # Yes, Maxima will ask this too
sage: limit (x^a, x=0)
0
sage: forget()
sage: assumptions() # check the assumptions were really forgotten
[]
```

The second limit below was computed incorrectly prior to Maxima 5.24 (trac ticket #10868):

```
sage: f(n) = 2 + 1/factorial(n)
sage: limit(f(n), n=infinity)
2
sage: limit(1/f(n), n=infinity)
1/2
```

The limit below was computed incorrectly prior to Maxima 5.30 (see trac ticket #13526):

```
sage: n = var('n')
sage: l = (3^n + (-2)^n) / (3^(n+1) + (-2)^(n+1))
sage: l.limit(n=oo)
1/3
```

The following limit computation used to incorrectly return 0 or infinity, depending on the domain (see trac ticket #15033):

```
sage: m = sage.calculus.calculus.maxima
sage: _ = m.eval('domain: real')  # much faster than 'domain: complex'
sage: limit(gamma(x + 1/2)/(sqrt(x)*gamma(x)), x=infinity)
1
sage: _ = m.eval('domain: complex')
```

sr_sum (*args)

Helper function to wrap calculus use of Maxima's summation.

TESTS:

Check that trac ticket #16224 is fixed:

```
sage: k = var('k')
sage: sum(x^(2*k)/factorial(2*k), k, 0, oo).canonicalize_radical()
cosh(x)
```

```
sage: x, y, k, n = var('x, y, k, n')
sage: sum(binomial(n,k) * x^k * y^n(n-k), k, 0, n)
(x + y)^n
sage: q, a = var('q, a')
sage: sum(a*q^k, k, 0, oo)
Traceback (most recent call last):
ValueError: Computation failed since Maxima requested additional
constraints; using the 'assume' command before evaluation *may* help
(example of legal syntax is 'assume(abs(q)-1>0)', see `assume?`
for more details)
Is abs(q)-1 positive, negative or zero?
sage: assume (q > 1)
sage: sum(a*q^k, k, 0, oo)
Traceback (most recent call last):
. . .
ValueError: Sum is divergent.
sage: forget()
sage: assume(abs(q) < 1)</pre>
sage: sum(a*q^k, k, 0, oo)
-a/(q - 1)
sage: forget()
sage: assumptions() # check the assumptions were really forgotten
[]
```

Taking the sum of all natural numbers informs us that the sum is divergent. Maxima (before 5.29.1) used to ask questions about m, leading to a different error (see trac ticket #11990):

```
sage: m = var('m')
sage: sum(m, m, 0, infinity)
Traceback (most recent call last):
...
ValueError: Sum is divergent.
```

An error with an infinite sum in Maxima (before 5.30.0, see trac ticket #13712):

```
sage: n = var('n')
sage: sum(1/((2*n-1)^2*(2*n+1)^2*(2*n+3)^2), n, 0, oo)
3/256*pi^2
```

Maxima correctly detects division by zero in a symbolic sum (see trac ticket #11894):

Similar situation for trac ticket #12410:

```
sage: x = var('x')
sage: sum(1/x*(-1)^x, x, 0, oo)
Traceback (most recent call last):
...
RuntimeError: ECL says: Error executing code in Maxima: Zero to negative
→power computed.
```

sr_tlimit (expr, v, a, dir=None)

Helper function to wrap calculus use of Maxima's Taylor series limits.

TESTS:

```
sage: f = (1+1/x)^x
sage: limit(f, x = I, taylor=True)
(-I + 1)^I
```

Bases: sage.interfaces.maxima_abstract.MaximaAbstractElement

Element of Maxima through library interface.

EXAMPLES:

Elements of this class should not be created directly. The targeted parent should be used instead:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib(4)
4
sage: maxima_lib(log(x))
log(_SAGE_VAR_x)
```

display2d (onscreen=True)

Return the 2d representation of this Maxima object.

INPUT:

•onscreen - boolean (default: True); whether to print or return

OUTPUT:

The representation is printed if onscreen is set to True and returned as a string otherwise.

ecl ()

Return the underlying ECL object of this MaximaLib object.

INPUT: none

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: maxima_lib(x+cos(19)).ecl()
<ECL: ((MPLUS SIMP) ((%COS SIMP) 19) |$_SAGE_VAR_x|)>
```

to_poly_solve (vars, options='')

Use Maxima's to_poly_solver package.

INPUT:

- •vars symbolic expressions
- •options string (default="")

OUTPUT: Maxima object

EXAMPLES:

The zXXX below are names for arbitrary integers and subject to change:

```
sage: from sage.interfaces.maxima_lib import maxima_lib
sage: sol = maxima_lib(sin(x) == 0).to_poly_solve(x)
sage: sol.sage()
[[x == pi*z54]]
```

Bases: sage.interfaces.maxima_lib.MaximaLibElement, sage.interfaces.maxima_abstract.Maxim

Create a Maxima function. See MaximaAbstractElementFunction for full documentation.

TESTS:

```
sage: from sage.interfaces.maxima_abstract import MaximaAbstractElementFunction
sage: MaximaAbstractElementFunction == loads(dumps(MaximaAbstractElementFunction))
True
sage: f = maxima.function('x,y','sin(x+y)')
sage: f == loads(dumps(f))
True
```

```
class sage.interfaces.maxima_lib. MaximaLibFunction (parent, name)
```

Bases: sage.interfaces.maxima_abstract.MaximaAbstractFunction

class sage.interfaces.maxima_lib. MaximaLibFunctionElement (obj, name)

 $Bases: \ sage.interfaces.maxima_abstract.MaximaAbstractFunctionElement$

```
sage.interfaces.maxima_lib. dummy_integrate (expr)
```

We would like to simply tie Maxima's integrate to sage.calculus.calculus.dummy_integrate, but we're being imported there so to avoid circularity we define it here.

INPUT:

•expr - ECL object; a Maxima %INTEGRATE expression

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, dummy_integrate
sage: f = maxima_lib('f(x)').integrate('x')
sage: f.ecl()
<ECL: ((%INTEGRATE SIMP) (($F SIMP) $X) $X)>
sage: dummy_integrate(f.ecl())
integrate(f(x), x)
```

```
sage: f = maxima_lib('f(x)').integrate('x',0,10)
sage: f.ecl()
<ECL: ((%INTEGRATE SIMP) (($F SIMP) $X) $X 0 10)>
sage: dummy_integrate(f.ecl())
integrate(f(x), x, 0, 10)
```

sage.interfaces.maxima_lib. is_MaximaLibElement (x)

Returns True if x is of type MaximaLibElement.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, is_MaximaLibElement
sage: m = maxima_lib(1)
sage: is_MaximaLibElement(m)
True
sage: is_MaximaLibElement(1)
False
```

sage.interfaces.maxima_lib. max_at_to_sage (expr)

Special conversion rule for AT expressions.

INPUT:

•expr - ECL object; a Maxima AT expression

OUTPUT: symbolic expression

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_at_to_sage
sage: a=maxima_lib("'at(f(x,y,z),[x=1,y=2,z=3])")
sage: a
'at(f(x,y,z),[x=1,y=2,z=3])
sage: max_at_to_sage(a.ecl())
f(1, 2, 3)
sage: a=maxima_lib("'at(f(x,y,z),x=1)")
sage: a
'at(f(x,y,z),x=1)
sage: max_at_to_sage(a.ecl())
f(1, y, z)
```

```
sage.interfaces.maxima_lib. max_harmonic_to_sage (expr)
EXAMPLES:
```

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_to_sr
sage: c=maxima_lib(harmonic_number(x,2))
sage: c.ecl()
<ECL: (($GEN_HARMONIC_NUMBER SIMP) 2 |$_SAGE_VAR_x|)>
sage: max_to_sr(c.ecl())
harmonic_number(x, 2)
```

sage.interfaces.maxima_lib. max_to_sr (expr)

Convert a Maxima object into a symbolic expression.

INPUT:

•expr - ECL object

OUTPUT: symbolic expression

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_to_sr
sage: f = maxima_lib('f(x)')
sage: f.ecl()
<ECL: (($F SIMP) $X)>
sage: max_to_sr(f.ecl())
f(x)
```

TESTS:

```
sage: from sage.interfaces.maxima_lib import sr_to_max, max_to_sr
sage: f = function('f')(x).diff()
sage: bool(max_to_sr(sr_to_max(f)) == f)
True
```

sage.interfaces.maxima_lib. max_to_string (s)

Return the Maxima string corresponding to this ECL object.

INPUT:

•s - ECL object

OUTPUT: string

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, max_to_string
sage: ecl = maxima_lib(cos(x)).ecl()
sage: max_to_string(ecl)
'cos(_SAGE_VAR_x)'
```

sage.interfaces.maxima_lib. mdiff_to_sage (expr)

Special conversion rule for %DERIVATIVE expressions.

INPUT:

•expr - ECL object; a Maxima %DERIVATIVE expression

OUTPUT: symbolic expression

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mdiff_to_sage
sage: f = maxima_lib('f(x)').diff('x',4)
sage: f.ecl()
<ECL: ((%DERIVATIVE SIMP) (($F SIMP) $X) $X 4)>
sage: mdiff_to_sage(f.ecl())
diff(f(x), x, x, x, x)
```

sage.interfaces.maxima_lib. mlist_to_sage (expr)

Special conversion rule for MLIST expressions.

INPUT:

•expr - ECL object; a Maxima MLIST expression (i.e., a list)

OUTPUT: a Python list of converted expressions.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mlist_to_sage
sage: L=maxima_lib("[1,2,3]")
sage: L.ecl()
<ECL: ((MLIST SIMP) 1 2 3)>
sage: mlist_to_sage(L.ecl())
[1, 2, 3]
```

sage.interfaces.maxima_lib. mqapply_to_sage (expr)

Special conversion rule for MQAPPLY expressions.

INPUT:

•expr - ECL object; a Maxima MQAPPLY expression

OUTPUT: symbolic expression

MQAPPLY is used for function as li[x](y) and psi[x](y).

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mqapply_to_sage
sage: c = maxima_lib('li[2](3)')
sage: c.ecl()
<ECL: ((MQAPPLY SIMP) (($LI SIMP ARRAY) 2) 3)>
sage: mqapply_to_sage(c.ecl())
dilog(3)
```

sage.interfaces.maxima_lib. mrat_to_sage (expr)

Convert a Maxima MRAT expression to Sage SR.

INPUT:

•expr - ECL object; a Maxima MRAT expression

OUTPUT: symbolic expression

Maxima has an optimised representation for multivariate rational expressions. The easiest way to translate those to SR is by first asking Maxima to give the generic representation of the object. That is what RATDISREP does in Maxima.

```
sage: from sage.interfaces.maxima_lib import maxima_lib, mrat_to_sage
sage: var('x y z')
```

sage.interfaces.maxima_lib. parse_max_string (s)

Evaluate string in Maxima without any further simplification.

INPUT:

•s - string

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import parse_max_string
sage: parse_max_string('1+1')
<ECL: ((MPLUS) 1 1)>
```

sage.interfaces.maxima_lib. pyobject_to_max (obj)

Convert a (simple) Python object into a Maxima object.

INPUT:

•expr - Python object

OUTPUT: ECL object

Note: This uses functions defined in sage.libs.ecl.

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import pyobject_to_max
sage: pyobject_to_max(4)
<ECL: 4>
sage: pyobject_to_max('z')
<ECL: Z>
sage: var('x')
x
sage: pyobject_to_max(x)
Traceback (most recent call last):
...
TypeError: Unimplemented type for python_to_ecl
```

sage.interfaces.maxima_lib. reduce_load_MaximaLib ()

Unpickle the (unique) Maxima library interface.

```
sage: from sage.interfaces.maxima_lib import reduce_load_MaximaLib
sage: reduce_load_MaximaLib()
Maxima_lib
```

```
sage: from sage.interfaces.maxima_lib import sage_rat
sage: sage_rat(1,7)
1/7
```

```
sage.interfaces.maxima_lib. sr_to_max ( expr)
```

Convert a symbolic expression into a Maxima object.

INPUT:

•expr - symbolic expression

OUTPUT: ECL object

EXAMPLES:

```
sage: from sage.interfaces.maxima_lib import sr_to_max
sage: var('x')
x
sage: sr_to_max(x)
<ECL: $X>
sage: sr_to_max(cos(x))
<ECL: ((%COS) $X)>
sage: f = function('f')(x)
sage: sr_to_max(f.diff())
<ECL: ((%DERIVATIVE) (($F) $X) $X 1)>
```

TESTS:

We should be able to convert derivatives evaluated at a point, trac ticket #12796:

```
sage: from sage.interfaces.maxima_lib import sr_to_max, max_to_sr
sage: f = function('f')
sage: f_prime = f(x).diff(x)
sage: max_to_sr(sr_to_max(f_prime(x = 1)))
D[0](f)(1)
```

sage.interfaces.maxima_lib. stdout_to_string (s)

Evaluate command s and catch Maxima stdout (not the result of the command!) into a string.

INPUT:

•s - string; command to evaluate

OUTPUT: string

This is currently used to implement display2d().

```
sage: from sage.interfaces.maxima_lib import stdout_to_string
sage: stdout_to_string('1+1')
''
sage: stdout_to_string('disp(1+1)')
'2\n\n'
```

Sage Reference Manual: Interpreter Interfaces, Release 7.5		

CHAPTER

TWENTYEIGHT

INTERFACE TO MUPAD

AUTHOR:

- · Mike Hansen
- · William Stein

You must have the optional commercial MuPAD interpreter installed and available as the command code{mupkern} in your PATH in order to use this interface. You do not have to install any optional sage packages.

TESTS:

```
sage: mupad.package('"MuPAD-Combinat"')
                                                             # optional - mupad
sage: combinat = mupad.combinat
                                                             # optional - mupad
sage: examples = mupad.examples
                                                             # optional - mupad
sage: S = examples.SymmetricFunctions()
                                                             # optional - mupad
                                                             # optional - mupad
sage: S.s[2,1]^2
s[3, 3] + s[4, 2] + s[2, 2, 1, 1] + s[2, 2, 2] + 2 s[3, 2, 1] + s[4, 1, 1] +
s[3, 1, 1, 1]
                                                             # optional - mupad
sage: S.omega( S.s[3] )
s[1, 1, 1]
sage: s = S.s
                                                             # optional - mupad
sage: p = S.p
                                                             # optional - mupad
                                                             # optional - mupad
sage: s(s[2,1] + p[2,1])
s[2, 1] + s[3] - s[1, 1, 1]
                                                             # optional - mupad
sage: s(_)
s[2, 1] + s[3] - s[1, 1, 1]
                                                             # optional - mupad #___
sage: combinat.tableaux.list(3)
→note: the order of the result seems to depend on the version of MuPAD / MuPAD-
→Combinat
                                                  | 3 | |
                            | 3 |
                                      | 2 |
           | | 1 | 2 | 3 |, | 1 | 2 |, | 1 | 3 |, | 1 | |
           -- +---+ +---+ +---+ +---+ +---+
sage: three = mupad(3)
                                                            # optional - mupad
sage: three.combinat.tableaux.list()
                                                            # optional - mupad
                                                  +---+ --
                                                  | 3 | |
                                       | 2 |
           | | 1 | 2 | 3 |, | 1 | 2 |, | 1 | 3 |, | 1 | |
           -- +---+--+ +---+ +---+ +---+ +---+
```

```
sage: t = [1]
                                                        # optional - mupad
sage: t
                                                        # optional - mupad
                          +---+
                          | 1 | 2 | 3 |
sage: combinat.tableaux.conjugate(t)
                                                       # optional - mupad
                             | 3 |
                             1 2 1
                             +---+
                             | 1 |
                                                     # optional - mupad
sage: combinat.ribbonsTableaux.list([2,2],[1,1],2)
                    -- +---+ +---+ --
                    | | 2 | |
                    | + + +, +---+ |
                    | | 1 | | | 1
                    -- +---+ +---+ --
                                                      # optional - mupad
sage: combinat.tableaux.kAtom([2,1],3)
                           1 1 2 1
                          | +---+ |
                           | | 1 | 1 | |
                           -- +---+ --
sage: M = S.Macdonald()
                                                     # optional - mupad
sage: a = M.P[1]^2
                                                     # optional - mupad
sage: mupad.mapcoeffs(a, 'normal')
                                                     # optional - mupad
                         q - t + q t - 1
                   P[2] + ----- P[1, 1]
                            q t - 1
```

```
Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect
```

Interface to the MuPAD interpreter.

```
completions ( string, strip=False)
```

EXAMPLES:

```
sage: mupad.completions('linal') # optional - mupad
['linalg']
```

console ()

Spawn a new MuPAD command-line session.

```
sage: mupad.console() #not tested

*---* MuPAD Pro 4.0.2 -- The Open Computer Algebra System
/| /|
*----* | Copyright (c) 1997 - 2007 by SciFace Software
| *--|-* All rights reserved.
|/ |/
*----* Licensed to: ...
```

cputime (t=None)

EXAMPLES:

```
sage: t = mupad.cputime() #random, optional - MuPAD
0.116000000000001
```

eval (code, strip=True, **kwds)

EXAMPLES:

```
sage: mupad.eval('2+2') # optional - mupad
4
```

expect ()

EXAMPLES:

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: mupad.set('a', 4) # optional - mupad
sage: mupad.get('a').strip() # optional - mupad
'4'
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: mupad.set('a', 4) # optional - mupad
sage: mupad.get('a').strip() # optional - mupad
'4'
```

class sage.interfaces.mupad. MupadElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.tab_completion.ExtraTabCompletion

sage.interfaces.expect.ExpectElement

 ${f class}$ sage.interfaces.mupad. ${f MupadFunction}$ (${\it parent, name}$)

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.ExpectFunction

class sage.interfaces.mupad. MupadFunctionElement (obj, name)

Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.FunctionElement

sage.interfaces.mupad. mupad_console ()

Spawn a new MuPAD command-line session.

```
sage: from sage.interfaces.mupad import mupad_console
sage: mupad_console() #not tested

*---* MuPAD Pro 4.0.2 -- The Open Computer Algebra System
// //
```

```
*---* | Copyright (c) 1997 - 2007 by SciFace Software | *--|-* All rights reserved. | / | / | / | Licensed to: ...
```

sage.interfaces.mupad.reduce_load_mupad()

```
sage: from sage.interfaces.mupad import reduce_load_mupad
sage: reduce_load_mupad()
Mupad
```

CHAPTER

TWENTYNINE

INTERFACE TO MWRANK

sage.interfaces.mwrank. **Mwrank** (*options=''*, *server=None*, *server_tmpdir=None*) Create and return an mwrank interpreter, with given options.

INPUT:

•options - string; passed when starting mwrank. The format is:

```
-h
         help
                         prints this info and quits
        quiet turns OFF banner display and prompt
verbosity sets verbosity to n (default=1)
-q
−v n
        PARI/GP output turns ON extra PARI/GP short output (default is OFF)
-0
-p n
        precision sets precision to n decimals (default=15)
        quartic bound bound on quartic point search (default=10)
-b n
-x n
        n aux number of aux primes used for sieving (default=6)
-1
        list
                        turns ON listing of points (default ON unless v=0)
-s
        selmer_only if set, computes Selmer rank only (default: not set)
-d
                                 if set, skips the second descent for curves.
        skip_2nd_descent
→with 2-torsion (default: not set)
        sat_bd
                         upper bound on saturation primes (default=100, -1...
→for automatic)
```

EXAMPLES:

```
sage: M = Mwrank('-v 0 -l')
sage: print(M('0 0 1 -1 0'))
Curve [0,0,1,-1,0] : Rank = 1
Generator 1 is [0:-1:1]; height 0.0511114082399688
Regulator = 0.0511114082399688
```

class sage.interfaces.mwrank. Mwrank_class (options='', server=None, server_tmpdir=None)
 Bases: sage.interfaces.expect.Expect

Interface to the Mwrank interpreter.

console ()

Start the mwrank console.

EXAMPLE:

```
sage: mwrank.console() # not tested: expects console input
Program mwrank: ...
```

eval(s, **kwds)

Return mwrank's output for the given input.

INPUT:

- •s (str) a Sage object which when converted to a string gives valid input to mwrank. The conversion is done by validate_mwrank_input(). Possible formats are:
 - -a string representing exactly five integers separated by whitespace, for example '1 2 3 4 5'
 - -a string representing exactly five integers separated by commas, preceded by '[' and followed by ']' (with arbitrary whitespace), for example '[1 2 3 4 5]'
 - -a list or tuple of exactly 5 integers.

Note: If a RuntimeError exception is raised, then the mwrank interface is restarted and the command is retried once.

EXAMPLES:

```
sage: mwrank.eval('12 3 4 5 6')
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval('[12, 3, 4, 5, 6]')
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval([12, 3, 4, 5, 6])
'Curve [12,3,4,5,6] :...'
sage: mwrank.eval((12, 3, 4, 5, 6))
'Curve [12,3,4,5,6] :...'
```

sage.interfaces.mwrank.mwrank_console ()

Start the mwrank console.

EXAMPLE:

```
sage: mwrank_console() # not tested: expects console input
Program mwrank: ...
```

sage.interfaces.mwrank.validate_mwrank_input (s)

Returns a string suitable for mwrank input, or raises an error.

INPUT:

- $\bullet s$ one of the following:
 - -a list or tuple of 5 integers [a1,a2,a3,a4,a6] or (a1,a2,a3,a4,a6)
 - -a string of the form '[a1,a2,a3,a4,a6]' or 'a1 a2 a3 a4 a6' where a1, a2, a3, a4, a6 are integers

OUTPUT:

For valid input, a string of the form '[a1,a2,a3,a4,a6]'. For invalid input a ValueError is raised.

EXAMPLES:

A list or tuple of 5 integers:

```
sage: from sage.interfaces.mwrank import validate_mwrank_input
sage: validate_mwrank_input([1,2,3,4,5])
'[1, 2, 3, 4, 5]'
sage: validate_mwrank_input((-1,2,-3,4,-55))
'[-1, 2, -3, 4, -55]'
sage: validate_mwrank_input([1,2,3,4])
Traceback (most recent call last):
...
ValueError: [1, 2, 3, 4] is not valid input to mwrank (should have 5 entries)
sage: validate_mwrank_input([1,2,3,4,i])
```

```
Traceback (most recent call last):
...
ValueError: [1, 2, 3, 4, I] is not valid input to mwrank (entries should be integers)
```

A string of the form '[a1,a2,a3,a4,a6]' with any whitespace and integers ai:

```
sage: validate_mwrank_input('0 -1 1 -7 6')
'[0,-1,1,-7,6]'
sage: validate_mwrank_input("[0,-1,1,0,0]\n")
'[0,-1,1,0,0]'
sage: validate_mwrank_input('0\t -1\t 1\t 0\t 0\n')
'[0,-1,1,0,0]'
sage: validate_mwrank_input('0 -1 1 -7 ')
Traceback (most recent call last):
...
ValueError: 0 -1 1 -7 is not valid input to mwrank
```

INTERFACE TO GNU OCTAVE

GNU Octave is a free software (GPL) MATLAB-like program with numerical routines for integrating, solving systems of equations, special functions, and solving (numerically) differential equations. Please see http://octave.org/ for more details.

The commands in this section only work if you have the optional "octave" interpreter installed and available in your PATH. It's not necessary to install any special Sage packages.

EXAMPLES:

```
sage: octave.eval('2+2')  # optional - octave
'ans = 4'

sage: a = octave(10)  # optional - octave
sage: a**10  # optional - octave
1e+10
```

LOG: - creation (William Stein) - ? (David Joyner, 2005-12-18) - Examples (David Joyner, 2005-01-03)

30.1 Computation of Special Functions

Octave implements computation of the following special functions (see the maxima and gp interfaces for even more special functions):

```
airv
   Airy functions of the first and second kind, and their derivatives.
   airy(0,x) = Ai(x), airy(1,x) = Ai'(x), airy(2,x) = Bi(x), airy(3,x) = Bi'(x)
   Bessel functions of the first kind.
bessely
   Bessel functions of the second kind.
besseli
   Modified Bessel functions of the first kind.
besselk
   Modified Bessel functions of the second kind.
besselh
   Compute Hankel functions of the first (k = 1) or second (k = 2) kind.
bet.a
   The Beta function,
         beta (a, b) = gamma (a) * gamma (b) / gamma (a + b).
   The incomplete Beta function,
erf
   The error function,
```

```
erfinv
The inverse of the error function.
gamma
The Gamma function,
gammainc
The incomplete gamma function,
```

For example,

```
sage: octave("airy(3,2)")
                                 # optional - octave
4.10068
sage: octave("beta(2,2)") # optional - octave
0.166667
sage: octave("betainc(0.2,2,2)") # optional - octave
0.104
sage: octave("besselh(0,2)")
                                 # optional - octave
(0.223891, 0.510376)
sage: octave("besselh(0,1)")
                                # optional - octave
(0.765198, 0.088257)
sage: octave("besseli(1,2)")
                                # optional - octave
sage: octave("besselj(1,2)")
                                 # optional - octave
0.576725
sage: octave("besselk(1,2)")
                                 # optional - octave
0.139866
sage: octave("erf(0)")
                                 # optional - octave
sage: octave("erf(1)")
                                 # optional - octave
0.842701
sage: octave("erfinv(0.842)")
                                 # optional - octave
0.998315
sage: octave("gamma(1.5)")
                                # optional - octave
0.886227
sage: octave("gammainc(1.5,1)")
                                 # optional - octave
0.77687
```

The Octave interface reads in even very long input (using files) in a robust manner:

```
sage: t = '"%s"'%10^10000 # ten thousand character string.
sage: a = octave.eval(t + ';') # optional - octave, < 1/100th of a second
sage: a = octave(t) # optional - octave</pre>
```

Note that actually reading a back out takes forever. This *must* be fixed as soon as possible, see trac ticket #940.

30.2 Tutorial

```
sage: octave('4+10')  # optional - octave
14
sage: octave('date')  # optional - octave; random output
18-Oct-2007
sage: octave('5*10 + 6')  # optional - octave
56
sage: octave('(6+6)/3')  # optional - octave
4
```

```
sage: octave('9')^2  # optional - octave
81
sage: a = octave(10); b = octave(20); c = octave(30)  # optional - octave
sage: avg = (a+b+c)/3  # optional - octave
sage: avg  # optional - octave
20
sage: parent(avg)  # optional - octave
Octave
```

```
sage: my_scalar = octave('3.1415')
                                         # optional - octave
sage: my_scalar
                                         # optional - octave
3.1415
sage: my_vector1 = octave('[1,5,7]')
                                         # optional - octave
sage: my_vector1
                                         # optional - octave
     5
                                         # optional - octave
sage: my_vector2 = octave('[1;5;7]')
sage: my_vector2
                                         # optional - octave
1
5
sage: my_vector1 * my_vector2
                                         # optional - octave
```

Bases: sage.interfaces.expect.Expect

Interface to the Octave interpreter.

EXAMPLES:

clear (var)

Clear the variable named var.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.clear('x') # optional - octave
sage: octave.get('x') # optional - octave
"error: 'x' undefined near line ... column 1"
```

console ()

Spawn a new Octave command-line session.

This requires that the optional octave program be installed and in your PATH, but no optional Sage packages need be installed.

30.2. Tutorial 221

EXAMPLES:

```
sage: octave_console()  # not tested
GNU Octave, version 2.1.73 (i386-apple-darwin8.5.3).
Copyright (C) 2006 John W. Eaton.
...
octave:1> 2+3
ans = 5
octave:2> [ctl-d]
```

Pressing ctrl-d exits the octave console and returns you to Sage. octave, like Sage, remembers its history from one session to another.

de_system_plot (f, ics, trange)

Plots (using octave's interface to gnuplot) the solution to a 2×2 system of differential equations.

INPUT:

- •f a pair of strings representing the differential equations; The independent variable must be called x and the dependent variable must be called y.
- •ics a pair [x0,y0] such that x(t0) = x0, y(t0) = y0
- •trange a pair [t0,t1]

OUTPUT: a gnuplot window appears

EXAMPLES:

```
sage: octave.de_system_plot(['x+y','x-y'], [1,-1], [0,2]) # not tested --_
\rightarrow does this actually work (on OS X it fails for me -- William Stein, 2007-10)
```

This should yield the two plots (t, x(t)), (t, y(t)) on the same graph (the t-axis is the horizontal axis) of the system of ODEs

$$x' = x + y, x(0) = 1;$$
 $y' = x - y, y(0) = -1,$ for $0 < t < 2.$

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.get('x') # optional - octave
' 2'
```

quit (verbose=False)

EXAMPLES:

```
sage: o = Octave()
sage: o._start()  # optional - octave
sage: o.quit(True)  # optional - octave
Exiting spawned Octave process.
```

sage2octave_matrix_string (A)

Return an octave matrix from a Sage matrix.

INPUT: A Sage matrix with entries in the rationals or reals.

OUTPUT: A string that evaluates to an Octave matrix.

```
sage: M33 = MatrixSpace(QQ,3,3)
sage: A = M33([1,2,3,4,5,6,7,8,0])
sage: octave.sage2octave_matrix_string(A) # optional - octave
'[1, 2, 3; 4, 5, 6; 7, 8, 0]'
```

AUTHORS:

•David Joyner and William Stein

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: octave.set('x', '2') # optional - octave
sage: octave.get('x') # optional - octave
' 2'
```

set seed (seed=None)

Sets the seed for the random number generator for this octave interpreter.

EXAMPLES:

```
sage: o = Octave() # optional - octave
sage: o.set_seed(1) # optional - octave
1
sage: [o.rand() for i in range(5)] # optional - octave
[ 0.134364,  0.847434,  0.763775,  0.255069,  0.495435]
```

$solve_linear_system(A, b)$

Use octave to compute a solution x to A*x = b, as a list.

INPUT:

- $\bullet A$ mxn matrix A with entries in \mathbf{Q} or \mathbf{R}
- •b m-vector b entries in \mathbf{Q} or \mathbf{R} (resp)

OUTPUT: A list x (if it exists) which solves M*x = b

EXAMPLES:

AUTHORS:

•David Joyner and William Stein

version ()

Return the version of Octave.

OUTPUT: string

EXAMPLES:

30.2. Tutorial 223

```
sage: v = octave.version() # optional - octave
sage: v # optional - octave; random
'2.13.7'

sage: import re
sage: assert re.match("\d+\.\d+\.\d+\", v) is not None # optional - octave
```

class sage.interfaces.octave. OctaveElement (parent, value, is_name=False, name=None)

Bases: sage.interfaces.expect.ExpectElement

```
sage.interfaces.octave.octave_console ()
```

Spawn a new Octave command-line session.

This requires that the optional octave program be installed and in your PATH, but no optional Sage packages need be installed.

EXAMPLES:

```
sage: octave_console() # not tested
GNU Octave, version 2.1.73 (i386-apple-darwin8.5.3).
Copyright (C) 2006 John W. Eaton.
...
octave:1> 2+3
ans = 5
octave:2> [ctl-d]
```

Pressing ctrl-d exits the octave console and returns you to Sage. octave, like Sage, remembers its history from one session to another.

```
sage.interfaces.octave.octave_version()
```

DEPRECATED: Return the version of Octave installed.

EXAMPLES:

```
sage: octave_version() # optional - octave
doctest:...: DeprecationWarning: This has been deprecated. Use
octave.version() instead
See http://trac.sagemath.org/21135 for details.
'...'
```

sage.interfaces.octave. reduce_load_Octave ()

EXAMPLES:

```
sage: from sage.interfaces.octave import reduce_load_Octave
sage: reduce_load_Octave()
Octave
```

sage.interfaces.octave. to_complex (octave_string, R)

Helper function to convert octave complex number

TESTS:

```
sage: from sage.interfaces.octave import to_complex
sage: to_complex('(0,1)', CDF)
1.0*I
sage: to_complex('(1.3231,-0.2)', CDF)
1.3231 - 0.2*I
```

CHAPTER

THIRTYONE

INTERFACE TO PHC.

PHC computes numerical information about systems of polynomials over the complex numbers.

PHC implements polynomial homotopy methods to exploit structure in order to better approximate all isolated solutions. The package also includes extra tools to handle positive dimensional solution components.

AUTHORS:

- PHC was written by J. Verschelde, R. Cools, and many others (?)
- William Stein and Kelly ?? first version of interface to PHC
- Marshall Hampton second version of interface to PHC
- Marshall Hampton and Alex Jokela third version, path tracking

class sage.interfaces.phc. PHC

A class to interface with PHCpack, for computing numerical homotopies and root counts.

EXAMPLES:

```
sage: from sage.interfaces.phc import phc
sage: R. \langle x, y \rangle = PolynomialRing(CDF, 2)
sage: testsys = [x^2 + 1, x*y - 1]
sage: phc.mixed_volume(testsys)
                                       # optional -- phc
sage: v = phc.blackbox(testsys, R)
                                       # optional -- phc
sage: sols = v.solutions()
                                       # optional -- phc
sage: sols.sort()
                                       # optional -- phc
sage: sols
                                       # optional -- phc
[[-1.0000000000000*I, 1.0000000000000*I], [1.0000000000000*I, -1.
→00000000000000*I]]
sage: sol_dict = v.solution_dicts()
                                       # optional -- phc
sage: x_sols_from_dict = [d[x] for d in sol_dict] # optional -- phc
                                                    # optional -- phc
sage: x_sols_from_dict.sort(); x_sols_from_dict
[-1.000000000000000*I, 1.000000000000*I]
sage: residuals = [[test_equation.change_ring(CDF).subs(sol) for test_equation in_
→testsys] for sol in v.solution_dicts()] # optional -- phc
                                       # optional -- phc
sage: residuals
[[0, 0], [0, 0]]
```

blackbox (polys, input_ring, verbose=False)

Returns as a string the result of running PHC with the given polynomials under blackbox mode (the '-b' option).

INPUT:

•polys – a list of multivariate polynomials (elements of a multivariate polynomial ring).

- •input_ring for coercion of the variables into the desired ring.
- •verbose print lots of verbose information about what this function does.

OUTPUT:

•a PHC_Object object containing the phcpack output string.

EXAMPLES:

```
sage: from sage.interfaces.phc import *
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: start_sys = [x^6-y^2,y^5-1]
sage: sol = phc.blackbox(start_sys, R2)  # optional -- phc
sage: len(sol.solutions())  # optional -- phc
30
```

mixed_volume (polys, verbose=False)

Computes the mixed volume of the polynomial system given by the input polys.

INPUT:

- •polys a list of multivariate polynomials (elements of a multivariate polynomial ring).
- •verbose print lots of verbose information about what this function does.

OUTPUT:

•The mixed volume.

EXAMPLES:

```
sage: from sage.interfaces.phc import *
sage: R2.<x,y,z> = PolynomialRing(QQ,3)
sage: test_sys = [(x+y+z)^2-1,x^2-x,y^2-1]
sage: phc.mixed_volume(test_sys) # optional -- phc
4
```

path_track (start_sys, end_sys, input_ring, c_skew=0.001, saved_start=None)

This function computes homotopy paths between the solutions of start_sys and end_sys.

INPUT:

- •start_sys a square polynomial system, given as a list of polynomials
- •end_sys same type as start_sys
- •input_ring for coercion of the variables into the desired ring.
- •c_skew optional. the imaginary part of homotopy multiplier; nonzero values are often necessary to avoid intermediate path collisions
- •saved_start optional. A phc output file. If not given, start system solutions are computed via the phc.blackbox function.

OUTPUT:

•a list of paths as dictionaries, with the keys variables and t-values on the path.

```
sage: from sage.interfaces.phc import *
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: start_sys = [x^6-y^2,y^5-1]
sage: sol = phc.blackbox(start_sys, R2) # optional -- phc
```

This returns a graphics object of solution paths in the complex plane.

INPUT:

- •start_sys a square polynomial system, given as a list of polynomials
- •end_sys same type as start_sys
- •input_ring for coercion of the variables into the desired ring.
- •c_skew optional. the imaginary part of homotopy multiplier; nonzero values are often necessary to avoid intermediate path collisions
- •endpoints optional. Whether to draw in the ends of paths as points.
- •saved_start optional. A phc output file. If not given, start system solutions are computed via the phc.blackbox function.

OUTPUT:

•lines and points of solution paths

EXAMPLES:

start_from (*start_filename_or_string*, *polys*, *input_ring*, *path_track_file=None*, *verbose=False*) This computes solutions starting from a phcpack solution file.

INPUT:

- •start_filename_or_string the filename for a phcpack start system, or the contents of such a file as a string. Variable names must match the inputring variables. The value of the homotopy variable t should be 1, not 0.
- •polys a list of multivariate polynomials (elements of a multivariate polynomial ring).
- •input_ring: for coercion of the variables into the desired ring.
- •path_track_file: whether to save path-tracking information
- •verbose print lots of verbose information about what this function does.

OUTPUT:

•A solution in the form of a PHCObject.

EXAMPLES:

```
sage: from sage.interfaces.phc import *
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: start_sys = [x^6-y^2,y^5-1]
sage: sol = phc.blackbox(start_sys, R2)  # optional -- phc
sage: start_save = sol.save_as_start()  # optional -- phc
sage: end_sys = [x^7-2,y^5-x^2]  # optional -- phc
sage: sol = phc.start_from(start_save, end_sys, R2)  # optional -- phc
sage: len(sol.solutions())  # optional -- phc
30
```

class sage.interfaces.phc. PHC_Object (output_file_contents, input_ring)

A container for data from the PHCpack program - lists of float solutions, etc. Currently the file contents are kept as a string; for really large outputs this would be bad.

INPUT:

output_file_contents: the string output of PHCpack

•input_ring: for coercion of the variables into the desired ring.

EXAMPLES:

```
sage: from sage.interfaces.phc import phc
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: start_sys = [(x-1)^2+(y-1)-1, x^2+y^2-1]
sage: sol = phc.blackbox(start_sys, R2) # optional -- phc
sage: str(sum([x[0] for x in sol.solutions()]).real())[0:3] # optional -- phc
'2.0'
```

classified_solution_dicts()

Returns a dictionary of lists of dictionaries of solutions. Its not as crazy as it sounds; the keys are the types of solutions as classified by phcpack: regular vs. singular, complex vs. real

INPUT:

•None

OUTPUT:

•A dictionary of lists of dictionaries of solutions

EXAMPLES:

```
sage: from sage.interfaces.phc import phc
sage: R.<x,y> = PolynomialRing(CC,2)
sage: p_sys = [x^10-y,y^2-1]
sage: sol = phc.blackbox(p_sys,R)  # optional -- phc
sage: classifieds = sol.classified_solution_dicts()  # optional -- phc
sage: str(sum([q[y] for q in classifieds['real']]))[0:3]  # optional -- phc
'2.0'
```

```
save_as_start ( start_filename=None, sol_filter='')
```

Saves a solution as a phcpack start file. The usual output is just as a string, but it can be saved to a file as well. Even if saved to a file, it still returns the output string.

```
sage: from sage.interfaces.phc import phc
sage: R2.<x,y> = PolynomialRing(QQ,2)
sage: start_sys = [x^3-y^2,y^5-1]
sage: sol = phc.blackbox(start_sys, R2) # optional -- phc
sage: start_save = sol.save_as_start() # optional -- phc
sage: end_sys = [x^7-2,y^5-x^2] # optional -- phc
sage: sol = phc.start_from(start_save, end_sys, R2) # optional -- phc
sage: len(sol.solutions()) # optional -- phc
15
```

solution_dicts (get_failures=False)

Returns a list of solutions in dictionary form: variable:value.

INPUT:

- •self for access to self_out_file_contents, the string of raw PHCpack output.
- •get_failures (optional) a boolean. The default (False) is to not process failed homotopies. These either lie on positive-dimensional components or at infinity.

OUTPUT:

•solution_dicts: a list of dictionaries. Each dictionary element is of the form variable:value, where the variable is an element of the input_ring, and the value is in ComplexField.

EXAMPLES:

```
sage: from sage.interfaces.phc import *
sage: R.<x,y,z> = PolynomialRing(QQ,3)
sage: fs = [x^2-1,y^2-x,z^2-y]
sage: sol = phc.blackbox(fs,R)  # optional -- phc
sage: s_list = sol.solution_dicts()  # optional -- phc
sage: s_list.sort()  # optional -- phc
sage: s_list[0]  # optional -- phc
{y: 1.000000000000000, z: -1.000000000000000, x: 1.00000000000000}
```

solutions (get_failures=False)

Returns a list of solutions in the ComplexField.

Use the variable_list function to get the order of variables used by PHCpack, which is usually different than the term order of the input_ring.

INPUT:

- •self for access to self_out_file_contents, the string of raw PHCpack output.
- •get_failures (optional) a boolean. The default (False) is to not process failed homotopies. These either lie on positive-dimensional components or at infinity.

OUTPUT:

•solutions: a list of lists of ComplexField-valued solutions.

```
sage: from sage.interfaces.phc import *
sage: R2.<x1,x2> = PolynomialRing(QQ,2)
sage: test_sys = [x1^5-x1*x2^2-1, x2^5-x1*x2-1]
sage: sol = phc.blackbox(test_sys, R2)  # optional -- phc
sage: len(sol.solutions())  # optional -- phc
25
```

```
variable list()
```

Returns the variables, as strings, in the order in which PHCpack has processed them.

EXAMPLES:

```
sage: from sage.interfaces.phc import *
sage: R2.<x1,x2> = PolynomialRing(QQ,2)
sage: test_sys = [x1^5-x1*x2^2-1, x2^5-x1*x2-1]
sage: sol = phc.blackbox(test_sys, R2)  # optional -- phc
sage: sol.variable_list()  # optional -- phc
['x1', 'x2']
```

Returns a dictionary of lists of dictionaries of variable:value (key:value) pairs. Only used internally; see the classified_solution_dict function in the PHC_Object class definition for details.

INPUT:

- •output_file_contents phc solution output as a string
- •input_ring a PolynomialRing that variable names can be coerced into

OUTPUT:

•a dictionary of lists if dictionaries of solutions, classifies by type

EXAMPLES:

Returns a list of dictionaries of variable:value (key:value) pairs. Only used internally; see the solution_dict function in the PHC_Object class definition for details.

INPUT:

- output_file_contents phc solution output as a string
- •input_ring a PolynomialRing that variable names can be coerced into

OUTPUT:

a list of dictionaries of solutions

```
sage: from sage.interfaces.phc import *
sage: R2.<x1,x2> = PolynomialRing(QQ,2)
sage: test_sys = [(x1-1)^5-x2, (x2-1)^5-1]
sage: sol = phc.blackbox(test_sys, R2)  # optional -- phc
sage: test = get_solution_dicts(sol.output_file_contents,R2)  # optional -- phc
sage: str(sum([q[x1].real() for q in test]))[0:4]  # optional -- phc
'25.0'
```

```
sage.interfaces.phc. get_variable_list ( output_file_contents)
```

Returns the variables, as strings, in the order in which PHCpack has processed them.

```
sage: from sage.interfaces.phc import *
sage: R2.<x1,x2> = PolynomialRing(QQ,2)
sage: test_sys = [(x1-2)^5-x2, (x2-1)^5-1]
sage: sol = phc.blackbox(test_sys, R2)  # optional -- phc
sage: get_variable_list(sol.output_file_contents) # optional -- phc
['x1', 'x2']
```

POV-RAY, THE PERSISTENCE OF VISION RAY TRACER



CHAPTER

THIRTYTHREE

PARALLEL INTERFACE TO THE SAGE INTERPRETER

This is an expect interface to emph{multiple} copy of the sage interpreter, which can all run simultaneous calculations. A PSage object does not work as well as the usual Sage object, but does have the great property that when you construct an object in a PSage you get back a prompt immediately. All objects constructed for that PSage print <<currently executing code>> until code execution completes, when they print as normal.

note {BUG – currently non-idle PSage subprocesses do not stop when sage exits. I would very much like to fix this but don't know how.}

EXAMPLES:

We illustrate how to factor 3 integers in parallel. First start up 3 parallel Sage interfaces:

```
sage: v = [PSage() for _ in range(3)]
```

Next, request factorization of one random integer in each copy.

```
sage: w = [x('factor(2^*s-1)' * randint(250,310)) for x in v] # long time (5s on sage. <math>\rightarrow math, 2011)
```

Print the status:

```
sage: w  # long time, random output (depends on timing)
[3 * 11 * 31^2 * 311 * 11161 * 11471 * 73471 * 715827883 * 2147483647 * 4649919401 *

→18158209813151 * 5947603221397891 * 29126056043168521,

<<currently executing code>>,
9623 * 68492481833 *

→23579543011798993222850893929565870383844167873851502677311057483194673]
```

Note that at the point when we printed two of the factorizations had finished but a third one hadn't. A few seconds later all three have finished:

```
class sage.interfaces.psage. PSage ( **kwds)
    Bases: sage.interfaces.sage0.Sage
    eval ( x, strip=True, **kwds)
        x - code strip -ignored
```

CHAPTER

THIRTYFOUR

INTERFACE TO QEPCAD

The basic function of QEPCAD is to construct cylindrical algebraic decompositions (CADs) of \mathbb{R}^k , given a list of polynomials. Using this CAD, it is possible to perform quantifier elimination and formula simplification.

A CAD for a set A of k-variate polynomials decomposes \mathbf{R}^j into disjoint cells, for each j in $0 \le j \le k$. The sign of each polynomial in A is constant in each cell of \mathbf{R}^k , and for each cell in \mathbf{R}^j (j > 1), the projection of that cell into \mathbf{R}^{j-1} is a cell of \mathbf{R}^{j-1} . (This property makes the decomposition 'cylindrical'.)

Given a formula $\exists x. P(a,b,x) = 0$ (for a polynomial P), and a cylindrical algebraic decomposition for P, we can eliminate the quantifier (find an equivalent formula in the two variables a, b without the quantifier \exists) as follows. For each cell C in \mathbf{R}^2 , find the cells of \mathbf{R}^3 which project to C. (This collection is called the stack over C.) Mark C as true if some member of the stack has sign = 0; otherwise, mark C as false. Then, construct a polynomial formula in a, b which specifies exactly the true cells (this is always possible). The same technique works if the body of the quantifier is any boolean combination of polynomial equalities and inequalities.

Formula simplification is a similar technique. Given a formula which describes a simple set of \mathbb{R}^k in a complicated way as a boolean combination of polynomial equalities and inequalities, QEPCAD can construct a CAD for the polynomials and recover a simple equivalent formula.

Note that while the following documentation is tutorial in nature, and is written for people who may not be familiar with QEPCAD, it is documentation for the sage interface rather than for QEPCAD. As such, it does not cover several issues that are very important to use QEPCAD efficiently, such as variable ordering, the efficient use of the alternate quantifiers and <code>_root_</code> expressions, the <code>measure-zero-error</code> command, etc. For more information on QEPCAD, see the online documentation at <code>url{http://www.cs.usna.edu/~qepcad/B/QEPCAD.html}</code> and Chris Brown's tutorial handout and slides from <code>url{http://www.cs.usna.edu/~wcbrown/research/ISSAC04/Tutorial.html}</code>. (Several of the examples in this documentation came from these sources.)

The examples below require that the optional general package is installed.

QEPCAD can be run in a fully automatic fashion, or interactively. We first demonstrate the automatic use of QEPCAD.

Since sage has no built-in support for quantifiers, this interface provides <code>qepcad_formula</code> which helps construct quantified formulas in the syntax QEPCAD requires.

```
sage: var('a,b,c,d,x,y,z')
(a, b, c, d, x, y, z)
sage: qf = qepcad_formula
```

We start with a simple example. Consider an arbitrarily-selected ellipse:

```
sage: ellipse = 3*x^2 + 2*x*y + y^2 - x + y - 7
```

What is the projection onto the x axis of this ellipse? First we construct a formula asking this question.

```
sage: F = qf.exists(y, ellipse == 0); F
(E y)[3 x^2 + 2 x y + y^2 - x + y - 7 = 0]
```

Then we run qepcad to get the answer:

How about the projection onto the y axis?

```
sage: qepcad(qf.exists(x, ellipse == 0)) # optional - qepcad
8 y^2 + 16 y - 85 <= 0</pre>
```

QEPCAD deals with more quantifiers than just 'exists', of course. Besides the standard 'forall', there are also 'for infinitely many', 'for all but finitely many', 'for a connected subset', and 'for exactly k'. The <code>qepcad()</code> documentation has examples of all of these; here we will just give one example.

First we construct a circle:

```
sage: circle = x^2 + y^2 - 3
```

For what values k does a vertical line x = k intersect the combined figure of the circle and ellipse exactly three times?

Here we see that the solutions are among the eight (4+2+2) roots of the three polynomials inside the brackets, but not all of these roots are solutions; the polynomial inequalities outside the brackets are needed to select those roots that are solutions.

QEPCAD also supports an extended formula language, where $_{\tt root_k}\ P(\bar x,y)$ refers to a particular zero of $P(\bar x,y)$ (viewed as a polynomial in y). If there are n roots, then $_{\tt root_1}$ refers to the least root and $_{\tt root_n}$ refers to the greatest. Also, $_{\tt root_n}$ refers to the least root and $_{\tt root_1}$ refers to the greatest.

This extended language is available both on input and output; see the QEPCAD documentation for more information on how to use this syntax on input. We can request output that is intended to be easy to interpret geometrically; then QEPCAD will use the extended language to produce a solution formula without the selection polynomials.

```
sage: qepcad(F, solution='geometric') # not tested (random order)
x = _root_1 8 x^2 - 8 x - 29
\/
8 x^4 - 26 x^2 - 4 x + 13 = 0
\/
x = _root_-1 x^2 - 3
```

We then see that the 6 solutions correspond to the vertical tangent on the left side of the ellipse, the four intersections between the ellipse and the circle, and the vertical tangent on the right side of the circle.

Let us do some basic formula simplification and visualization. We will look at the region which is inside both the ellipse and the circle:

We get back the same formula we put in. This is not surprising (we started with a pretty simple formula, after all), but it is not very enlightening either. Again, if we ask for a 'geometric' output, then we see an output that lets us understand something about the shape of the solution set.

There is another reason to prefer output using $_root_$ expressions; not only does it sometimes give added insight into the geometric structure, it also can be more efficient to construct. Consider this formula for the projection of a particular semicircle onto the x axis:

Here, the formula x > 0 had to be introduced in order to get a solution formula; the original CAD of F did not include the polynomial x. To avoid having QEPCAD do the extra work to come up with a solution formula, we can tell it to use the extended language; it is always possible to construct a solution formula in the extended language without introducing new polynomials.

```
sage: qepcad(F, solution='extended') # not tested (random_
\rightarrow order)
x^2 - 3 <= 0 /\ x > _root_1 2 x^2 - 3
```

Up to this point, all the output we have seen has basically been in the form of strings; there is no support (yet) for parsing these outputs back into sage polynomials (partly because sage does not yet have support for symbolic conjunctions and disjunctions). The function qepcad() supports three more output types that give numbers which can be manipulated in sage: any-point, all-points, and cell-points.

These output types give dictionaries mapping variable names to values. With any-point, qepcad() either produces a single dictionary specifying a point where the formula is true, or raises an exception if the formula is false everywhere. With all-points, qepcad() either produces a list of dictionaries for all points where the formula is true, or raises an exception if the formula is true on infinitely many points. With cell-points, qepcad() produces a list of dictionaries with one point for each cell where the formula is true. (This means you will have at least one point in each connected component of the solution, although you will often have many more points than that.)

Let us revisit some of the above examples and get some points to play with. We will start by finding a point on our ellipse.

```
sage: p = qepcad(ellipse == 0, solution='any-point'); p # optional - qepcad
{'x': -1.468501968502953?, 'y': 0.9685019685029527?}
```

(Note that despite the decimal printing and the question marks, these are really exact numbers.)

We can verify that this point is a solution. To do so, we create a copy of ellipse as a polynomial over \mathbf{Q} (instead of a symbolic expression).

```
sage: pellipse = QQ['x,y'](ellipse)
sage: pellipse(**p) == 0  # optional - qepcad
True
```

For cell-points, let us look at points not on the ellipse.

```
sage: pts = qepcad(ellipse != 0, solution='cell-points'); pts # optional - qepcad
[{'x': 4, 'y': 0},
    {'x': 2.468501968502953?, 'y': 1},
    {'x': 2.468501968502953?, 'y': -9},
    {'x': 1/2, 'y': 9},
    {'x': 1/2, 'y': -1},
    {'x': 1/2, 'y': -5},
    {'x': -1.468501968502953?, 'y': 3},
    {'x': -1.468501968502953?, 'y': -1},
    {'x': -3, 'y': 0}]
```

For the points here which are in full-dimensional cells, QEPCAD has the freedom to choose rational sample points, and it does so.

And, of course, all these points really are not on the ellipse.

```
sage: [pellipse(**p) != 0 for p in pts] # optional - qepcad
[True, True, True, True, True, True, True, True]
```

Finally, for all-points, let us look again at finding vertical lines that intersect the union of the circle and the ellipse exactly three times.

Since y is bound by the quantifier, the solutions only refer to x.

We can substitute one of these solutions into the original equation:

and verify that it does have three roots:

Let us check all six solutions.

We said earlier that we can run QEPCAD either automatically or interactively. Now that we have discussed the automatic modes, let us turn to interactive uses.

If the <code>qepcad()</code> function is passed <code>interact=True</code>, then instead of returning a result, it returns an object of class <code>Qepcad</code> representing a running instance of QEPCAD that you can interact with. For example:

This object is a fairly thin wrapper over QEPCAD; most QEPCAD commands are available as methods on the <code>Qepcad</code> object. Given a <code>Qepcad</code> object <code>qe</code>, you can type <code>qe.[tab]</code> to see the available QEPCAD commands; to see the documentation for an individual QEPCAD command, for example <code>d_setting</code>, you can type <code>qe.d_setting</code>? . (In QEPCAD, this command is called <code>d-setting</code>. We systematically replace hyphens with underscores for this interface.)

The execution of QEPCAD is divided into four phases. Most commands are not available during all phases. We saw above that QEPCAD starts out in phase 'Before Normalization'. We see that the d_cell command is not available in this phase:

```
sage: qe.d_cell() # optional - qepcad
Error GETCID: This command is not active here.
```

We will focus here on the fourth (and last) phase, 'Before Solution', because this interface has special support for some operations in this phase. Consult the QEPCAD documentation for information on the other phases.

We can tell QEPCAD to finish off the current phase and move to the next with its go command. (There is also the step command, which partially completes a phase for phases that have multiple steps, and the finish command, which runs QEPCAD to completion.)

Note that the <code>Qepcad</code> object returns the new phase whenever the phase changes, as a convenience for interactive use; except that when the new phase is <code>'EXITED'</code>, the solution formula printed by QEPCAD is returned instead.

```
sage: qe.go() # optional - qepcad
4 c - b^2 > 0
sage: qe # optional - qepcad
QEPCAD object in phase 'EXITED'
```

Let us pick a nice, simple example, return to phase 4, and explore the resulting qe object.

```
sage: qe = qepcad(circle == 0, interact=True); qe  # optional - qepcad
QEPCAD object in phase 'Before Normalization'
sage: qe.go(); qe.go()  # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
```

We said before that QEPCAD creates 'cylindrical algebraic decompositions'; since we have a bivariate polynomial, we get decompositions of \mathbf{R}^0 , \mathbf{R}^1 , and \mathbf{R}^2 . In this case, where our example is a circle of radius $\sqrt{3}$ centered on the origin, these decompositions are as follows:

The decomposition of ${\bf R}^0$ is trivial (of course). The decomposition of ${\bf R}^1$ has five cells: $x<-\sqrt{3},\,x=-\sqrt{3},\,-\sqrt{3}< x<\sqrt{3},\,x=\sqrt{3},\,$ and $x>\sqrt{3}.$ These five cells comprise the stack over the single cell in the trivial decomposition of ${\bf R}^0$.

These five cells give rise to five stacks in \mathbb{R}^2 . The first and fifth stack have just one cell apiece. The second and fourth stacks have three cells: y < 0, y = 0, and y > 0. The third stack has five cells: below the circle, the lower semicircle, the interior of the circle, the upper semicircle, and above the circle.

QEPCAD (and this QEPCAD interface) number the cells in a stack starting with 1. Each cell has an index, which is a tuple of integers describing the path to the cell in the tree of all cells. For example, the cell 'below the circle' has index (3,1) (the first cell in the stack over the third cell of \mathbf{R}^1) and the interior of the circle has index (3,3).

We can view these cells with the QEPCAD command d_cell . For instance, let us look at the cell for the upper semicircle:

```
sage: qe.d_cell(3, 4)
                                                # optional - qepcad
----- Information about the cell (3,4) -----
Level
                          : 2
Dimension
Number of children : 0
Truth value : T
                         : T
                                 by trial evaluation.
Degrees after substitution : Not known yet or No polynomial.
Multiplicities : ((1,1))
Signs of Projection Factors
Level 1 : (-)
Level 2 : (0)
----- Sample point -----
The sample point is in a PRIMITIVE representation.
alpha = the unique root of x^2 - 3 between 0 and 4
     = 1.7320508076-
Coordinate 1 = 0
           = 0.000000000
Coordinate 2 = alpha
            = 1.7320508076 -
```

We see that, the level of this cell is 2, meaning that it is part of the decomposition of \mathbb{R}^2 . The dimension is 1, meaning that the cell is homeomorphic to a line (rather than a plane or a point). The sample point gives the coordinates of one point in the cell, both symbolically and numerically.

For programmatic access to cells, we have defined a sage wrapper class QepcadCell. These cells can be created with the cell() method; for example:

```
sage: c = qe.cell(3, 4); c # optional - qepcad
QEPCAD cell (3, 4)
```

A QepcadCell has accessor methods for the important state held within a cell. For instance:

```
sage: c.level()  # optional - qepcad
2
sage: c.index()  # optional - qepcad
(3, 4)
sage: qe.cell(3).number_of_children()  # optional - qepcad
5
sage: len(qe.cell(3))  # optional - qepcad
5
```

One particularly useful thing we can get from a cell is its sample point, as sage algebraic real numbers.

We have seen that we can get cells using the cell() method. There are several QEPCAD commands that print lists of cells; we can also get cells using the make_cells() method, passing it the output of one of these commands.

```
sage: qe.make_cells(qe.d_true_cells()) # optional - qepcad
[QEPCAD cell (4, 2), QEPCAD cell (3, 4), QEPCAD cell (3, 2),
QEPCAD cell (2, 2)]
```

Also, the cells in the stack over a given cell can be accessed using array subscripting or iteration. (Remember that cells in a stack are numbered starting with one; we preserve this convention in the array-subscripting syntax.)

```
sage: c = qe.cell(3)  # optional - qepcad
sage: c[1]  # optional - qepcad
QEPCAD cell (3, 1)
sage: [c2 for c2 in c]  # optional - qepcad
[QEPCAD cell (3, 1), QEPCAD cell (3, 2), QEPCAD cell (3, 3),
QEPCAD cell (3, 4), QEPCAD cell (3, 5)]
```

We can do one more thing with a cell: we can set its truth value. Once the truth values of the cells have been set, we can get QEPCAD to produce a formula which is true in exactly the cells we have selected. This is useful if QEPCAD's quantifier language is insufficient to express your problem.

For example, consider again our combined figure of the circle and the ellipse. Suppose you want to find all vertical lines that intersect the circle twice, and also intersect the ellipse twice. The vertical lines that intersect the circle twice can be found by simplifying:

```
sage: F = qf.exactly_k(2, y, circle == 0); F
(X2 y) [x^2 + y^2 - 3 = 0]
```

and the vertical lines that intersect the ellipse twice are expressed by:

```
sage: G = qf.exactly_k(2, y, ellipse == 0); G
(X2 y)[3 x^2 + 2 x y + y^2 - x + y - 7 = 0]
```

and the lines that intersect both figures would be:

```
sage: qf.and_(F, G)
Traceback (most recent call last):
...
ValueError: QEPCAD formulas must be in prenex (quantifiers outermost) form
```

...except that QEPCAD does not support formulas like this; in QEPCAD input, all logical connectives must be inside all quantifiers.

Instead, we can get QEPCAD to construct a CAD for our combined figure and set the truth values ourselves. (The exact formula we use doesn't matter, since we're going to replace the truth values in the cells; we just need to use a formula that uses both polynomials.)

Now we want to find all cells c in the decomposition of \mathbf{R}^1 such that the stack over c contains exactly two cells on the ellipse, and also contains exactly two cells on the circle.

Our input polynomials are 'level-2 projection factors', we see:

so we can test whether a cell is on the ellipse by checking that the sign of the corresponding projection factor is 0 in our cell. For instance, the cell (12,2) is on the ellipse:

```
sage: qe.cell(12,2).signs()[1][0] # optional - qepcad
0
```

So we can update the truth values as desired like this:

and then we can get our desired solution formula. (The 'G' stands for 'geometric', and gives solutions using the same rules as solution='geometric' described above.)

```
sage: qe.solution_extension('G')  # not tested (random order)
8 x^2 - 8 x - 29 < 0
/\
x^2 - 3 < 0</pre>
```

TESTS:

Check the qepcad configuration file:

```
sage: open('%s/default.qepcadrc'%SAGE_LOCAL).readlines()[-1]
'SINGULAR .../bin\n'
```

Tests related to the not tested examples (nondeterministic order of atoms):

```
sage: from sage.interfaces.gepcad import _gepcad_atoms
sage: var('a,b,c,d,x,y,z')
(a, b, c, d, x, y, z)
sage: qf = qepcad_formula
sage: ellipse = 3*x^2 + 2*x*y + y^2 - x + y - 7
sage: circle = x^2 + y^2 - 3
sage: F = qf.exactly_k(3, y, circle * ellipse == 0)
sage: _qepcad_atoms(qepcad(F))
                                                                 # optional - gepcad
\{ '8 x^2 - 8 x - 29 \le 0',
'8 x^2 - 8 x - 29 = 0',
'8 x^4 - 26 x^2 - 4 x + 13 = 0'
'8 x^4 - 26 x^2 - 4 x + 13 >= 0'
'x^2 - 3 <= 0'
'x^2 - 3 = 0'
sage: _qepcad_atoms(gepcad(F, solution='geometric'))
                                                                # optional - gepcad
\{ 18 \times 4 - 26 \times 2 - 4 \times + 13 = 0 \},
 'x = \_root\_-1 x^2 - 3'
'x = \_root_1 \ 8 \ x^2 - 8 \ x - 29'
sage: F = qf.and_(ellipse < 0, circle < 0)</pre>
sage: _qepcad_atoms(qepcad(F))
                                                                 # optional - gepcad
\{'y^2 + 2 \times y + y + 3 \times^2 - x - 7 < 0', 'y^2 + x^2 - 3 < 0'\}
sage: _qepcad_atoms(qepcad(F, solution='geometric'))
                                                                 # optional - gepcad
\{ 18 \times 4 - 26 \times 2 - 4 \times + 13 < 0 \},
'x < \_root\_-2 \ 8 \ x^4 - 26 \ x^2 - 4 \ x + 13',
'x = \_root\_-2 \ 8 \ x^4 - 26 \ x^2 - 4 \ x + 13',
'x = \_root_2 \ 8 \ x^4 - 26 \ x^2 - 4 \ x + 13',
'x > \_root_2 8 x^4 - 26 x^2 - 4 x + 13',
'y^2 + 2 x y + y + 3 x^2 - x - 7 < 0'
y^2 + x^2 - 3 < 0
sage: F = qf.exists(y, qf.and_(circle == 0, x + y > 0))
sage: _qepcad_atoms(qepcad(F))
                                                                 # optional - gepcad
\{'2 x^2 - 3 < 0', 'x > 0', 'x^2 - 3 \le 0'\}
sage: _qepcad_atoms(qepcad(F, solution='extended'))
                                                                # optional - gepcad
\{'x > \_root_1 \ 2 \ x^2 - 3', \ 'x^2 - 3 \le 0'\}
sage: qe = qepcad(qf.and_(ellipse == 0, circle == 0), interact=True) # optional_
→- qepcad
sage: qe.go(); qe.go(); qe.go()
                                                          # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
                                                          # optional - qepcad
sage: for c in qe.cell():
....: count_ellipse = 0
        count_circle = 0
. . . . :
        for c2 in c:
. . . . :
           count_ellipse += (c2.signs()[1][0] == 0)
. . . . :
            count_circle += (c2.signs()[1][1] == 0)
. . . . :
....: c.set_truth(count_ellipse == 2 and count_circle == 2)
sage: _qepcad_atoms(qe.solution_extension('G'))
                                                                 # optional - gepcad
\{'8 x^2 - 8 x - 29 < 0', 'x^2 - 3 < 0'\}
```

AUTHORS:

- Carl Witty (2008-03): initial version
- Thierry Monteil (2015-07) repackaging + noncommutative doctests.

The wrapper for QEPCAD.

answer ()

For a QEPCAD instance which is finished, return the simplified quantifier-free formula that it printed just before exiting.

EXAMPLES:

```
sage: qe = qepcad(x^3 - x == 0, interact=True)  # optional - qepcad
sage: qe.finish()  # not tested (random order)
x - 1 <= 0 /\ x + 1 >= 0 /\ [ x = 0 \/ x - 1 = 0 \/ x + 1 = 0 ]
sage: qe.answer()  # not tested (random order)
x - 1 <= 0 /\ x + 1 >= 0 /\ [ x = 0 \/ x - 1 = 0 \/ x + 1 = 0 ]
```

TESTS:

Tests related to the not tested examples (nondeterministic order of atoms):

assume (assume)

The following documentation is from qepcad.help.

Add an assumption to the problem. These will not be included in the solution formula.

For example, with input (E x)[a $x^2 + b x + c = 0$], if we issue the command

```
assume [a \neq 0]
```

we will get the solution formula $b^2 - 4ac >= 0$. Without the assumption we'd get something like [a = 0/b/e] / [a/e] / [a/e

EXAMPLES:

cell (*index)

Given a cell index, returns a <code>QepcadCell</code> wrapper for that cell. Uses a cache for efficiency.

```
sage: qe = qepcad(x + 3 == 42, interact=True) # optional - qepcad
sage: qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'At the end of projection phase'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.cell(2) # optional - qepcad
QEPCAD cell (2)
sage: qe.cell(2) is qe.cell(2) # optional - qepcad
True
```

final_stats()

For a QEPCAD instance which is finished, return the statistics that it printed just before exiting.

EXAMPLES:

make_cells (text)

Given the result of some QEPCAD command that returns cells (such as d_{cell}), $d_{witness_list}$), etc.), return a list of cell objects.

EXAMPLES:

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.make_cells(qe.d_false_cells()) # optional - qepcad
[QEPCAD cell (5, 1), QEPCAD cell (4, 3), QEPCAD cell (4, 1), QEPCAD cell (3, ...
$\to 5)$, QEPCAD cell (3, 3), QEPCAD cell (3, 1), QEPCAD cell (2, 3), QEPCAD cell $\to (2, 1)$, QEPCAD cell (1, 1)]
```

phase ()

Return the current phase of the QEPCAD program.

```
sage: qe = qepcad(x > 2/3, interact=True) # optional - qepcad
sage: qe.phase() # optional - qepcad
'Before Normalization'
sage: qe.go() # optional - qepcad
QEPCAD object has moved to phase 'At the end of projection phase'
sage: qe.phase() # optional - qepcad
'At the end of projection phase'
sage: qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Choice'
sage: qe.phase() # optional - qepcad
```

```
'Before Choice'
sage: qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Solution'
sage: qe.phase() # optional - qepcad
'Before Solution'
sage: qe.go() # optional - qepcad
3 x - 2 > 0
sage: qe.phase() # optional - qepcad
'EXITED'
```

set_truth_value (index, nv)

Given a cell index (or a cell) and an integer, set the truth value of the cell to that integer.

Valid integers are 0 (false), 1 (true), and 2 (undetermined).

EXAMPLES:

```
sage: qe = qepcad(x == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'At the end of projection phase'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.set_truth_value(1, 1) # optional - qepcad
```

solution extension (kind)

The following documentation is modified from <code>qepcad.help</code>:

solution-extension x

Use an alternative solution formula construction method. The parameter x is allowed to be T,E, or G. If x is T, then a formula in the usual language of Tarski formulas is produced. If x is E, a formula in the language of Extended Tarski formulas is produced. If x is G, then a geometry-based formula is produced.

```
sage: var('x,y')
(x, y)
sage: qf = qepcad_formula
sage: qe = qepcad(qf.and_(x^2 + y^2 - 3 == 0, x + y > 0), interact=True) #,
→optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: ge.solution_extension('E')
                                                   # not tested (random...
⇔order)
x > \_root_1 2 x^2 - 3 / y^2 + x^2 - 3 = 0 / [2 x^2 - 3 > 0 / y = \_root_1]
-y^2 + x^2 - 3
sage: qe.solution_extension('G')
                                                   # not tested (random
⇔order)
Γ
 2 x^2 - 3 < 0
   \/
   x = \_root\_-1 \ 2 \ x^2 - 3
 /\
 y = \_root_{-1} y^2 + x^2 - 3
```

TESTS:

Tests related to the not tested examples (nondeterministic order of atoms):

```
sage: from sage.interfaces.gepcad import _gepcad_atoms
sage: var('x,y')
(x, y)
sage: qf = qepcad_formula
sage: qe = qepcad(qf.and_(x^2 + y^2 - 3 == 0, x + y > 0), interact=True) #...
→optional - gepcad
sage: qe.go(); qe.go(); qe.go() # optional - gepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: _qepcad_atoms(qe.solution_extension('E'))
                                                 # optional - gepcad
\{'2 x^2 - 3 > 0',
'x > \_root_1 2 x^2 - 3',
'y = root_{-1} y^2 + x^2 - 3'
'v^2 + x^2 - 3 = 0'
sage: _qepcad_atoms(qe.solution_extension('G'))
                                                      # optional - gepcad
\{'2 x^2 - 3 < 0',
'x = \_root\_-1 \ 2 \ x^2 - 3',
'x > \_root_{-1} 2 x^2 - 3',
'x^2 - 3 <= 0'
'y = \_root\_-1 \ y^2 + x^2 - 3'
y^2 + x^2 - 3 = 0
sage: _qepcad_atoms(qe.solution_extension('T')) # optional - qepcad
\{ y + x > 0', y^2 + x^2 - 3 = 0' \}
```

class sage.interfaces.qepcad. QepcadCell (parent, lines)

A wrapper for a QEPCAD cell.

index ()

Give the index of a QEPCAD cell.

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.cell().index() # optional - qepcad
()
sage: qe.cell(1).index() # optional - qepcad
```

```
(1,)
sage: qe.cell(2, 2).index() # optional - qepcad
(2, 2)
```

level ()

Return the level of a QEPCAD cell.

EXAMPLES:

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.cell().level() # optional - qepcad
0
sage: qe.cell(1).level() # optional - qepcad
1
sage: qe.cell(2, 2).level() # optional - qepcad
2
```

number_of_children ()

Return the number of elements in the stack over a QEPCAD cell. (This is always an odd number, if the stack has been constructed.)

EXAMPLES:

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.cell().number_of_children() # optional - qepcad
5
sage: [c.number_of_children() for c in qe.cell()] # optional - qepcad
[1, 3, 5, 3, 1]
```

sample_point ()

Return the coordinates of a point in the cell, as a tuple of sage algebraic reals.

```
sage: qe = qepcad(x^2 - x - 1 == 0, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'At the end of projection phase'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: v1 = qe.cell(2).sample_point()[0]; v1 # optional - qepcad
-0.618033988749895?
sage: v2 = qe.cell(4).sample_point()[0]; v2 # optional - qepcad
1.618033988749895?
sage: v1 + v2 == 1 # optional - qepcad
True
```

sample point dict()

Return the coordinates of a point in the cell, as a dictionary mapping variable names (as strings) to sage algebraic reals.

EXAMPLES:

```
sage: qe = qepcad(x^2 - x - 1 == 0, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'At the end of projection phase'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.cell(4).sample_point_dict() # optional - qepcad
{'x': 1.618033988749895?}
```

set_truth (v)

Set the truth value of this cell, as used by QEPCAD for solution formula construction.

The argument v should be either a boolean or None (which will set the truth value to 'undetermined').

EXAMPLES:

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - qepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: qe.solution_extension('T') # optional - qepcad
y^2 + x^2 - 1 = 0
sage: qe.cell(3, 3).set_truth(True) # optional - qepcad
sage: qe.solution_extension('T') # optional - qepcad
y^2 + x^2 - 1 <= 0</pre>
```

signs ()

Return the sign vector of a QEPCAD cell.

This is a list of lists. The outer list contains one element for each level of the cell; the inner list contains one element for each projection factor at that level. These elements are either -1, 0, or 1.

```
sage: var('x,y')
(x, y)
sage: qe = qepcad(x^2 + y^2 == 1, interact=True) # optional - qepcad
sage: qe.go(); qe.go(); qe.go() # optional - gepcad
QEPCAD object has moved to phase 'Before Projection (y)'
QEPCAD object has moved to phase 'Before Choice'
QEPCAD object has moved to phase 'Before Solution'
sage: from sage.interfaces.gepcad import QepcadCell
sage: all_cells = flatten(qe.cell(), ltypes=QepcadCell, max_level=1) #_
→optional - qepcad
sage: [(c, c.signs()[1][0]) for c in all_cells] # optional - gepcad
[(QEPCAD cell (1, 1), 1), (QEPCAD cell (2, 1), 1), (QEPCAD cell (2, 2), 0),...
\hookrightarrow (QEPCAD cell (2, 3), 1), (QEPCAD cell (3, 1), 1), (QEPCAD cell (3, 2), 0),
\hookrightarrow (QEPCAD cell (3, 3), -1), (QEPCAD cell (3, 4), 0), (QEPCAD cell (3, 5), 1),
\hookrightarrow (QEPCAD cell (4, 1), 1), (QEPCAD cell (4, 2), 0), (QEPCAD cell (4, 3), 1),...
\hookrightarrow (QEPCAD cell (5, 1), 1)
```

```
class sage.interfaces.qepcad. QepcadFunction ( parent, name)
    Bases: sage.interfaces.expect.ExpectFunction
```

A wrapper for a QEPCAD command.

The low-level wrapper for OEPCAD.

Quantifier elimination and formula simplification using QEPCAD B.

If assume is specified, then the given formula is 'assumed', which is taken into account during final solution formula construction.

If interact=True is given, then a <code>Qepcad</code> object is returned which can be interacted with either at the command line or programmatically.

The type of solution returned can be adjusted with solution. The options are 'geometric', which tries to construct a solution formula with geometric meaning; 'extended', which gives a solution formula in an extended language that may be more efficient to construct; 'any-point', which returns any point where the formula is true; 'all-points', which returns a list of all points where the formula is true (or raises an exception if there are infinitely many); and 'cell-points', which returns one point in each cell where the formula is true.

All other keyword arguments are passed through to the <code>Qepcad</code> constructor.

For much more documentation and many more examples, see the module docstring for this module (type sage.interfaces.gepcad? to read this docstring from the sage command line).

The examples below require that the optional qepcad package is installed.

EXAMPLES:

```
sage: qf = qepcad_formula
sage: var('a,b,c,d,x,y,z,long_with_underscore_314159')
(a, b, c, d, x, y, z, long_with_underscore_314159)
sage: K. < q, r > = QQ[]
sage: qepcad('(E x)[a x + b > 0]', vars='(a,b,x)') # not tested (random,
ن→order)
a /= 0 // b > 0
sage: gepcad(a > b)
                                                # optional - gepcad
b - a < 0
sage: qepcad(qf.exists(x, a*x^2 + b*x + c == 0))
                                                         # not tested (random
→order)
4 \ a \ c - b^2 \le 0 / [c = 0 / a / = 0 / 4 \ a \ c - b^2 < 0]
sage: qepcad(qf.exists(x, a*x^2 + b*x + c == 0), assume=(a != 0))
                                                                    # optional -_
→ qepcad
4 \ a \ c - b^2 <= 0
```

For which values of a, b, c does $ax^2 + bx + c$ have 2 real zeroes?

one real zero?

No real zeroes?

```
sage: exact0 = qepcad(qf.forall(x, a*x^2 + b*x + c != 0)); exact0 # not_

\rightarrow tested (random order)
4 a c - b^2 >= 0 /\ c /= 0 /\ [ b = 0 \/ 4 a c - b^2 > 0 ]
```

3⁷⁵ real zeroes?

```
sage: qepcad(qf.exactly_k(3^75, x, a*x^2 + b*x + c == 0))
# optional - qepcad
FALSE
```

We can check that the results don't overlap:

and that the union of the results is as expected:

```
sage: qepcad(r'[[%s] \/ [%s] \/ [%s]]' % (exact0, exact1, exact2), vars=(a,b,c))

→# not tested (random order)
b /= 0 \/ a /= 0 \/ c /= 0
```

So we have finitely many zeroes if a, b, or c is nonzero; which means we should have infinitely many zeroes if they are all zero.

The polynomial is nonzero almost everywhere iff it is not identically zero.

The non-zeroes are continuous iff there are no zeroes or if the polynomial is zero.

The zeroes are continuous iff there are no or one zeroes, or if the polynomial is zero:

Since polynomials are continuous and y > 0 is an open set, they are positive infinitely often iff they are positive at least once.

However, since y >= 0 is not open, the equivalence does not hold if you replace 'positive' with 'nonnegative'. (We assume $a \neq 0$ to get simpler formulas.)

TESTS:

We verify that long variable names work. (Note that QEPCAD does not support underscores, so they are stripped from the formula.)

Tests related to the not tested examples (nondeterministic order of atoms):

```
\{'4 \ a \ c - b^2 < 0', '4 \ a \ c - b^2 = 0', 'a < 0', 'a = 0', 'a > 0'\}
sage: exact0 = qepcad(qf.forall(x, a*x^2 + b*x + c != 0)); _qepcad_atoms(exact0) ...

    # optional - gepcad
\{'4 \ a \ c - b^2 > 0', '4 \ a \ c - b^2 >= 0', 'b = 0', 'c /= 0'\}
sage: qepcad(r'[[%s] / [%s]]' % (exact0, exact1), vars='a,b,c')
        # optional - gepcad
sage: qepcad(r'[[%s]] / [%s]]' % (exact0, exact2), vars='a,b,c')
        # optional - qepcad
FALSE
sage: qepcad(r'[[%s] / [%s]]' % (exact1, exact2), vars='a,b,c')
        # optional - qepcad
FALSE
sage: _qepcad_atoms(qepcad(r'[[\$s] \/ [\$s]]' \$ (exact0, exact1, exact2),_
→vars=(a,b,c))) # optional - qepcad
{'a /= 0', 'b /= 0', 'c /= 0'}
sage: _qepcad_atoms(qepcad(qf.infinitely_many(x, a*x^2 + b*x + c == 0)))
     # optional - gepcad
\{ 'a = 0', 'b = 0', 'c = 0' \}
sage: _qepcad_atoms(qepcad(qf.all_but_finitely_many(x, a*x^2 + b*x + c != 0)))
       # optional - gepcad
{'a /= 0', 'b /= 0', 'c /= 0'}
sage: _{qepcad_atoms(qepcad(qf.connected_subset(x, a*x^2 + b*x + c != 0)))}
     # optional - qepcad
\{'4 \ a \ c - b^2 > 0', '4 \ a \ c - b^2 >= 0', 'a = 0'\}
sage: _qepcad_atoms(qepcad(qf.connected_subset(x, a*x^2 + b*x + c == 0)))
     # optional - qepcad
\hookrightarrow
\{ '4 \ a \ c - b^2 >= 0', 'a = 0' \}
sage: _qepcad_atoms(qepcad(r'[[\$s] \/ [\$s] \/ [a = 0 /\ b = 0 /\ c = 0]]' \$_
\{'4 \ a \ c - b^2 >= 0', 'a = 0'\}
sage: _qepcad_atoms(qepcad(qf.infinitely_many(x, a*x^2 + b*x + c > 0)))

    # optional - qepcad

\{'4 \text{ a c} - b^2 < 0', 'a > 0', 'c > 0'\}
sage: _qepcad_atoms(qepcad(qf.exists(x, a*x^2 + b*x + c > 0)))
     # optional - qepcad
\{'4 \ a \ c - b^2 < 0', 'a > 0', 'c > 0'\}
sage: _qepcad_atoms(qepcad(qf.infinitely_many(x, a*x^2 + b*x + c \ge 0), assume=(a_
\leftrightarrow!= 0))) # optional - qepcad
\{'4 \ a \ c - b^2 < 0', 'a > 0'\}
sage: _{qepcad}_atoms(_{qepcad}(_{qf.exists}(_{x}, _{a*x^2} + b*x + c >= 0), _{assume}=(a != 0)))
     # optional - qepcad
\{'4 \ a \ c - b^2 \le 0', 'a > 0'\}
```

sage.interfaces.qepcad.qepcad_banner ()
Return the QEPCAD startup banner.

EXAMPLES:

```
sage: from sage.interfaces.qepcad import qepcad_banner
sage: qepcad_banner() # optional - qepcad
_____
             Quantifier Elimination
                     in
          Elementary Algebra and Geometry
                     by
    Partial Cylindrical Algebraic Decomposition
                     by
                   Hoon Hong
              (hhong@math.ncsu.edu)
With contributions by: Christopher W. Brown, George E.
Collins, Mark J. Encarnacion, Jeremy R. Johnson
Werner Krandick, Richard Liska, Scott McCallum,
Nicolas Robidoux, and Stanly Steinberg
______
```

sage.interfaces.gepcad.gepcad_console (memcells=None)

Run QEPCAD directly. To exit early, press Control-C.

EXAMPLES:

```
sage: qepcad_console() # not tested
...
Enter an informal description between '[' and ']':
```

class sage.interfaces.qepcad. qepcad_formula_factory

Contains routines to help construct formulas in QEPCAD syntax.

A (v, formula)

Given a variable (or list of variables) and a formula, returns the universal quantification of the formula over the variables.

This method is available both as forall() and A() (the QEPCAD name for this quantifier).

The input formula may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.forall(a, a^2 + b > b^2 + a)
(A a) [a^2 + b > b^2 + a]
sage: qf.forall((a, b), a^2 + b^2 > 0)
(A a) (A b) [a^2 + b^2 > 0]
sage: qf.A(b, b^2 != a)
(A b) [b^2 /= a]
```

C (v, formula, allow multi=False)

Given a variable and a formula, returns a new formula which is true iff the set of values for the variable at which the original formula was true is connected (including cases where this set is empty or is a single point).

This method is available both as $connected_subset()$ and C() (the QEPCAD name for this quantifier).

The input formula may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.connected_subset(a, a^2 + b > b^2 + a)
(C a) [a^2 + b > b^2 + a]
sage: qf.C(b, b^2 != a)
(C b) [b^2 /= a]
```

\mathbf{E} (v, formula)

Given a variable (or list of variables) and a formula, returns the existential quantification of the formula over the variables.

This method is available both as exists() and E() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.exists(a, a^2 + b > b^2 + a)
(E a)[a^2 + b > b^2 + a]
sage: qf.exists((a, b), a^2 + b^2 < 0)
(E a) (E b)[a^2 + b^2 < 0]
sage: qf.E(b, b^2 == a)
(E b)[b^2 = a]</pre>
```

F (v, formula)

Given a variable and a formula, returns a new formula which is true iff the original formula was true for infinitely many values of the variable.

This method is available both as $infinitely_many()$ and F() (the QEPCAD name for this quantifier).

The input formula may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.infinitely_many(a, a^2 + b > b^2 + a)
(F a)[a^2 + b > b^2 + a]
sage: qf.F(b, b^2 != a)
(F b)[b^2 /= a]
```

G(v, formula)

Given a variable and a formula, returns a new formula which is true iff the original formula was true for all but finitely many values of the variable.

This method is available both as $all_but_finitely_many()$ and G() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.all_but_finitely_many(a, a^2 + b > b^2 + a)
(G a)[a^2 + b > b^2 + a]
sage: qf.G(b, b^2 != a)
(G b)[b^2 /= a]
```

X (k, v, formula, allow_multi=False)

Given a nonnegative integer k, a variable, and a formula, returns a new formula which is true iff the original formula is true for exactly k values of the variable.

This method is available both as exactly k() and X() (the QEPCAD name for this quantifier).

(Note that QEPCAD does not support k = 0 with this syntax, so if k = 0 is requested we implement it with forall() and not_().)

The input formula may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.exactly_k(1, x, x^2 + a*x + b == 0)
(X1 x)[a x + x^2 + b = 0]
sage: qf.exactly_k(0, b, a*b == 1)
(A b)[~a b = 1]
```

all_but_finitely_many (v, formula)

Given a variable and a formula, returns a new formula which is true iff the original formula was true for all but finitely many values of the variable.

This method is available both as $all_but_finitely_many()$ and G() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.all_but_finitely_many(a, a^2 + b > b^2 + a)
(G a) [a^2 + b > b^2 + a]
sage: qf.G(b, b^2 != a)
(G b) [b^2 /= a]
```

and_ (*formulas)

Return the conjunction of its input formulas.

(This method would be named 'and' if that were not a Python keyword.)

Each input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLES:

```
sage: var('a,b,c,x')
(a, b, c, x)
sage: qf = qepcad_formula
sage: qf.and_(a*b, a*c, b*c != 0)
[a b = 0 /\ a c = 0 /\ b c /= 0]
sage: qf.and_(a*x^2 == 3, qf.or_(a > b, b > c))
[a x^2 = 3 /\ [a > b \/ b > c]]
```

atomic (*lhs*, *op='='*, *rhs=0*)

Construct a QEPCAD formula from the given inputs.

INPUT:

- •lhs a polynomial, or a symbolic equality or inequality
- •op a relational operator, default '='
- •rhs a polynomial, default 0

If lhs is a symbolic equality or inequality, then op and rhs are ignored.

This method works by printing the given polynomials, so we do not care what ring they are in (as long as they print with integral or rational coefficients).

EXAMPLES:

```
sage: qf = qepcad_formula
sage: var('a,b,c')
(a, b, c)
sage: K.<x,y> = QQ[]
sage: def test_qf(qf):
...:     return qf, qf.vars
sage: test_qf(qf.atomic(a^2 + 17))
(a^2 + 17 = 0, frozenset({'a'}))
sage: test_qf(qf.atomic(a*b*c <= c^3))
(a b c <= c^3, frozenset({'a', 'b', 'c'}))
sage: test_qf(qf.atomic(x+y^2, '!=', a+b))
(y^2 + x /= a + b, frozenset({'a', 'b', 'x', 'y'}))
sage: test_qf(qf.atomic(x, operator.lt))
(x < 0, frozenset({'x'}))</pre>
```

connected_subset (v, formula, allow_multi=False)

Given a variable and a formula, returns a new formula which is true iff the set of values for the variable at which the original formula was true is connected (including cases where this set is empty or is a single point).

This method is available both as $connected_subset()$ and C() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
```

```
sage: qf.connected_subset(a, a^2 + b > b^2 + a)
(C a) [a^2 + b > b^2 + a]
sage: qf.C(b, b^2 != a)
(C b) [b^2 /= a]
```

exactly_k (k, v, formula, allow_multi=False)

Given a nonnegative integer k, a variable, and a formula, returns a new formula which is true iff the original formula is true for exactly k values of the variable.

This method is available both as $exactly_k()$ and X() (the QEPCAD name for this quantifier).

(Note that QEPCAD does not support k=0 with this syntax, so if k=0 is requested we implement it with forall() and not_().)

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.exactly_k(1, x, x^2 + a*x + b == 0)
(X1 x)[a x + x^2 + b = 0]
sage: qf.exactly_k(0, b, a*b == 1)
(A b)[~a b = 1]
```

exists (v, formula)

Given a variable (or list of variables) and a formula, returns the existential quantification of the formula over the variables.

This method is available both as exists() and E() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.exists(a, a^2 + b > b^2 + a)
(E a) [a^2 + b > b^2 + a]
sage: qf.exists((a, b), a^2 + b^2 < 0)
(E a) (E b) [a^2 + b^2 < 0]
sage: qf.E(b, b^2 == a)
(E b) [b^2 = a]</pre>
```

forall (v, formula)

Given a variable (or list of variables) and a formula, returns the universal quantification of the formula over the variables.

This method is available both as forall () and A () (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.forall(a, a^2 + b > b^2 + a)
(A a) [a^2 + b > b^2 + a]
sage: qf.forall((a, b), a^2 + b^2 > 0)
(A a) (A b) [a^2 + b^2 > 0]
sage: qf.A(b, b^2 != a)
(A b) [b^2 /= a]
```

formula (formula)

Constructs a QEPCAD formula from the given input.

INPUT:

•formula - a polynomial, a symbolic equality or inequality, or a list of polynomials, equalities, or inequalities

A polynomial p is interpreted as the equation p = 0. A list is interpreted as the conjunction ('and') of the elements.

EXAMPLES:

```
sage: var('a,b,c,x')
(a, b, c, x)
sage: qf = qepcad_formula
sage: qf.formula(a*x + b)
a x + b = 0
sage: qf.formula((a*x^2 + b*x + c, a != 0))
[a x^2 + b x + c = 0 /\ a /= 0]
```

iff (f1, f2)

Return the equivalence of its input formulas (that is, given formulas P and Q, returns 'P iff Q').

The input formulas may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

EXAMPLES:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.iff(a, b)
[a = 0 <==> b = 0]
sage: qf.iff(a^2 < b, b^2 < a)
[a^2 < b <==> b^2 < a]</pre>
```

implies (f1, f2)

Return the implication of its input formulas (that is, given formulas P and Q, returns 'P implies Q').

The input formulas may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.implies(a, b)
[a = 0 ==> b = 0]
```

```
sage: qf.implies(a^2 < b, b^2 < a)
[a^2 < b ==> b^2 < a]</pre>
```

infinitely_many (v, formula)

Given a variable and a formula, returns a new formula which is true iff the original formula was true for infinitely many values of the variable.

This method is available both as $infinitely_many()$ and F() (the QEPCAD name for this quantifier).

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLE:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.infinitely_many(a, a^2 + b > b^2 + a)
(F a) [a^2 + b > b^2 + a]
sage: qf.F(b, b^2 != a)
(F b) [b^2 /= a]
```

not (formula)

Return the negation of its input formula.

(This method would be named 'not' if that were not a Python keyword.)

The input formula may be a *qformula* as returned by the methods of qepcad_formula, a symbolic equality or inequality, or a polynomial p (meaning p = 0).

EXAMPLES:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.not_(a > b)
[~a > b]
sage: qf.not_(a^2 + b^2)
[~a^2 + b^2 = 0]
sage: qf.not_(qf.and_(a > 0, b < 0))
[~[a > 0 /\ b < 0]]</pre>
```

or_ (*formulas)

Return the disjunction of its input formulas.

(This method would be named 'or' if that were not a Python keyword.)

Each input formula may be a qformula as returned by the methods of $qepcad_formula$, a symbolic equality or inequality, or a polynomial p (meaning p=0).

```
sage: var('a,b,c,x')
(a, b, c, x)
sage: qf = qepcad_formula
sage: qf.or_(a*b, a*c, b*c != 0)
[a b = 0 \/ a c = 0 \/ b c /= 0]
sage: qf.or_(a*x^2 == 3, qf.and_(a > b, b > c))
[a x^2 = 3 \/ [a > b /\ b > c]]
```

quantifier (kind, v, formula, allow_multi=True)

A helper method for building quantified QEPCAD formulas; not expected to be called directly.

Takes the quantifier kind (the string label of this quantifier), a variable or list of variables, and a formula, and returns the quantified formula.

EXAMPLES:

```
sage: var('a,b')
(a, b)
sage: qf = qepcad_formula
sage: qf.quantifier('NOT_A_REAL_QEPCAD_QUANTIFIER', a, a*b==0)
(NOT_A_REAL_QEPCAD_QUANTIFIER a) [a b = 0]
sage: qf.quantifier('FOO', (a, b), a*b)
(FOO a) (FOO b) [a b = 0]
```

sage.interfaces.qepcad. qepcad_version ()

Return a string containing the current QEPCAD version number.

EXAMPLES:

```
sage: qepcad_version() # random, optional - qepcad
'Version B 1.69, 16 Mar 2012'
```

TESTS:

```
sage: qepcad_version() # optional - qepcad
'Version B ..., ...'
```

class sage.interfaces.qepcad. qformula (formula, vars, qvars=[])

A qformula holds a string describing a formula in QEPCAD's syntax, and a set of variables used.

THIRTYFIVE

INTERFACE TO BILL HART'S QUADRATIC SIEVE

sage.interfaces.qsieve. data_to_list (out, n, time)

Convert output of Hart's sieve and n to a list and time.

INPUT:

- •out snapshot of text output of Hart's QuadraticSieve program
- •n the integer being factored

OUTPUT:

- •list proper factors found so far
- •str time information

sage.interfaces.qsieve. qsieve (n, block=True, time=False, verbose=False)

Run Hart's quadratic sieve and return the distinct proper factors of the integer n that it finds.

CONDITIONS:

The conditions for the quadratic sieve to work are as follows:

- •No small factors
- •Not a perfect power
- •Not prime

If any of these fails, the sieve will also.

INPUT:

- •n an integer with at least 40 digits
- •block (default: True) if True, you must wait until the sieve computation is complete before using Sage further. If False, Sage will run while the sieve computation runs in parallel. If q is the returned object, use q.quit() to terminate a running factorization.
- •time (default: False) if True, time the command using the UNIX "time" command (which you might have to install).
- •verbose (default: False) if True, print out verbose logging information about what happened during the Sieve run (for non-blocking Sieve, verbose information is always available via the log() method.)

OUTPUT:

- •list a list of the distinct proper factors of n found
- •str the time in cpu seconds that the computation took, as given by the command line time command. (If time is False, this is always an empty string.)

sage.interfaces.qsieve. qsieve_block (n, time, verbose=False)

Compute the factorization of n using Hart's quadratic Sieve blocking until complete.

```
class sage.interfaces.qsieve. qsieve_nonblock ( n, time)
```

A non-blocking version of Hart's quadratic sieve.

The sieve starts running when you create the object, but you can still use Sage in parallel.

EXAMPLES:

```
sage: k = 19; n = next_prime(10^k) *next_prime(10^(k+1))
sage: q = qsieve(n, block=False, time=True) # optional - time
                # random output; optional - time
Proper factors so far: []
               # random output; optional - time
sage: q
([1000000000000000051, 1000000000000000039], '0.21')
sage: q.list() # random output; optional - time
[1000000000000000051, 1000000000000000039]
sage: q.time() # random output; optional - time
'0.21'
sage: q = qsieve(next_prime(10^20)*next_prime(10^21), block=False)
sage: q # random output
Proper factors so far: [10000000000000000039, 10000000000000000117]
                 # random output
[100000000000000000039, 10000000000000000117]
```

cputime ()

Return the time in seconds (as a string) that it took to factor n, or return '?' if the factorization has not completed or the time is unknown.

done ()

Return True if the sieve process has completed.

list (

Return a list of the factors found so far, as Sage integers.

log()

Return all output of running the sieve so far.

n ()

Return the integer that is being factored.

pid()

Return the PIN id of the QuadraticSieve process (actually of the time process that spawns the sieve process).

quit ()

Terminate the QuadraticSieve process, in case you want to give up on computing this factorization.

```
sage: n = next_prime(2^310)*next_prime(2^300)
sage: qs = qsieve(n, block=False)
sage: qs
Proper factors so far: []
sage: qs.quit()
sage: qs
Factorization was terminated early.
```

time ()

Return the time in seconds (as a string) that it took to factor n, or return '?' if the factorization has not completed or the time is unknown.

```
sage.interfaces.qsieve.tmpdir()
```



CHAPTER

THIRTYSIX

INTERFACES TO R

This is the reference to the Sagemath R interface, usable from any Sage program.

The %r interface creating an R cell in the sage notebook is decribed in the Notebook manual.

The %R and %%R interface creating an R line or an R cell in the Jupyter notebook are briefly decribed at the end of this page. This documentation will be expanded and placed in the Jupyter notebook manual when this manual exists.

The following examples try to follow "An Introduction to R" which can be found at http://cran.r-project.org/doc/manuals/R-intro.html .

EXAMPLES:

Simple manipulations; numbers and vectors

The simplest data structure in R is the numeric vector which consists of an ordered collection of numbers. To create a vector named x using the R interface in Sage, you pass the R interpreter object a list or tuple of numbers:

```
sage: x = r([10.4,5.6,3.1,6.4,21.7]); x
[1] 10.4 5.6 3.1 6.4 21.7
```

You can invert elements of a vector x in R by using the invert operator or by doing 1/x:

```
sage: ~x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
sage: 1/x
[1] 0.09615385 0.17857143 0.32258065 0.15625000 0.04608295
```

The following assignment creates a vector y with 11 entries which consists of two copies of x with a 0 in between:

```
sage: y = r([x,0,x]); y
[1] 10.4 5.6 3.1 6.4 21.7 0.0 10.4 5.6 3.1 6.4 21.7
```

Vector Arithmetic

The following command generates a new vector v of length 11 constructed by adding together (element by element) 2x repeated 2.2 times, y repeated just once, and 1 repeated 11 times:

```
sage: v = 2*x+y+1; v
[1] 32.2 17.8 10.3 20.2 66.1 21.8 22.6 12.8 16.9 50.8 43.5
```

One can compute the sum of the elements of an R vector in the following two ways:

```
sage: sum(x)
[1] 47.2
sage: x.sum()
[1] 47.2
```

One can calculate the sample variance of a list of numbers:

```
sage: ((x-x.mean())^2/(x.length()-1)).sum()
[1] 53.853
sage: x.var()
[1] 53.853
sage: x.sort()
[1] 3.1 5.6 6.4 10.4 21.7
sage: x.min()
[1] 3.1
sage: x.max()
[1] 21.7
sage: x
[1] 10.4 5.6 3.1 6.4 21.7
sage: r(-17).sqrt()
[1] NaN
sage: r('-17+0i').sqrt()
[1] 0+4.123106i
```

Generating an arithmetic sequence:

```
sage: r('1:10')
[1] 1 2 3 4 5 6 7 8 9 10
```

Because from is a keyword in Python, it can't be used as a keyword argument. Instead, from_ can be passed, and R will recognize it as the correct thing:

```
sage: r.seq(length=10, from_=-1, by=.2)
[1] -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8

sage: x = r([10.4,5.6,3.1,6.4,21.7]);
sage: x.rep(2)
[1] 10.4 5.6 3.1 6.4 21.7 10.4 5.6 3.1 6.4 21.7

sage: x.rep(times=2)
[1] 10.4 5.6 3.1 6.4 21.7 10.4 5.6 3.1 6.4 21.7

sage: x.rep(each=2)
[1] 10.4 10.4 5.6 5.6 3.1 3.1 6.4 6.4 21.7 21.7
```

Missing Values:

```
sage: na = r('NA')
sage: z = r([1,2,3,na])
sage: z
[1] 1 2 3 NA
sage: ind = r.is_na(z)
sage: ind
[1] FALSE FALSE TRUE
sage: zero = r(0)
sage: zero / zero
[1] NaN
sage: inf = r('Inf')
sage: inf-inf
[1] NaN
sage: r.is_na(inf)
[1] FALSE
sage: r.is_na(inf-inf)
[1] TRUE
```

```
sage: r.is_na(zero/zero)
[1] TRUE
sage: r.is_na(na)
[1] TRUE
sage: r.is_nan(inf-inf)
[1] TRUE
sage: r.is_nan(zero/zero)
[1] TRUE
sage: r.is_nan(na)
[1] FALSE
```

Character Vectors:

```
sage: labs = r.paste('c("X","Y")', '1:10', sep='""'); labs
[1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7" "Y8" "X9" "Y10"
```

Index vectors; selecting and modifying subsets of a data set:

```
sage: na = r('NA')
sage: x = r([10.4,5.6,3.1,6.4,21.7,na]); x
[1] 10.4 5.6 3.1 6.4 21.7 NA
sage: x['!is.na(self)']
[1] 10.4 5.6 3.1 6.4 21.7

sage: x = r([10.4,5.6,3.1,6.4,21.7,na]); x
[1] 10.4 5.6 3.1 6.4 21.7 NA
sage: (x+1)['(!is.na(self)) & self>0']
[1] 11.4 6.6 4.1 7.4 22.7
sage: x = r([10.4,-2,3.1,-0.5,21.7,na]); x
[1] 10.4 -2.0 3.1 -0.5 21.7 NA
sage: (x+1)['(!is.na(self)) & self>0']
[1] 11.4 4.1 0.5 22.7
```

Distributions:

```
sage: r.options(width="60");
$widt.h
[1] 100
sage: rr = r.dnorm(r.seq(-3,3,0.1))
sage: rr
[1] 0.004431848 0.005952532 0.007915452 0.010420935
[5] 0.013582969 0.017528300 0.022394530 0.028327038
[9] 0.035474593 0.043983596 0.053990967 0.065615815
[13] 0.078950158 0.094049077 0.110920835 0.129517596
[17] 0.149727466 0.171368592 0.194186055 0.217852177
[21] 0.241970725 0.266085250 0.289691553 0.312253933
[25] 0.333224603 0.352065327 0.368270140 0.381387815
[29] 0.391042694 0.396952547 0.398942280 0.396952547
[33] 0.391042694 0.381387815 0.368270140 0.352065327
[37] 0.333224603 0.312253933 0.289691553 0.266085250
[41] 0.241970725 0.217852177 0.194186055 0.171368592
[45] 0.149727466 0.129517596 0.110920835 0.094049077
[49] 0.078950158 0.065615815 0.053990967 0.043983596
[53] 0.035474593 0.028327038 0.022394530 0.017528300
[57] 0.013582969 0.010420935 0.007915452 0.005952532
[61] 0.004431848
```

Convert R Data Structures to Python/Sage:

```
sage: rr = r.dnorm(r.seq(-3,3,0.1))
sage: sum(rr._sage_())
9.9772125168981...
```

Or you get a dictionary to be able to access all the information:

It is also possible to access the plotting capabilities of R through Sage. For more information see the documentation of r.plot() or r.png().

THE JUPYTER NOTEBOOK INTERFACE (work in progress).

The %r interface described in the Sage notebook manual is not useful in the Jupyter notebook : it creates a inferior R interpreter which cannot be escaped.

The RPy2 library allows the creation of an R cell in the Jupyter notebook analogous to the %r escape in command line or %r cell in a Sage notebook.

The interface is loaded by a cell containing the sole code:

"%load ext rpy2.ipython"

After execution of this code, the %R and %%R magics are available:

- %R allows the execution of a single line of R code. Data exchange is possible via the -i and -o options. Do "%R?" in a standalone cell to get the documentation.
- % R alows the execution in R of the whole text of a cell, with similar options (do "%%R?" in a standalone cell for documentation).

A few important points must be noted:

- The R interpreter launched by this interface IS (currently) DIFFERENT from the R interpreter used br other r... functions.
- Data exchanged via the -i and -o options have a format DIFFERENT from the format used by the r... functions (RPy2 mostly uses arrays, and bugs the user to use the pandas Python package).
- R graphics are (beautifully) displayed in output cells, but are not directly importable. You have to save them as .png, .pdf or .svg files and import them in Sage for further use.

In its current incarnation, this interface is mostly useful to statisticians needing Sage for a few symbolic computations but mostly using R for applied work.

AUTHORS:

- Mike Hansen (2007-11-01)
- William Stein (2008-04-19)
- Harald Schilly (2008-03-20)

- Mike Hansen (2008-04-19)
- Emmanuel Charpentier (2015-12-12, RPy2 interface)

```
{\bf class} \; {\tt sage.interfaces.r.} \; {\tt HelpExpression}
```

Bases: str

Used to improve printing of output of r.help.

class sage.interfaces.r. R (maxread=None, script_subdirectory=None, server_tmpdir=None, logfile=None, server=None, init_list_length=1024, seed=None)

```
Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.Expect
```

An interface to the R interpreter.

R is a comprehensive collection of methods for statistics, modelling, bioinformatics, data analysis and much more. For more details, see http://www.r-project.org/about.html

Resources:

- •http://r-project.org/ provides more information about R.
- •http://rseek.org/ R's own search engine.

EXAMPLES:

TESTS:

```
sage: r == loads(dumps(r))
True
```

available_packages ()

Returns a list of all available R package names.

This list is not necessarily sorted.

OUTPUT: list of strings

Note: This requires an internet connection. The CRAN server is that is checked is defined at the top of sage/interfaces/r.py.

EXAMPLES:

```
sage: ap = r.available_packages() # optional - internet
sage: len(ap) > 20 # optional - internet
True
```

call (function_name, *args, **kwds)

This is an alias for function_call().

```
sage: r.call('length', [1,2,3])
[1] 3
```

chdir (dir)

Changes the working directory to dir

INPUT:

•dir – the directory to change to.

EXAMPLES:

```
sage: import tempfile
sage: tmpdir = tempfile.mkdtemp()
sage: r.chdir(tmpdir)
```

Check that tmpdir and r.getwd() refer to the same directory. We need to use realpath() in case \$TMPDIR (by default /tmp) is a symbolic link (see trac ticket #10264).

```
sage: os.path.realpath(tmpdir) == sageobj(r.getwd()) # known bug (trac #9970)
True
```

completions (s)

Return all commands names that complete the command starting with the string s. This is like typing s[Ctrl-T] in the R interpreter.

INPUT:

•s – string

OUTPUT: list – a list of strings

EXAMPLES:

```
sage: dummy = r._tab_completion(use_disk_cache=False) #clean doctest
sage: r.completions('tes')
['testInheritedMethods', 'testPlatformEquivalence', 'testVirtual']
```

console ()

Runs the R console as a separate new R process.

EXAMPLES:

```
sage: r.console() # not tested

R version 2.6.1 (2007-11-26)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
...
```

convert_r_list (l)

Converts an R list to a Python list.

```
'package:methods',
'Autoloads',
'package:base']
```

eval (code, globals=None, locals=None, synchronize=True, *args, **kwds)

Evaluates a command inside the R interpreter and returns the output as a string.

EXAMPLES:

```
sage: r.eval('1+1')
'[1] 2'
```

function_call (function, args=None, kwds=None)

Return the result of calling an R function, with given args and keyword args.

OUTPUT: RElement – an object in R

EXAMPLES:

```
sage: r.function_call('length', args=[ [1,2,3] ])
[1] 3
```

get (var)

Returns the string representation of the variable var.

INPUT:

•var – a string

OUTPUT: string

EXAMPLES:

```
sage: r.set('a', 2)
sage: r.get('a')
'[1] 2'
```

help (command)

Returns help string for a given command.

INPUT: - command - a string

OUTPUT: HelpExpression – a subclass of string whose __repr__ method is __str__, so it prints nicely

EXAMPLES:

install_packages (package_name)

Install an R package into Sage's R installation.

```
sage: r.install_packages('aaMI') # not tested
...
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
...
Please restart Sage in order to use 'aaMI'.
```

library (library_name)

Load the library library_name into the R interpreter.

This function raises an ImportError if the given library is not known.

INPUT:

•library_name - string

EXAMPLES:

```
sage: r.library('grid')
sage: 'grid' in r.eval('(.packages())')
True
sage: r.library('foobar')
Traceback (most recent call last):
...
ImportError: ...
```

na ()

Returns the NA in R.

OUTPUT: RElement - an element of R

EXAMPLES:

```
sage: r.na()
[1] NA
```

```
plot ( *args, **kwds)
```

The R plot function. Type r.help('plot') for much more extensive documentation about this function. See also below for a brief introduction to more plotting with R.

If one simply wants to view an R graphic, using this function is is sufficient (because it calls dev.off() to turn off the device).

However, if one wants to save the graphic to a specific file, it should be used as in the example below to write the output.

EXAMPLES:

This example saves a plot to the standard R output, usually a filename like Rplot001.png - from the command line, in the current directory, and in the cell directory in the notebook:

To save to a specific file name, one should use png() to set the output device to that file. If this is done in the notebook, it must be done in the same cell as the plot itself:

Please note that for more extensive use of R's plotting capabilities (such as the lattices package), it is advisable to either use an interactive plotting device or to use the notebook. The following examples are not tested, because they differ depending on operating system:

In the notebook, one can use r.png() to open the device, but would need to use the following since R lattice graphics do not automatically print away from the command line:

png (*args, **kwds)

Creates an R PNG device.

This should primarily be used to save an R graphic to a custom file. Note that when using this in the notebook, one must plot in the same cell that one creates the device. See r.plot() documentation for more information about plotting via R in Sage.

These examples won't work on the many platforms where R still gets built without graphics support.

```
1
sage: import os; os.unlink(filename) # We remove the file for doctesting;
optional -- rgraphics
```

We want to make sure that we actually can view R graphics, which happens differently on different platforms:

read (filename)

Read filename into the R interpreter by calling R's source function on a read-only file connection.

EXAMPLES:

```
sage: filename = tmp_filename()
sage: f = open(filename, 'w')
sage: f.write('a <- 2+2\n')
sage: f.close()
sage: r.read(filename)
sage: r.get('a')
'[1] 4'</pre>
```

require (library_name)

Load the library library_name into the R interpreter.

This function raises an ImportError if the given library is not known.

INPUT:

•library_name - string

EXAMPLES:

```
sage: r.library('grid')
sage: 'grid' in r.eval('(.packages())')
True
sage: r.library('foobar')
Traceback (most recent call last):
...
ImportError: ...
```

set (var, value)

Set the variable var in R to what the string value evaluates to in R.

INPUT:

- •var a string
- •value a string

```
sage: r.set('a', '2 + 3')
sage: r.get('a')
'[1] 5'
```

```
set seed ( seed=None)
```

Sets the seed for R interpeter. The seed should be an integer.

EXAMPLES:

```
sage: r = R()
sage: r.set_seed(1)
1
sage: r.sample("1:10", 5)
[1] 3 4 5 7 2
```

source(s)

Display the R source (if possible) about the function named s.

INPUT:

•s – a string representing the function whose source code you want to see

OUTPUT: string - source code

EXAMPLES:

```
sage: print(r.source("c"))
function (..., recursive = FALSE) .Primitive("c")
```

version ()

Return the version of R currently running.

OUTPUT: tuple of ints; string

EXAMPLES:

```
sage: r.version() # not tested
((3, 0, 1), 'R version 3.0.1 (2013-05-16)')
sage: rint, rstr = r.version()
sage: rint[0] >= 3
True
sage: rstr.startswith('R version')
True
```

```
class sage.interfaces.r. RElement ( parent, value, is_name=False, name=None)
```

```
Bases: sage.interfaces.tab_completion.ExtraTabCompletion sage.interfaces.expect.ExpectElement
```

dot_product (other)

Implements the notation self. other.

INPUT:

•self, other – R elements

OUTPUT: R element

```
sage: c = r.c(1,2,3,4)
sage: c.dot_product(c.t())
    [,1] [,2] [,3] [,4]
[1,]
      1 2
               3
[2,]
     2
               6
                   8
           4
         6
               9
[3,]
      3
                  12
[4,]
      4
          8
              12
                  16
```

```
sage: v = r([3,-1,8])
sage: v.dot_product(v)
      [,1]
[1,] 74
```

stat_model (x)

The tilde regression operator in R.

EXAMPLES:

```
sage: x = r([1,2,3,4,5])
sage: y = r([3,5,7,9,11])
sage: a = r.lm( y.tilde(x) ) # lm( y ~ x )
sage: d = a._sage_()
sage: d['DATA']['coefficients']['DATA'][1]
2
```

tilde (x)

The tilde regression operator in R.

EXAMPLES:

```
sage: x = r([1,2,3,4,5])
sage: y = r([3,5,7,9,11])
sage: a = r.lm( y.tilde(x) ) # lm( y ~ x )
sage: d = a._sage_()
sage: d['DATA']['coefficients']['DATA'][1]
2
```

class sage.interfaces.r. RFunction (parent, name, r_name=None)

Bases: sage.interfaces.expect.ExpectFunction

A Function in the R interface.

INPUT:

•parent – the R interface

•name – the name of the function for Python

•r_name – the name of the function in R itself (which can have dots in it)

EXAMPLES:

```
sage: length = r.length
sage: type(length)
<class 'sage.interfaces.r.RFunction'>
sage: loads(dumps(length))
length
```

```
class sage.interfaces.r. RFunctionElement (obj, name)
```

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.r. is_RElement (x)

Return True if x is an element in an R interface.

INPUT:

•x – object

OUTPUT: bool

EXAMPLES:

```
sage: from sage.interfaces.r import is_RElement
sage: is_RElement(2)
False
sage: is_RElement(r(2))
True
```

sage.interfaces.r. r_console ()

Spawn a new R command-line session.

EXAMPLES:

```
sage: r.console() # not tested

R version 2.6.1 (2007-11-26)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
...
```

sage.interfaces.r. r_version()

Return the R version.

EXAMPLES:

```
sage: r_version() # not tested
((3, 0, 1), 'R version 3.0.1 (2013-05-16)')
sage: rint, rstr = r_version()
sage: rint[0] >= 3
True
sage: rstr.startswith('R version')
True
```

sage.interfaces.r. reduce_load_R ()

Used for reconstructing a copy of the R interpreter from a pickle.

```
sage: from sage.interfaces.r import reduce_load_R
sage: reduce_load_R()
R Interpreter
```

CHAPTER

INTERFACE TO SEVERAL RUBIK'S CUBE SOLVERS.

The first is by Michael Reid, and tries to find an optimal solution given the cube's state, and may take a long time. See http://www.math.ucf.edu/~reid/Rubik/optimal solver.html

The second is by Eric Dietz, and uses a standard (?) algorithm to solve the cube one level at a time. It is extremely fast, but often returns a far from optimal solution. See http://wrongway.org/?rubiksource

The third is by Dik Winter and implements Kociemba's algorithm which finds reasonable solutions relatively quickly, and if it is kept running will eventually find the optimal solution.

AUTHOR: - Optimal was written by Michael Reid <reid@math.ucf.edu> (2004) - Cubex was written by Eric Dietz <root@wrongway.org> (2003) - Kociemba was written by Dik T. Winter <dik.winter@cwi.nl> (1993) - Initial interface by Robert Bradshaw (2007-08)

class sage.interfaces.rubik. CubexSolver

```
format_cube (facets)
solve (facets)
```

EXAMPLES: sage: from sage.interfaces.rubik import * sage: C = RubiksCube("R U") sage: Cubex-Solver().solve(C.facets()) 'R U' sage: C = RubiksCube("R U F L B D") sage: sol = Cubex-Solver().solve(C.facets()); sol "U' L' L' U L U' L U D L L D' L' D L' D' L D L' U' L D' L' U L' B' U' L' U B L D L D' U' L' U L B L B' L' U L U' L' F' L' F L' F L' D' L' D D L D' B L B' L B' L B F' L F F B' L F' B D' D' L D B' B' L' D' B U' U' L' B' D' F' F' L D F''' sage: RubiksCube(sol) == C True sage: C = RubiksCube("R2 F"'') sage: CubexSolver().solve(C.facets()) "R' R' F''' sage: C = RubiksCube().scramble() sage: sol = CubexSolver().solve(C.facets()) sage: C = RubiksCube(sol) True

class sage.interfaces.rubik. DikSolver

```
format_cube (facets)
solve (facets, timeout=10, extra_time=2)
```

EXAMPLES: sage: from sage.interfaces.rubik import * sage: C = RubiksCube().move("R U") sage: DikSolver().solve(C.facets()) 'R U' sage: C = RubiksCube().move("R U F L B D") sage: DikSolver().solve(C.facets()) 'R U F L B D' sage: C = RubiksCube().move("R2 F"') sage: DikSolver().solve(C.facets()) "R2 F"'

class sage.interfaces.rubik. OptimalSolver (verbose=False, wait=True)
 Interface to Michael Reid's optimal Rubik's Cube solver.

```
format_cube (facets)
ready ( )
```

```
solve ( facets)
    The initial startup and precomputation are substantial...

TODO: Let it keep searching once it found a solution?

EXAMPLES: sage: from sage.interfaces.rubik import * sage: solver = DikSolver() sage: solver =
    OptimalSolver() # long time (28s on sage.math, 2012) Initializing tables... Done. sage: C = RubiksCube("R U") sage: solver.solve(C.facets()) 'R U' sage: C = RubiksCube("R U F L B D") sage:
    solver.solve(C.facets()) 'R U F L B D' sage: C = RubiksCube("R2 D2") sage: solver.solve(C.facets())
    'R2 D2'

start ( )
    stop ( )

class sage.interfaces.rubik. SingNot ( s)
```

This class is to resolve difference between various Singmaster notation. Case is ignored, and the second and third letters may be swapped.

EXAMPLE: sage: from sage.interfaces.rubik import SingNot sage: SingNot("acb") == SingNot("ACB") True sage: SingNot("acb") == SingNot("bca") False

CHAPTER

THIRTYEIGHT

INTERFACE TO SAGE

This is an expect interface to *another* copy of the Sage interpreter.

Expect interface to the Sage interpreter itself.

INPUT:

•server - (optional); if specified runs Sage on a remote machine with address. You must have ssh keys setup so you can login to the remote machine by typing "ssh remote_machine" and no password, call _install_hints_ssh() for hints on how to do that.

The version of Sage should be the same as on the local machine, since pickling is used to move data between the two Sage process.

EXAMPLES: We create an interface to a copy of Sage. This copy of Sage runs as an external process with its own memory space, etc.

```
sage: s = Sage()
```

Create the element 2 in our new copy of Sage, and cube it.

```
sage: a = s(2)
sage: a^3
```

Create a vector space of dimension 4, and compute its generators:

```
sage: V = s('QQ^4')
sage: V.gens()
((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1))
```

Note that V is a not a vector space, it's a wrapper around an object (which happens to be a vector space), in another running instance of Sage.

```
sage: type(V)
<class 'sage.interfaces.sage0.SageElement'>
sage: V.parent()
Sage
sage: g = V.0; g
(1, 0, 0, 0)
```

```
sage: g.parent()
Sage
```

We can still get the actual parent by using the name attribute of g, which is the variable name of the object in the child process.

```
sage: s('%s.parent()'%g.name())
Vector space of dimension 4 over Rational Field
```

Note that the memory space is completely different.

```
sage: x = 10
sage: s('x = 5')
5
sage: x
10
sage: s('x')
```

We can have the child interpreter itself make another child Sage process, so now three copies of Sage are running:

```
sage: s3 = s('Sage()')
sage: a = s3(10)
sage: a
10
```

This a = 10 is in a subprocess of a subprocesses of your original Sage.

```
sage: _ = s.eval('%s.eval("x=8")'%s3.name())
sage: s3('"x"')
8
sage: s('x')
5
sage: x
10
```

The double quotes are needed because the call to s3 first evaluates its arguments using the s interpreter, so the call to s3 is passed s("x""), which is the string "x" in the s interpreter.

clear (var)

Clear the variable named var.

Note that the exact format of the NameError for a cleared variable is slightly platform dependent, see trac ticket #10539.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
sage: sage0.clear('x')
sage: 'NameError' in sage0.get('x')
True
```

console ()

Spawn a new Sage command-line session.

```
sage: sage0.console() #not tested

| SageMath version ..., Release Date: ... |
| Type notebook() for the GUI, and license() for information. |
...
```

cputime (t=None)

Return cputime since this Sage subprocess was started.

EXAMPLES:

eval (line, strip=True, **kwds)

Send the code x to a second instance of the Sage interpreter and return the output as a string.

This allows you to run two completely independent copies of Sage at the same time in a unified way.

INPUT:

- •line input line of code
- •strip -ignored

EXAMPLES:

```
sage: sage0.eval('2+2')
'4'
```

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
```

$\mathbf{new} \ (\ x)$

EXAMPLES:

```
sage: sage0.new(2)
2
sage: _.parent()
Sage
```

preparse(x)

Returns the preparsed version of the string s.

```
sage: sage0.preparse('2+2')
'Integer(2)+Integer(2)'
```

```
set (var, value)
```

Set the variable var to the given value.

EXAMPLES:

```
sage: sage0.set('x', '2')
sage: sage0.get('x')
'2'
```

version ()

EXAMPLES:

```
sage: sage0.version()
'SageMath version ..., Release Date: ...'
sage: sage0.version() == version()
True
```

```
class sage.interfaces.sage0. SageElement (parent, value, is_name=False, name=None)
```

Bases: sage.interfaces.expect.ExpectElement

```
class sage.interfaces.sage0. SageFunction (obj, name)
```

Bases: sage.interfaces.expect.FunctionElement

sage.interfaces.sage0. reduce_load_Sage ()

EXAMPLES:

```
sage: from sage.interfaces.sage0 import reduce_load_Sage
sage: reduce_load_Sage()
Sage
```

sage.interfaces.sage0. reduce_load_element (s)

EXAMPLES:

```
sage: from sage.interfaces.sage0 import reduce_load_element
sage: s = dumps(1/2)
sage: half = reduce_load_element(s); half
1/2
sage: half.parent()
Sage
```

sage.interfaces.sage0. sage0_console ()

Spawn a new Sage command-line session.

EXAMPLES:

sage.interfaces.sage0.sage0_version()

```
sage: from sage.interfaces.sage0 import sage0_version
sage: sage0_version() == version()
True
```

CHAPTER

THIRTYNINE

INTERFACE TO SCILAB

Scilab is a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications. Scilab includes hundreds of mathematical functions with the possibility to add interactively programs from various languages (C, C++, Fortran...). It has sophisticated data structures (including lists, polynomials, rational functions, linear systems...), an interpreter and a high level programming language.

The commands in this section only work if you have the "scilab" interpreter installed and available in your PATH. It's not necessary to install any special Sage packages.

EXAMPLES:

Tutorial based the MATLAB interface tutorial:

```
# optional - scilab
sage: scilab('4+10')
14.
sage: scilab('date')
                                         # optional - scilab; random output
15-Feb-2010
sage: scilab('5*10 + 6')
                                         # optional - scilab
sage: scilab('(6+6)/3')
                                         # optional - scilab
sage: scilab('9')^2
                                         # optional - scilab
sage: a = scilab(10); b = scilab(20); c = scilab(30)
                                                        # optional - scilab
                                         # optional - scilab
sage: avg = (a+b+c)/3
sage: avg
                                         # optional - scilab
20.
                                         # optional - scilab
sage: parent(avg)
Scilab
sage: my_scalar = scilab('3.1415')
                                         # optional - scilab
sage: my_scalar
                                          # optional - scilab
3.1415
                                         # optional - scilab
sage: my_vector1 = scilab('[1,5,7]')
                                          # optional - scilab
sage: my_vector1
      5.
```

```
sage: my_vector2 = scilab('[1;5;7]') # optional - scilab
sage: my vector2
                                     # optional - scilab
1.
5.
sage: my_vector1 * my_vector2 # optional - scilab
75.
sage: row_vector1 = scilab('[1 2 3]')
sage: row_vector2 = scilab('[3 2 1]')
                                             # optional - scilab
                                             # optional - scilab
sage: matrix_from_row_vec = scilab('[%s; %s]'%(row_vector1.name(), row_vector2.
→name())) # optional - scilab
                                              # optional - scilab
sage: matrix_from_row_vec
1. 2. 3.
3.
     2.
sage: column_vector1 = scilab('[1;3]') # optional - scilab
sage: column_vector2 = scilab('[2;8]')
                                               # optional - scilab
sage: matrix_from_col_vec = scilab('[%s %s]'%(column_vector1.name(), column_vector2.
\hookrightarrowname()))
                                        # optional - scilab
sage: matrix_from_col_vec
                                                 # optional - scilab
1.
    2..
3.
sage: my_matrix = scilab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]') # optional -__
⇔scilab
sage: my_matrix
                                                # optional - scilab
   8. 12. 19.
         3.
   7.
                2.
               23.
   12. 4.
   8.
        1.
                1.
sage: combined_matrix = scilab('[%s, %s]'%(my_matrix.name(), my_matrix.name()))
                     # optional - scilab
sage: combined_matrix
                                               # optional - scilab
    7.
                  12. 4.
     4.
           23.
                               23.
12.
                        1.
                 8.
     1.
           1.
                               1.
sage: tm = scilab('0.5:2:10')
                                               # optional - scilab
                                               # optional - scilab
sage: tm
0.5 2.5 4.5 6.5 8.5
sage: my_vector1 = scilab('[1,5,7]')
                                              # optional - scilab
                                               # optional - scilab
sage: my_vector1(1)
sage: my_vector1(2)
                                               # optional - scilab
                                               # optional - scilab
sage: my_vector1(3)
```

Matrix indexing works as follows:

One can also use square brackets:

```
sage: my_matrix[3,2] # optional - scilab
4.
```

Setting using parenthesis cannot work (because of how the Python language works). Use square brackets or the set function:

```
sage: my_matrix = scilab('[8, 12, 19; 7, 3, 2; 12, 4, 23; 8, 1, 1]')
                                                                      # optional -_
⇔scilab
sage: my_matrix.set(2,3, 1999)
                                                      # optional - scilab
sage: my_matrix
                                                       # optional - scilab
      8.
                12.
                           19.
      7.
                 3.
                          1999.
     12.
                  4.
                           23.
      8.
                  1.
                             1.
sage: my_matrix[2,3] = -126
                                                      # optional - scilab
sage: my_matrix
                                                      # optional - scilab
                 12.
                             19.
      8.
      7.
                  3.
                          - 126.
     12.
                            23.
                  4.
      8.
                  1.
                             1.
```

TESTS:

```
sage: M = scilab(x)
                                                         # optional - scilab
Traceback (most recent call last):
TypeError: _interface_init_() takes exactly one argument (0 given)
sage: M = scilab(matrix(3, range(9))); M
                                                         # optional - scilab
   0. 1. 2.
    3.
         4.
               5.
    6.
          7.
               8.
sage: M(10)
                                                         # optional - scilab
Traceback (most recent call last):
TypeError: Error executing code in Scilab
Invalid index.
sage: M[10]
                                                        # optional - scilab
Traceback (most recent call last):
TypeError: Error executing code in Scilab
Invalid index.
sage: M(4,2)
                                                        # optional - scilab
Traceback (most recent call last):
TypeError: Error executing code in Scilab
Invalid index.
sage: M[2,4]
                                                        # optional - scilab
Traceback (most recent call last):
TypeError: Error executing code in Scilab
Invalid index.
sage: M(9) = x
                                                         # optional - scilab
Traceback (most recent call last):
```

```
SyntaxError: can't assign to function call (..., line 1)
```

AUTHORS:

- Ronan Paixao (2008-11-26), based on the MATLAB tutorial by William Stein (2006-10-11)

Bases: sage.interfaces.expect.Expect

Interface to the Scilab interpreter.

EXAMPLES:

console ()

Starts Scilab console.

EXAMPLES:

```
sage: scilab.console() # optional - scilab; not tested
```

eval (command, *args, **kwds)

Evaluates commands.

EXAMPLES:

```
sage: scilab.eval("5") # optional - scilab
'ans =
```

5.' sage: scilab.eval("d=44") # optional - scilab 'd =

44.

get (var)

Get the value of the variable var.

EXAMPLES:

```
sage: scilab.eval('b=124;') # optional - scilab

sage: scilab.get('b') # optional - scilab
''
```

124.

sage2scilab_matrix_string (A)

Return a Scilab matrix from a Sage matrix.

INPUT: A Sage matrix with entries in the rationals or reals.

OUTPUT: A string that evaluates to an Scilab matrix.

EXAMPLES:

```
sage: M33 = MatrixSpace(QQ,3,3)  # optional - scilab
sage: A = M33([1,2,3,4,5,6,7,8,0])  # optional - scilab
sage: scilab.sage2scilab_matrix_string(A)  # optional - scilab
'[1, 2, 3; 4, 5, 6; 7, 8, 0]'
```

set (var, value)

Set the variable var to the given value.

EXAMPLES:

```
sage: scilab.set('a', 123) # optional - scilab
sage: scilab.get('a') # optional - scilab
'
```

123.

set seed (seed=None)

Sets the seed for gp interpeter. The seed should be an integer.

EXAMPLES:

version ()

Returns the version of the Scilab software used.

EXAMPLES:

```
sage: scilab.version() # optional - scilab
'scilab-...'
```

whos (name=None, typ=None)

Returns information about current objects. Arguments: nam: first characters of selected names typ: name of selected Scilab variable type

```
class sage.interfaces.scilab. ScilabElement ( parent, value, is_name=False, name=None)
```

Bases: sage.interfaces.expect.ExpectElement

```
set (i, j, x)
```

Set the variable var to the given value.

EXAMPLES:

```
sage: scilab.set('c', 125) # optional - scilab
sage: scilab.get('c') # optional - scilab
'
```

125.

```
sage.interfaces.scilab. scilab_console ()
```

This requires that the optional Scilab program be installed and in your PATH, but no optional Sage packages need to be installed.

EXAMPLES:

Typing quit exits the Scilab console and returns you to Sage. Scilab, like Sage, remembers its history from one session to another.

```
sage.interfaces.scilab. scilab_version ()
```

Return the version of Scilab installed.

```
sage: from sage.interfaces.scilab import scilab_version # optional - scilab
sage: scilab_version() # optional - scilab
'scilab-...'
```

INTERFACE TO SINGULAR

AUTHORS:

- David Joyner and William Stein (2005): first version
- Martin Albrecht (2006-03-05): code so singular.[tab] and x = singular(...), x.[tab] includes all singular commands.
- Martin Albrecht (2006-03-06): This patch adds the equality symbol to singular. Also fix a problem in which "" as prompt means comparison will break all further communication with Singular.
- Martin Albrecht (2006-03-13): added current_ring() and current_ring_name()
- William Stein (2006-04-10): Fixed problems with ideal constructor
- Martin Albrecht (2006-05-18): added sage_poly.
- Simon King (2010-11-23): Reduce the overhead caused by waiting for the Singular prompt by doing garbage collection differently.
- Simon King (2011-06-06): Make conversion from Singular to Sage more flexible.
- Simon King (2015): Extend pickling capabilities.

40.1 Introduction

This interface is extremely flexible, since it's exactly like typing into the Singular interpreter, and anything that works there should work here.

The Singular interface will only work if Singular is installed on your computer; this should be the case, since Singular is included with Sage. The interface offers three pieces of functionality:

- 1. singular_console() A function that dumps you into an interactive command-line Singular session.
- 2. singular (expr, type='def') Creation of a Singular object. This provides a Pythonic interface to Singular. For example, if f=singular(10), then f.factorize() returns the factorization of 10 computed using Singular.
- 3. singular.eval(expr) Evaluation of arbitrary Singular expressions, with the result returned as a string.

Of course, there are polynomial rings and ideals in Sage as well (often based on a C-library interface to Singular). One can convert an object in the Singular interpreter interface to Sage by the method sage ().

40.2 Tutorial

EXAMPLES: First we illustrate multivariate polynomial factorization:

```
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: R1
polynomial ring, over a field, global ordering
// characteristic : 0
// number of vars : 2
//
          block 1 : ordering dp
//
                         : names
//
            block 2 : ordering C
sage: f = singular('9x16 - 18x13y2 - 9x12y3 + 9x10y4 - 18x11y2 + 36x8y4 + 18x7y5 - ...)
\hookrightarrow 18x5y6 + 9x6y4 - 18x3y6 - 9x2y7 + 9y8')
sage: f
9 \times x^{16} - 18 \times x^{13} \times y^{2} - 9 \times x^{12} \times y^{3} + 9 \times x^{10} \times y^{4} - 18 \times x^{11} \times y^{2} + 36 \times x^{8} \times y^{4} + 18 \times x^{7} \times y^{5} - 18 \times x^{5} \times y^{6}
\leftrightarrow 6+9*x^6*y^4-18*x^3*y^6-9*x^2*y^7+9*y^8
sage: f.parent()
Singular
```

```
sage: F = f.factorize(); F
[1]:
    _[1]=9
    _[2]=x^6-2*x^3*y^2-x^2*y^3+y^4
    _[3]=-x^5+y^2
[2]:
    1,1,2
```

```
sage: F[1]
9,
x^6-2*x^3*y^2-x^2*y^3+y^4,
-x^5+y^2
sage: F[1][2]
x^6-2*x^3*y^2-x^2*y^3+y^4
```

We can convert f and each exponent back to Sage objects as well.

This example illustrates polynomial GCD's:

```
sage: R2 = singular.ring(0, '(x,y,z)', 'lp')
sage: a = singular.new('3x2*(x+y)')
sage: b = singular.new('9x*(y2-x2)')
sage: g = a.gcd(b)
sage: g
x^2+x*y
```

This example illustrates computation of a Groebner basis:

```
c^2*d^6-c^2*d^2-d^4+1,

c^3*d^2+c^2*d^3-c-d,

b*d^4-b+d^5-d,

b*c-b*d^5+c^2*d^4+c*d-d^6-d^2,

b^2+2*b*d+d^2,

a+b+c+d
```

The following example is the same as the one in the Singular - Gap interface documentation:

This example illustrates moving a polynomial from one ring to another. It also illustrates calling a method of an object with an argument.

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: f = singular('x3+y3+(x-y)*x2y2+z2')
sage: f
x^3*y^2-x^2*y^3+x^3+y^3+z^2
sage: R1 = singular.ring(0, '(x,y,z)', 'ds')
sage: f = R.fetch(f)
sage: f
z^2+x^3+y^3+x^3*y^2-x^2*y^3
```

We can calculate the Milnor number of f:

```
sage: _=singular.LIB('sing.lib') # assign to _ to suppress printing
sage: f.milnor()
4
```

The Jacobian applied twice yields the Hessian matrix of f, with which we can compute.

```
sage: H = f.jacob().jacob()
sage: H
6*x+6*x*y^2-2*y^3, 6*x^2*y-6*x*y^2, 0,
6*x^2*y-6*x*y^2, 6*y+2*x^3-6*x^2*y, 0,
                                                                                        Ο,
0,
sage: H.sage()
 [6*x + 6*x*y^2 - 2*y^3    6*x^2*y - 6*x*y^2]
                                                                                                                                                                                                                                                                                                                                    0]
                 6*x^2*y - 6*x*y^2 6*y + 2*x^3 - 6*x^2*y
                                                                                                                                                                                                                                                                                                                                     01
                                                                                                        0
                                                                                                                                                                                                                                                                                                                                      21
                                                                       # This is a polynomial in Singular
sage: H.det()
72 * x * y + 24 * x ^4 - 72 * x ^3 * y + 72 * x * y ^3 - 24 * y ^4 - 48 * x ^4 * y ^2 + 64 * x ^3 * y ^3 - 48 * x ^2 * y ^4 + 24 * x ^4 * y ^4 +
sage: H.det().sage() # This is the corresponding polynomial in Sage
72 * x * y + 24 * x^4 - 72 * x^3 * y + 72 * x * y^3 - 24 * y^4 - 48 * x^4 * y^2 + 64 * x^3 * y^3 - 48 * x^2 * y^4
```

The 1x1 and 2x2 minors:

40.2. Tutorial 297

```
sage: H.minor(1)
2,
6*y+2*x^3-6*x^2*y,
6*x^2*y-6*x*y^2,
6*x^2*y-6*x*y^2,
6*x+6*x*y^2-2*y^3
sage: H.minor(2)
12*y+4*x^3-12*x^2*y,
12*x^2*y-12*x*y^2,
12*x^2*y-12*x*y^2,
12*x+12*x*y^2-4*y^3,
-36*x*y-12*x^4+36*x^3*y-36*x*y^3+12*y^4+24*x^4*y^2-32*x^3*y^3+24*x^2*y^4
```

```
sage: _=singular.eval('option(redSB)')
sage: H.minor(1).groebner()
1
```

40.3 Computing the Genus

We compute the projective genus of ideals that define curves over Q. It is *very important* to load the normal.lib library before calling the genus command, or you'll get an error message.

EXAMPLE:

```
sage: singular.lib('normal.lib')
sage: R = singular.ring(0,'(x,y)','dp')
sage: i2 = singular.ideal('y9 - x2*(x-1)^9 + x')
sage: i2.genus()
40
```

Note that the genus can be much smaller than the degree:

```
sage: i = singular.ideal('y9 - x2*(x-1)^9')
sage: i.genus()
0
```

40.4 An Important Concept

AUTHORS:

· Neal Harris

The following illustrates an important concept: how Sage interacts with the data being used and returned by Singular. Let's compute a Groebner basis for some ideal, using Singular through Sage.

```
sage: I = singular.ideal('cyclic(6)')
sage: g = singular('groebner(I)')
Traceback (most recent call last):
...
TypeError: Singular error:
...
```

We restart everything and try again, but correctly.

```
sage: singular.quit()
sage: singular.lib('poly.lib'); R = singular.ring(32003, '(a,b,c,d,e,f)', 'lp')
sage: I = singular.ideal('cyclic(6)')
sage: I.groebner()
f^48-2554*f^42-15674*f^36+12326*f^30-12326*f^18+15674*f^12+2554*f^6-1,
...
```

It's important to understand why the first attempt at computing a basis failed. The line where we gave singular the input 'groebner(I)' was useless because Singular has no idea what 'I' is! Although 'I' is an object that we computed with calls to Singular functions, it actually lives in Sage. As a consequence, the name 'I' means nothing to Singular. When we called I.groebner(), Sage was able to call the groebner function on'I' in Singular, since 'I' actually means something to Sage.

40.5 Long Input

The Singular interface reads in even very long input (using files) in a robust manner, as long as you are creating a new object.

TESTS:

We test an automatic coercion:

```
sage: a = 3*singular('2'); a
6
sage: type(a)
<class 'sage.interfaces.singular.SingularElement'>
sage: a = singular('2')*3; a
6
sage: type(a)
<class 'sage.interfaces.singular.SingularElement'>
```

Create a ring over GF(9) to check that gftables has been installed, see trac ticket #11645:

```
sage: singular.eval("ring testgf9 = (9,x), (a,b,c,d,e,f), (M((1,2,3,0)),wp(2,3),lp);")
```

Interface to the Singular interpreter.

EXAMPLES: A Groebner basis example.

AUTHORS:

•David Joyner and William Stein

LIB (*lib*, *reload=False*)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

clear (var)

Clear the variable named var.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
sage: singular.clear('x')
```

"Clearing the variable" means to allow to free the memory that it uses in the Singular sub-process. However, the actual deletion of the variable is only committed when the next element in the Singular interface is created:

```
sage: singular.get('x')
'2'
sage: a = singular(3)
sage: singular.get('x')
'`x`'
```

console ()

EXAMPLES:

cputime (t=None)

Returns the amount of CPU time that the Singular session has used. If t is not None, then it returns the

difference between the current CPU time and t .

EXAMPLES:

current_ring()

Returns the current ring of the running Singular session.

EXAMPLES:

```
sage: r = PolynomialRing(GF(127), 3, 'xyz', order='invlex')
sage: r. singular ()
polynomial ring, over a field, global ordering
  characteristic : 127
//
//
   number of vars : 3
//
       block 1 : ordering rp
//
                : names x y z
// block 2 : ordering C
sage: singular.current_ring()
polynomial ring, over a field, global ordering
// characteristic : 127
//
   number of vars : 3
//
    block 1 : ordering rp
//
                 : names x y z
//
    block 2 : ordering C
```

current_ring_name ()

Returns the Singular name of the currently active ring in Singular.

OUTPUT: currently active ring's name

EXAMPLES:

```
sage: r = PolynomialRing(GF(127),3,'xyz')
sage: r._singular_().name() == singular.current_ring_name()
True
```

eval (x, allow_semicolon=True, strip=True, **kwds)

Send the code x to the Singular interpreter and return the output as a string.

INPUT:

- •x string (of code)
- •allow_semicolon default: False; if False then raise a TypeError if the input line contains a semicolon.
- •strip -ignored

EXAMPLES:

```
sage: singular.eval('2 > 1')
'1'
sage: singular.eval('2 + 2')
'4'
```

if the verbosity level is > 1 comments are also printed and not only returned.

```
sage: r = singular.ring(0,'(x,y,z)','dp')
sage: i = singular.ideal(['x^2','y^2','z^2'])
sage: s = i.std()
sage: singular.eval('hilb(%s)'%(s.name()))
'// 1 t^0\n// -3 t^2\n// 3 t^4\n// -1 t^6\n\n// 1 t^0\n//
3 t^1\n// 3 t^2\n// 1 t^3\n// dimension (affine) = 0\n//
degree (affine) = 8'
```

```
sage: set_verbose(1)
sage: o = singular.eval('hilb(%s)'%(s.name()))
          1 t^0
//
//
         -3 t^2
//
          3 t^4
//
         -1 t^6
//
          1 t^0
//
          3 t^1
//
          3 t^2
//
          1 t^3
// dimension (affine) = 0
// degree (affine) = 8
```

This is mainly useful if this method is called implicitly. Because then intermediate results, debugging outputs and printed statements are printed

```
sage: o = s.hilb()
          1 t^0
//
//
         -3 t^2
          3 t^4
//
         -1 t^6
//
//
          1 t^0
          3 t^1
//
//
          3 t^2
//
          1 t^3
// dimension (affine) = 0
// degree (affine) = 8
// ** right side is not a datum, assignment ignored
```

rather than ignored

```
sage: set_verbose(0)
sage: o = s.hilb()
```

get (var)

Get string representation of variable named var.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
```

ideal (*gens)

Return the ideal generated by gens.

INPUT:

•gens - list or tuple of Singular objects (or objects that can be made into Singular objects via evaluation)

OUTPUT: the Singular ideal generated by the given list of gens

EXAMPLES: A Groebner basis example done in a different way.

```
sage: i2 = singular.ideal('groebner(%s);'%i1.name())
sage: i2
x1^2*x2^2,
x0*x2^3-x1^2*x2^2+x1*x2^3,
x0*x1-x0*x2-x1*x2,
x0^2*x2-x0*x2^2-x1*x2^2
```

lib (*lib*, *reload=False*)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

list(x)

Creates a list in Singular from a Sage list x.

EXAMPLES:

```
sage: singular.list([1,2])
[1]:
    1
[2]:
    2
```

load (lib, reload=False)

Load the Singular library named lib.

Note that if the library was already loaded during this session it is not reloaded unless the optional reload argument is True (the default is False).

EXAMPLES:

```
sage: singular.lib('sing.lib')
sage: singular.lib('sing.lib', reload=True)
```

matrix (nrows, ncols, entries=None)

EXAMPLES:

```
sage: singular.lib("matrix")
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: A = singular.matrix(3,2,'1,2,3,4,5,6')
sage: A
1,2,
3,4,
5,6
sage: A.gauss_col()
2,-1,
1,0,
0,1
```

AUTHORS:

•Martin Albrecht (2006-01-14)

option (cmd=None, val=None)

Access to Singular's options as follows:

Syntax: option() Returns a string of all defined options.

Syntax: option('option_name') Sets an option. Note to disable an option, use the prefix no.

Syntax: option('get') Returns an intvec of the state of all options.

Syntax: option('set', intvec_expression) Restores the state of all options from an intvec (produced by option('get')).

EXAMPLES:

```
sage: singular.option()
//options: redefine loadLib usage prompt
sage: singular.option('get')
0,
10321
sage: old_options = _
sage: singular.option('noredefine')
sage: singular.option()
//options: loadLib usage prompt
sage: singular.option('set', old_options)
sage: singular.option('get')
0,
10321
```

ring (char=0, vars='(x)', order='lp', check=True)

Create a Singular ring and makes it the current ring.

INPUT:

- •char characteristic of the base ring (see examples below), which must be either 0, prime (!), or one of several special codes (see examples below).
- •vars a tuple or string that defines the variable names
- •order string the monomial order (default: 'lp')
- •check if True, check primality of the characteristic if it is an integer.

OUTPUT: a Singular ring

Note: This function is *not* identical to calling the Singular ring function. In particular, it also attempts to "kill" the variable names, so they can actually be used without getting errors, and it sets printing of

elements for this range to short (i.e., with *'s and carets).

EXAMPLES: We first declare $\mathbf{Q}[x,y,z]$ with degree reverse lexicographic ordering.

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: R
polynomial ring, over a field, global ordering
// characteristic : 0
// number of vars : 3
// block 1 : ordering dp
// : names x y z
// block 2 : ordering C
```

```
sage: R1 = singular.ring(32003, '(x,y,z)', 'dp')
sage: R2 = singular.ring(32003, '(a,b,c,d)', 'lp')
```

This is a ring in variables named x(1) through x(10) over the finite field of order 7:

```
sage: R3 = singular.ring(7, '(x(1..10))', 'ds')
```

This is a polynomial ring over the transcendental extension $\mathbf{Q}(a)$ of \mathbf{Q} :

```
sage: R4 = singular.ring('(0,a)', '(mu,nu)', 'lp')
```

This is a ring over the field of single-precision floats:

```
sage: R5 = singular.ring('real', '(a,b)', 'lp')
```

This is over 50-digit floats:

```
sage: R6 = singular.ring('(real,50)', '(a,b)', 'lp')
sage: R7 = singular.ring('(complex,50,i)', '(a,b)', 'lp')
```

To use a ring that you've defined, use the set_ring() method on the ring. This sets the ring to be the "current ring". For example,

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.new('10*a')
(1.000e+01)*a
sage: R.set_ring()
sage: singular.new('10*a')
3*a
```

set (type, name, value)

Set the variable with given name to the given value.

REMARK:

If a variable in the Singular interface was previously marked for deletion, the actual deletion is done here, before the new variable is created in Singular.

EXAMPLES:

```
sage: singular.set('int', 'x', '2')
sage: singular.get('x')
'2'
```

We test that an unused variable is only actually deleted if this method is called:

```
sage: a = singular(3)
sage: n = a.name()
sage: del a
sage: singular.eval(n)
'3'
sage: singular.set('int', 'y', '5')
sage: singular.eval('defined(%s)'%n)
'0'
```

set ring(R)

Sets the current Singular ring to R.

EXAMPLES:

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
polynomial ring, over a field, global ordering
// characteristic : 0 (real)
//
   number of vars : 2
//
     block 1 : ordering lp
//
                  : names a b
//
        block 2 : ordering C
sage: singular.set_ring(R)
sage: singular.current_ring()
polynomial ring, over a field, local ordering
// characteristic : 7
//
   number of vars : 2
//
    block 1 : ordering ds
//
                 : names
                           a b
       block 2 : ordering C
//
```

set_seed (seed=None)

Sets the seed for singular interpeter. The seed should be an integer at least 1 and not more than 30 bits. See http://www.singular.uni-kl.de/Manual/html/sing_19.htm#SEC26 and http://www.singular.uni-kl.de/Manual/html/sing_283.htm#SEC323

EXAMPLES:

```
sage: s = Singular()
sage: s.set_seed(1)
1
sage: [s.random(1,10) for i in range(5)]
[8, 10, 4, 9, 1]
```

setring(R)

Sets the current Singular ring to R.

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
polynomial ring, over a field, global ordering
// characteristic : 0 (real)
// number of vars : 2
// block 1 : ordering lp
```

```
//
                   : names
         block 2 : ordering C
//
sage: singular.set_ring(R)
sage: singular.current_ring()
polynomial ring, over a field, local ordering
    characteristic : 7
//
   number of vars : 2
//
       block 1 : ordering ds
//
                  : names a b
//
         block 2 : ordering C
```

string(x)

Creates a Singular string from a Sage string. Note that the Sage string has to be "double-quoted".

EXAMPLES:

```
sage: singular.string('"Sage"')
Sage
```

version ()

Return the version of Singular being used.

EXAMPLES:

```
sage: singular.version()
"Singular ... version 4.1.0 ...
```

EXAMPLES:

```
sage: a = singular(2)
sage: loads(dumps(a))
2
```

attrib (name, value=None)

Get and set attributes for self.

INPUT:

•name - string to choose the attribute

•value - boolean value or None for reading, (default:None)

VALUES: isSB - the standard basis property is set by all commands computing a standard basis like groebner, std, stdhilb etc.; used by lift, dim, degree, mult, hilb, vdim, kbase isHomog - the weight vector for homogeneous or quasihomogeneous ideals/modules isCI - complete intersection property isCM - Cohen-Macaulay property rank - set the rank of a module (see nrows) withSB - value of type ideal, resp. module, is std withHilb - value of type intvec is hilb(_,1) (see hilb) withRes - value of type list is a free resolution withDim - value of type int is the dimension (see dim) withMult - value of type int is the multiplicity (see mult)

EXAMPLE:

```
sage: P.<x,y,z> = PolynomialRing(QQ)
sage: I = Ideal([z^2, y*z, y^2, x*z, x*y, x^2])
sage: Ibar = I._singular_()
sage: Ibar.attrib('isSB')
```

```
sage: singular.eval('vdim(%s)'%Ibar.name()) # sage7 name is random
// ** sage7 is no standard basis

sage: Ibar.attrib('isSB',1)
sage: singular.eval('vdim(%s)'%Ibar.name())
'4'
```

is_string()

Tell whether this element is a string.

EXAMPLES:

```
sage: singular('"abc"').is_string()
True
sage: singular('1').is_string()
False
```

sage_flattened_str_list()

EXAMPLES:

```
sage: R=singular.ring(0,'(x,y)','dp')
sage: RL = R.ringlist()
sage: RL.sage_flattened_str_list()
['0', 'x', 'y', 'dp', '1,1', 'C', '0', '_[1]=0']
```

sage_global_ring()

Return the current basering in Singular as a polynomial ring or quotient ring.

```
sage: singular.eval('ring r3 = (3,z),(a,b,c),dp')
''
sage: singular.eval('minpoly = 1+z+z2+z3+z4')
''
sage: singular('r3').sage_global_ring()
Multivariate Polynomial Ring in a, b, c over Finite Field in z of size 3^4
```

Real and complex fields in both Singular and Sage are defined with a precision. The precision in Singular is given in terms of digits, but in Sage it is given in terms of bits. So, the digit precision is internally converted to a reasonable bit precision:

The case of complex coefficients is not fully supported, yet, since the generator of a complex field in Sage is always called "I":

An example where the base ring is a polynomial ring over an extension of the rational field:

In our last example, the base ring is a quotient ring:

```
sage: singular.eval('ring r6 = (9,a), (x,y,z),lp')
''
sage: Q = singular('std(ideal(x^2,x+y^2+z^3))', type='qring')
sage: Q.sage_global_ring()
Quotient of Multivariate Polynomial Ring in x, y, z over Finite Field in a of_
size 3^2 by the ideal (y^4 - y^2*z^3 + z^6, x + y^2 + z^3)
```

AUTHOR:

•Simon King (2011-06-06)

sage_matrix (R, sparse=True)

Returns Sage matrix for self

INPUT:

- •R (default: None); an optional ring, over which the resulting matrix is going to be defined. By default, the output of <code>sage_global_ring()</code> is used.
- •sparse (default: True); determines whether the resulting matrix is sparse or not.

EXAMPLES:

```
sage: R = singular.ring(0, '(x,y,z)', 'dp')
sage: A = singular.matrix(2,2)
sage: A.sage_matrix(ZZ)
[0 0]
[0 0]
sage: A.sage_matrix(RDF)
[0.0 0.0]
[0.0 0.0]
```

sage_poly (R=None, kcache=None)

Returns a Sage polynomial in the ring r matching the provided poly which is a singular polynomial.

INPUT:

- •R (default: None); an optional polynomial ring. If it is provided, then you have to make sure that it matches the current singular ring as, e.g., returned by singular.current_ring(). By default, the output of sage_global_ring() is used.
- •kcache (default: None); an optional dictionary for faster finite field lookups, this is mainly useful for finite extension fields

OUTPUT: MPolynomial

EXAMPLES:

```
sage: R = PolynomialRing(GF(2^8,'a'),2,'xy')
sage: f=R('a^20*x^2*y+a^10+x')
sage: f._singular_().sage_poly(R) == f
True
sage: R = PolynomialRing(GF(2^8,'a'),1,'x')
sage: f=R('a^20*x^3+x^2+a^10')
sage: f._singular_().sage_poly(R) == f
True
```

```
sage: P.<x,y> = PolynomialRing(QQ, 2)
sage: f = x*y**3 - 1/9 * x + 1; f
x*y^3 - 1/9*x + 1
sage: singular(f)
x*y^3-1/9*x+1
sage: P(singular(f))
x*y^3 - 1/9*x + 1
```

TESTS:

```
sage: singular.eval('ring r = (3,z),(a,b,c),dp')
''
sage: singular.eval('minpoly = 1+z+z2+z3+z4')
''
sage: p = singular('z^4*a^3+z^2*a*b*c')
sage: p.sage_poly()
(-z^3 - z^2 - z - 1)*a^3 + (z^2)*a*b*c
sage: singular('z^4')
(-z3-z2-z-1)
```

AUTHORS:

- •Martin Albrecht (2006-05-18)
- •Simon King (2011-06-06): Deal with Singular's short polynomial representation, automatic construction of a polynomial ring, if it is not explicitly given.

Note: For very simple polynomials $eval(SingularElement.sage_polystring())$ is faster than SingularElement.sage_poly(R), maybe we should detect the crossover point (in dependence of the string length) and choose an appropriate conversion strategy

sage_polystring()

If this Singular element is a polynomial, return a string representation of this polynomial that is suitable for evaluation in Python. Thus * is used for multiplication and ** for exponentiation. This function is primarily used internally.

The short=0 option *must* be set for the parent ring or this function will not work as expected. This option is set by default for rings created using singular.ring or set using ring_name.set_ring().

EXAMPLES:

```
sage: R = singular.ring(0,'(x,y)')
sage: f = singular('x^3 + 3*y^11 + 5')
sage: f
x^3+3*y^11+5
sage: f.sage_polystring()
'x**3+3*y**11+5'
```

sage_structured_str_list()

If self is a Singular list of lists of Singular elements, returns corresponding Sage list of lists of strings.

EXAMPLES:

```
sage: R=singular.ring(0,'(x,y)','dp')
sage: RL=R.ringlist()
sage: RL
[1]:
[2]:
   [1]:
   [2]:
[3]:
   [1]:
      [1]:
         db
      [2]:
         1,1
   [2]:
      [1]:
         С
      [2]:
[4]:
  _[1]=0
sage: RL.sage_structured_str_list()
['0', ['x', 'y'], [['dp', '1,\n1 '], ['C', '0 ']], '0']
```

set_ring()

Sets the current ring in Singular to be self.

EXAMPLES:

```
sage: R = singular.ring(7, '(a,b)', 'ds')
sage: S = singular.ring('real', '(a,b)', 'lp')
sage: singular.current_ring()
polynomial ring, over a field, global ordering
// characteristic : 0 (real)
// number of vars : 2
//
    block 1 : ordering lp
//
                : names a b
//
       block 2 : ordering C
sage: R.set_ring()
sage: singular.current_ring()
polynomial ring, over a field, local ordering
   characteristic : 7
//
//
    number of vars : 2
//
     block 1 : ordering ds
                 : names
                           a b
//
//
    block 2 : ordering C
```

type ()

Returns the internal type of this element.

EXAMPLES:

```
sage: R = PolynomialRing(GF(2^8,'a'),2,'x')
sage: R._singular_().type()
'ring'
sage: fs = singular('x0^2','poly')
sage: fs.type()
'poly'
```

```
exception sage.interfaces.singular. SingularError
```

Bases: exceptions.RuntimeError

Raised if Singular printed an error message

```
class sage.interfaces.singular. SingularFunction ( parent, name)
```

Bases: sage.interfaces.expect.ExpectFunction

class sage.interfaces.singular. SingularFunctionElement (obj, name)

Bases: sage.interfaces.expect.FunctionElement

class sage.interfaces.singular. SingularGBDefaultContext (singular=None)

Within this context all Singular Groebner basis calculations are reduced automatically.

AUTHORS:

- •Martin Albrecht
- •Simon King

class sage.interfaces.singular. SingularGBLogPrettyPrinter (verbosity=1)

A device which prints Singular Groebner basis computation logs more verbatim.

flush ()

```
sage: from sage.interfaces.singular import SingularGBLogPrettyPrinter
sage: s3 = SingularGBLogPrettyPrinter(verbosity=3)
sage: s3.flush()
```

write (s)

EXAMPLE:

```
sage: from sage.interfaces.singular import SingularGBLogPrettyPrinter
sage: s3 = SingularGBLogPrettyPrinter(verbosity=3)
sage: s3.write("(S:1337)")
Performing complete reduction of 1337 elements.
sage: s3.write("M[389,12]")
Parallel reduction of 389 elements with 12 non-zero output elements.
```

sage.interfaces.singular.generate_docstring_dictionary ()

Generate global dictionaries which hold the docstrings for Singular functions.

EXAMPLE:

```
sage: from sage.interfaces.singular import generate_docstring_dictionary
sage: generate_docstring_dictionary()
```

sage.interfaces.singular.get_docstring (name)

Return the docstring for the function name.

INPUT:

•name - a Singular function name

EXAMPLE:

```
sage: from sage.interfaces.singular import get_docstring
sage: 'groebner' in get_docstring('groebner')
True
sage: 'standard.lib' in get_docstring('groebner')
True
```

sage.interfaces.singular. is_SingularElement (x)

Returns True is x is of type SingularElement.

EXAMPLES:

```
sage: from sage.interfaces.singular import is_SingularElement
sage: is_SingularElement(singular(2))
True
sage: is_SingularElement(2)
False
```

```
sage.interfaces.singular. reduce_load ()
```

This is for backwards compatibility only.

To be precise, it only serves at unpickling the invalid singular elements that are stored in the pickle jar.

EXAMPLES:

By trac ticket #18848, pickling actually often works:

```
sage: loads(dumps(singular.ring()))
polynomial ring, over a field, global ordering
// characteristic : 0
// number of vars : 1
// block 1 : ordering lp
// : names x
// block 2 : ordering C
```

 ${\tt sage.interfaces.singular.} \ \ {\tt reduce_load_Singular} \ \ (\)$

EXAMPLES:

```
sage: from sage.interfaces.singular import reduce_load_Singular
sage: reduce_load_Singular()
Singular
```

sage.interfaces.singular.singular_console()

Spawn a new Singular command-line session.

EXAMPLES:

sage.interfaces.singular.singular_gb_standard_options (func)

Decorator to force a reduced Singular groebner basis.

TESTS:

```
sage: P.<a,b,c,d,e> = PolynomialRing(GF(127))
sage: J = sage.rings.ideal.Cyclic(P).homogenize()
sage: from sage.misc.sageinspect import sage_getsource
sage: "basis" in sage_getsource(J.interreduced_basis) #indirect doctest
True
```

The following tests against a bug that was fixed in trac ticket #11298:

Note: This decorator is used automatically internally so the user does not need to use it manually.

```
sage.interfaces.singular.singular version()
```

Return the version of Singular being used.

EXAMPLES:

```
sage: singular.version()
"Singular ... version 4.1.0 ...
```

FORTYONE

THE TACHYON RAY TRACER

AUTHOR:

· John E. Stone

```
class sage.interfaces.tachyon. TachyonRT
    Bases: sage.structure.sage_object.SageObject
    The Tachyon Ray Tracer
    tachyon_rt(model, outfile='sage.png', verbose=1, block=True, extra_opts='')
    DINTER
```

- INPUT:
 - •model a string that describes a 3d model in the Tachyon modeling format. Type tachyon_rt.help() for a description of this format.
 - •outfile (default: 'sage.png') output filename; the extension of the filename determines the type. Supported types include:
 - -tga 24-bit (uncompressed)
 - -bmp 24-bit Windows BMP (uncompressed)
 - -ppm 24-bit PPM (uncompressed)
 - -rgb 24-bit SGI RGB (uncompressed)
 - -png 24-bit PNG (compressed, lossless)
 - •verbose integer; (default: 1)
 - -0 silent
 - -1 some output
 - -2 very verbose output
 - •block bool (default: True); if False, run the rendering command in the background.
 - •extra_opts passed directly to tachyon command line. Use tachyon_rt.usage() to see some of the possibilities.

OUTPUT:

- •Some text may be displayed onscreen.
- •The file outfile is created.

```
__call__ ( model, outfile='sage.png', verbose=1, extra_opts='')
This executes the tachyon program, given a scene file input.
```

INPUT:

- •model string. The tachyon model.
- •outfile string, default 'sage.png'. The filename to save the model to.
- •verbose -0, 1, (default) or 2. The verbosity level.
- •extra_opts string (default: empty string). Extra options that will be appended to the tachyon commandline.

EXAMPLES:

```
sage: from sage.interfaces.tachyon import TachyonRT
sage: tgen = Tachyon()
sage: tgen.texture('t1')
sage: tgen.sphere((0,0,0),1,'t1')
sage: tgen.str()[30:40]
'resolution'
sage: t = TachyonRT()
sage: import os
sage: t(tgen.str(), outfile=os.devnull)
tachyon ...
Tachyon Parallel/Multiprocessor Ray Tracer...
```

TESTS:

help (use_pager=True)

Prints (pages) the help file written by John Stone describing scene files for Tachyon. The output is paged unless use_pager=False.

TESTS:

```
sage: from sage.interfaces.tachyon import TachyonRT
sage: t = TachyonRT()
sage: t.help(use_pager=False)
This help, which was written by John Stone, describes ...
```

usage (use_pager=True)

Returns the basic description of using the Tachyon raytracer (simply what is returned by running tachyon with no input). The output is paged unless use_pager=False.

TESTS:

```
sage: from sage.interfaces.tachyon import TachyonRT
sage: t = TachyonRT()
```

sage: t.usage(use_pager=False)
Tachyon Parallel/Multiprocessor Ray Tracer Version...

CHAPTER

FORTYTWO

INTERFACE TO TIDES

This module contains tools to write the .c files needed for TIDES [TIDES].

Tides is an integration engine based on the Taylor method. It is implemented as a c library. The user must translate its initial value problem (IVP) into a pair of .c files that will then be compiled and linked against the TIDES library. The resulting binary will produce the desired output. The tools in this module can be used to automate the generation of these files from the symbolic expression of the differential equation.

AUTHORS:

- Miguel Marco (06-2014) Implementation of tides solver
- Marcos Rodriguez (06-2014) Implementation of tides solver
- Alberto Abad (06-2014) tides solver
- Roberto Barrio (06-2014) tides solver

REFERENCES:

- [ABBR2012]
- [TIDES]

Generate the needed files for the min_tides library.

INPUT:

- •integrator the name of the integrator file.
- •driver the name of the driver file.
- •f the function that determines the differential equation.
- •ics a list or tuple with the initial conditions.
- •initial the initial time for the integration.
- •final the final time for the integration.
- •delta the step of the output.

- •tolrel the relative tolerance.
- •tolabs the absolute tolerance.
- •output the name of the file that the compiled integrator will write to

This function creates two files, integrator and driver, that can be used later with the min_tides library [TIDES].

TESTS:

```
sage: from sage.interfaces.tides import genfiles_mintides
sage: import os
sage: import shutil
sage: from sage.misc.temporary_file import tmp_dir
sage: tempdir = tmp_dir()
sage: intfile = os.path.join(tempdir, 'integrator.c')
sage: drfile = os.path.join(tempdir ,'driver.c')
sage: var('t,x,y,X,Y')
(t, x, y, X, Y)
sage: f(t,x,y,X,Y) = [X, Y, -x/(x^2+y^2)^(3/2), -y/(x^2+y^2)^(3/2)]
sage: genfiles_mintides(intfile, drfile, f, [1,0, 0, 0.2], 0, 10, 0.1, output =

→ 'out.')

sage: fileint = open(intfile)
sage: l = fileint.readlines()
sage: fileint.close()
sage: 1[5]
    #include "minc_tides.h"\n'
sage: 1[15]
    double XX[TT+1][MO+1]; \n'
sage: 1[25]
'\n'
sage: 1[35]
'\t XX[1][i+1] = XX[3][i] / (i+1.0); \n'
sage: filedr = open(drfile)
sage: l = filedr.readlines()
sage: filedr.close()
sage: 1[6]
    #include "minc_tides.h"\n'
sage: 1[15]
    double tolrel, tolabs, tini, tend, dt; \n'
sage: 1[25]
'\ttolrel = 9.999999999999998e-17;\n'
sage: shutil.rmtree(tempdir)
```

Check that ticket trac ticket #17179 is fixed (handle expressions like

```
sage: l = fileint.readlines()
sage: fileint.close()
sage: l[30]
'\t\tXX[8][i] = pow_mc_c(XX[7],-1.500000000000000,XX[8], i);\n'
sage: filedr = open(drfile)
sage: l = filedr.readlines()
sage: filedr.close()
sage: filedr.close()
sage: l[18]
' \tv[0] = 3.1415926535897931; \n'
sage: shutil.rmtree(tempdir)
```

sage.interfaces.tides.genfiles_mpfr (integrator, driver, f, ics, initial, final, delta, parameters=None, parameter_values=None, dig=20, tolrel=1e-16, tolabs=1e-16, output='')

Generate the needed files for the mpfr module of the tides library.

INPUT:

- •integrator the name of the integrator file.
- •driver the name of the driver file.
- •f the function that determines the differential equation.
- •ics a list or tuple with the initial conditions.
- •initial the initial time for the integration.
- •final the final time for the integration.
- •delta the step of the output.

•parameters – the variables inside the function that should be treated as parameters.

*parameter_values - the values of the parameters for the particular initial value problem.

- •dig the number of digits of precision that will be used in the integration
- •tolrel the relative tolerance.
- •tolabs the absolute tolerance.
- •output the name of the file that the compiled integrator will write to

This function creates two files, integrator and driver, that can be used later with the tides library ([TIDES]).

TESTS:

```
sage: fileint.close()
sage: 1[5]
   #include "mp_tides.h"\n'
sage: 1[15]
'\tstatic int PARAMETERS = 0;\n'
sage: 1[25]
'\t\tmpfrts_var_t(itd, link[5], var[3], i);\n'
sage: 1[30]
'\t\tmpfrts_pow_t_c(itd, link[2], "-1.
sage: 1[35]
'\n'
sage: 1[36]
   } \n'
sage: 1[37]
  write_mp_solution(); \n'
sage: filedr = open(drfile)
sage: l = filedr.readlines()
sage: filedr.close()
sage: 1[6]
   #include "mpfr.h"\n'
sage: 1[16]
   int nfun = 0; \n'
sage: 1[26]
\hookrightarrow10, TIDES_RND); \n'
sage: 1[30]
'\tmpfr_init2(tolabs, TIDES_PREC); \n'
sage: 1[34]
'\tmpfr_init2(tini, TIDES_PREC); \n'
sage: 1[40]
'\tmp_tides_delta(function_iteration, NULL, nvar, npar, nfun, v, p, tini, dt,,,
→nipt, tolrel, tolabs, NULL, fd);\n'
sage: shutil.rmtree(tempdir)
```

Check that ticket trac ticket #17179 is fixed (handle expressions like pi):

```
sage: from sage.interfaces.tides import genfiles_mpfr
sage: import os
sage: import shutil
sage: from sage.misc.temporary_file import tmp_dir
sage: tempdir = tmp_dir()
sage: intfile = os.path.join(tempdir, 'integrator.c')
sage: drfile = os.path.join(tempdir ,'driver.c')
sage: var('t,x,y,X,Y')
(t, x, y, X, Y)
sage: f(t,x,y,X,Y) = [X, Y, -x/(x^2+y^2)^(3/2), -y/(x^2+y^2)^(3/2)]
sage: genfiles_mpfr(intfile, drfile, f, [pi, 0, 0, 0.2], 0, 10, 0.1, output = 'out
\rightarrow', dig = 50)
sage: fileint = open(intfile)
sage: l = fileint.readlines()
sage: fileint.close()
sage: 1[30]
'\t\tmpfrts_pow_t_c(itd, link[2], "-1.
sage: filedr = open(drfile)
sage: l = filedr.readlines()
```

```
sage: filedr.close()
sage: 1[24]
'\tmpfr_set_str(v[0], "3.141592653589793238462643383279502884197169399375101",
→10, TIDES_RND);\n'
sage: shutil.rmtree(tempdir)
```

sage.interfaces.tides.remove_constants (l1,l2)

Given two lists, remove the entries in the first that are real constants, and also the corresponding elements in the second one.

```
sage: from sage.interfaces.tides import subexpressions_list, remove_constants sage: f(a)=[1+\cos(7)*a] \text{ sage: } 11,12=\text{subexpressions\_list}(f) \text{ sage: } 11,12 ([\sin(7),\cos(7),a*\cos(7),a*\cos(7)+1],[('\sin',7),('\cos',7),('mul',\cos(7),a),('add',1,a*\cos(7))]) \text{ sage: } remove\_constants(11,12) \text{ sage: } 11,12 ([a*\cos(7),a*\cos(7)+1],[('mul',\cos(7),a),('add',1,a*\cos(7))])
```

```
sage.interfaces.tides.remove_repeated (l1,l2)
```

Given two lists, remove the repeated elements in 11, and the elements in 12 that are on the same position. positions.

EXAMPLES:

```
sage: from sage.interfaces.tides import (subexpressions_list, remove_repeated)
sage: f(a) = [1 + a^2, arcsin(a)]
sage: 11, 12 = subexpressions_list(f)
sage: 11, 12
([a^2, a^2 + 1, a^2, -a^2, -a^2 + 1, sqrt(-a^2 + 1), arcsin(a)],
[('mul', a, a),
('add', 1, a^2),
('mul', a, a),
('mul', -1, a^2),
('add', 1, -a^2),
('pow', -a^2 + 1, 0.5),
('asin', a)])
sage: remove_repeated(11, 12)
sage: 11, 12
([a^2, a^2 + 1, -a^2, -a^2 + 1, sqrt(-a^2 + 1), arcsin(a)],
[('mul', a, a),
('add', 1, a^2),
('mul', -1, a^2),
('add', 1, -a^2),
('pow', -a^2 + 1, 0.5),
('asin', a)])
```

sage.interfaces.tides. subexpressions_list (f, pars=None)

Construct the lists with the intermediate steps on the evaluation of the function.

INPUT:

- •f a symbolic function of several components.
- •pars a list of the parameters that appear in the function this should be the symbolic constants that appear in f but are not arguments.

OUTPUT:

- •a list of the intermediate subexpressions that appear in the evaluation of f.
- •a list with the operations used to construct each of the subexpressions. each element of this list is a tuple, formed by a string describing the operation made, and the operands.

For the trigonometric functions, some extra expressions will be added. These extra expressions will be used later to compute their derivatives.

EXAMPLES:

```
sage: from sage.interfaces.tides import subexpressions_list
sage: var('x,y')
(x, y)
sage: f(x,y) = [x^2+y, cos(x)/log(y)]
sage: subexpressions_list(f)
([x^2, x^2 + y, sin(x), cos(x), log(y), cos(x)/log(y)],
[('mul', x, x),
('add', y, x^2),
('sin', x),
('cos', x),
('log', y),
('div', log(y), cos(x))])
```

```
sage: f(a) = [cos(a), arctan(a)]
sage: from sage.interfaces.tides import subexpressions_list
sage: subexpressions_list(f)
([sin(a), cos(a), a^2, a^2 + 1, arctan(a)],
[('sin', a), ('cos', a), ('mul', a, a), ('add', 1, a^2), ('atan', a)])
```

```
sage: from sage.interfaces.tides import subexpressions_list
sage: var('s,b,r')
(s, b, r)
sage: f(t,x,y,z) = [s*(y-x),x*(r-z)-y,x*y-b*z]
sage: subexpressions_list(f,[s,b,r])
([-y,
x - y,
s*(x - y),
-s*(x - y),
-z,
r - z
(r - z) *x,
-y,
(r - z) *x - y
х*У,
b*z,
-b*z,
x*y - b*z],
[('mul', -1, y),
('add', -y, x),
('mul', x - y, s),
('mul', -1, s*(x - y)),
('mul', -1, z),
('add', -z, r),
('mul', x, r - z),
('mul', -1, y),
('add', -y, (r - z)*x),
('mul', y, x),
('mul', z, b),
('mul', -1, b*z),
('add', -b*z, x*y)])
```

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

```
sage: f(x,y) = [exp(x^2+sin(y))]
sage: from sage.interfaces.tides import *
sage: subexpressions_list(f)
([x^2, sin(y), cos(y), x^2 + sin(y), e^(x^2 + sin(y))],
[('mul', x, x),
('sin', y),
('cos', y),
('add', sin(y), x^2),
('exp', x^2 + sin(y))])
```

CHAPTER

FORTYTHREE

INTERFACE TO THE SAGE CLEANER

Triva Note: For the name "sage-cleaner", think of the "The Cleaner" from Pulp Fiction: http://www.frankjankowski. de/quiz/illus/keitel.jpg

```
sage.interfaces.cleaner. cleaner ( pid, cmd='')
```

Write a line to the ${\tt spawned_processes}$ file with the given ${\tt pid}$ and ${\tt cmd}$.

sage.interfaces.cleaner. ${\bf start_cleaner}$ ()

Start sage-cleaner in a new process group.

Sage Reference Manual: Interpreter Interfaces, Release 7.5		

CHAPTER

FORTYFOUR

QUITTING INTERFACES

sage.interfaces.quit. expect_quitall (verbose=False)
EXAMPLES:

```
sage.interfaces.quit.invalidate_all ()
```

Invalidate all of the expect interfaces.

This is used, e.g., by the fork-based @parallel decorator.

EXAMPLES:

However the maxima and gp sessions should still work out, though with their state reset:

```
sage: a = maxima(2); b = gp(3) sage: a, b (2, 3)
sage.interfaces.quit. is_running ( pid)
   Return True if and only if there is a process with id pid running.
sage.interfaces.quit. kill_spawned_jobs ( verbose=False)
INPLIT:
```

•verbose - bool (default: False); if True, display a message each time a process is sent a kill signal

EXAMPLES:

```
sage: gp.eval('a=10')
'10'
sage: sage.interfaces.quit.kill_spawned_jobs(verbose=False)
sage: sage.interfaces.quit.expect_quitall()
sage: gp.eval('a=10')
'10'
sage: sage.interfaces.quit.kill_spawned_jobs(verbose=True)
Killing spawned job ...
```

After doing the above, we do the following to avoid confusion in other doctests:

```
sage: sage.interfaces.quit.expect_quitall()
```

AN INTERFACE TO READ DATA FILES

```
sage.interfaces.read_data. read_data ( f, t)
Read data from file 'f' and class 't' (one element per line), and returns a list of elements.

INPUT:

•'f' - a file name

•'t' - a class (objects will be coerced to that class)

OUTPUT:
```

EXAMPLES:

a list of elements of class 't'.

```
sage: indata = tmp_filename()
sage: f = open(indata, "w")
sage: f.write("17\n42\n")
sage: f.close()
sage: l = read_data(indata, ZZ); l
[17, 42]
sage: f = open(indata, "w")
sage: f.write("1.234\n5.678\n")
sage: f.close()
sage: l = read_data(indata, RealField(17)); l
[1.234, 5.678]
```

Sage Reference Manual: Interpreter Interfaces, Release 7.5		

CHAPTER

FORTYSIX

INDICES AND TABLES

- Index
- Module Index
- Search Page

```
sage.interfaces.axiom, 15
sage.interfaces.cleaner, 329
sage.interfaces.ecm, 21
sage.interfaces.expect,9
sage.interfaces.four_ti_2,27
sage.interfaces.fricas, 33
sage.interfaces.frobby, 45
sage.interfaces.gap, 49
sage.interfaces.gap3,61
sage.interfaces.gfan,71
sage.interfaces.giac, 73
sage.interfaces.gnuplot,81
sage.interfaces.gp, 83
sage.interfaces.interface, 3
sage.interfaces.jmoldata,93
sage.interfaces.kash,95
sage.interfaces.lie, 103
sage.interfaces.lisp, 111
sage.interfaces.macaulay2, 115
sage.interfaces.magma, 125
sage.interfaces.magma_free, 145
sage.interfaces.maple, 147
sage.interfaces.mathematica, 155
sage.interfaces.matlab, 163
sage.interfaces.maxima, 167
sage.interfaces.maxima abstract, 177
sage.interfaces.maxima_lib, 195
sage.interfaces.mupad, 211
sage.interfaces.mwrank, 215
sage.interfaces.octave, 219
sage.interfaces.phc, 225
sage.interfaces.povray, 233
sage.interfaces.psage, 235
sage.interfaces.qepcad, 237
sage.interfaces.gsieve, 265
sage.interfaces.quit, 331
```

Sage Reference Manual: Interpreter Interfaces, Release 7.5

```
sage.interfaces.r, 269
sage.interfaces.read_data, 333
sage.interfaces.rubik, 283
sage.interfaces.sage0, 285
sage.interfaces.sagespawn, 13
sage.interfaces.scilab, 289
sage.interfaces.singular, 295
sage.interfaces.tachyon, 317
sage.interfaces.tides, 321
```

338 Python Module Index

Symbols

```
call () (sage.interfaces.tachyon.TachyonRT method), 317
_sage_() (sage.interfaces.fricas.FriCASElement method), 39
Α
A() (sage.interfaces.qepcad_qepcad_formula_factory method), 256
alexander dual() (sage.interfaces.frobby.Frobby method), 45
all_but_finitely_many() (sage.interfaces.qepcad_qepcad_formula_factory method), 258
and_() (sage.interfaces.qepcad_formula_factory method), 258
answer() (sage.interfaces.qepcad.Qepcad method), 246
arguments() (sage.interfaces.maxima_abstract.MaximaAbstractElementFunction method), 191
as_type() (sage.interfaces.axiom.PanAxiomElement method), 18
AsciiArtString (class in sage.interfaces.interface), 3
assign names() (sage.interfaces.magma.MagmaElement method), 137
AssignNames() (sage.interfaces.magma.MagmaElement method), 137
associated_primes() (sage.interfaces.frobby.Frobby method), 46
assume() (sage.interfaces.qepcad.Qepcad method), 246
atomic() (sage.interfaces.qepcad.qepcad formula factory method), 259
Attach() (sage.interfaces.magma.Magma method), 129
attach() (sage.interfaces.magma.Magma method), 130
attach spec() (sage.interfaces.magma.Magma method), 130
AttachSpec() (sage.interfaces.magma.Magma method), 129
attrib() (sage.interfaces.singular.SingularElement method), 307
attribute() (sage.interfaces.interface.InterfaceElement method), 5
available packages() (sage.interfaces.r.R method), 273
Axiom (class in sage.interfaces.axiom), 17
axiom_console() (in module sage.interfaces.axiom), 19
AxiomElement (class in sage.interfaces.axiom), 17
AxiomExpectFunction (class in sage.interfaces.axiom), 17
AxiomFunctionElement (class in sage.interfaces.axiom), 17
B
bar_call() (sage.interfaces.magma.Magma method), 131
blackbox() (sage.interfaces.phc.PHC method), 225
bool() (sage.interfaces.fricas.FriCASElement method), 42
bool() (sage.interfaces.gap.GapElement_generic method), 53
bool() (sage.interfaces.gp.GpElement method), 90
```

```
bool() (sage.interfaces.interface.InterfaceElement method), 5
bool() (sage.interfaces.lisp.LispElement method), 113
bool() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 185
C
C() (sage.interfaces.gepcad.gepcad formula factory method), 256
call() (sage.interfaces.four ti 2.FourTi2 method), 27
call() (sage.interfaces.interface.Interface method), 3
call() (sage.interfaces.r.R method), 273
cell() (sage.interfaces.gepcad.Qepcad method), 246
chdir() (sage.interfaces.magma.Magma method), 131
chdir() (sage.interfaces.mathematica.Mathematica method), 160
chdir() (sage.interfaces.matlab.Matlab method), 165
chdir() (sage.interfaces.maxima abstract.MaximaAbstract method), 177
chdir() (sage.interfaces.r.R method), 273
circuits() (sage.interfaces.four_ti_2.FourTi2 method), 27
classified solution dicts() (sage.interfaces.phc.PHC Object method), 228
clean output() (in module sage.interfaces.mathematica), 162
cleaner() (in module sage.interfaces.cleaner), 329
clear() (sage.interfaces.giac.Giac method), 76
clear() (sage.interfaces.interface.Interface method), 3
clear() (sage.interfaces.magma.Magma method), 131
clear() (sage.interfaces.maple.Maple method), 150
clear() (sage.interfaces.maxima.Maxima method), 174
clear() (sage.interfaces.maxima lib.MaximaLib method), 196
clear() (sage.interfaces.octave.Octave method), 221
clear() (sage.interfaces.sage0.Sage method), 286
clear() (sage.interfaces.singular.Singular method), 300
clear prompts() (sage.interfaces.expect.Expect method), 9
close() (sage.interfaces.sagespawn.SagePtyProcess method), 13
cls() (sage.interfaces.macaulay2.Macaulay2Element method), 119
comma() (sage.interfaces.axiom.PanAxiomElement method), 18
comma() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 186
command() (sage.interfaces.expect.Expect method), 9
completions() (sage.interfaces.giac.Giac method), 76
completions() (sage.interfaces.maple.Maple method), 150
completions() (sage.interfaces.maxima abstract.MaximaAbstract method), 178
completions() (sage.interfaces.mupad.Mupad method), 212
completions() (sage.interfaces.r.R method), 274
connected subset() (sage.interfaces.qepcad.qepcad formula factory method), 259
console() (in module sage.interfaces.expect), 12
console() (sage.interfaces.axiom.Axiom method), 17
console() (sage.interfaces.fricas.FriCAS method), 36
console() (sage.interfaces.gap.Gap method), 51
console() (sage.interfaces.gap3.Gap3 method), 66
console() (sage.interfaces.giac.Giac method), 76
console() (sage.interfaces.gnuplot.Gnuplot method), 81
console() (sage.interfaces.gp.Gp method), 85
console() (sage.interfaces.interface.Interface method), 3
console() (sage.interfaces.kash.Kash method), 101
```

```
console() (sage.interfaces.lie.LiE method), 107
console() (sage.interfaces.lisp.Lisp method), 111
console() (sage.interfaces.macaulay2.Macaulay2 method), 116
console() (sage.interfaces.magma.Magma method), 132
console() (sage.interfaces.maple.Maple method), 150
console() (sage.interfaces.mathematica.Mathematica method), 160
console() (sage.interfaces.matlab.Matlab method), 165
console() (sage.interfaces.maxima abstract.MaximaAbstract method), 178
console() (sage.interfaces.mupad.Mupad method), 212
console() (sage.interfaces.mwrank.Mwrank class method), 215
console() (sage.interfaces.octave.Octave method), 221
console() (sage.interfaces.r.R method), 274
console() (sage.interfaces.sage0.Sage method), 286
console() (sage.interfaces.scilab.Scilab method), 292
console() (sage.interfaces.singular.Singular method), 300
convert_r_list() (sage.interfaces.r.R method), 274
cputime() (sage.interfaces.gap.Gap method), 51
cputime() (sage.interfaces.gap3.Gap3 method), 67
cputime() (sage.interfaces.giac.Giac method), 77
cputime() (sage.interfaces.gp.Gp method), 85
cputime() (sage.interfaces.interface.Interface method), 3
cputime() (sage.interfaces.macaulay2.Macaulay2 method), 116
cputime() (sage.interfaces.magma.Magma method), 132
cputime() (sage.interfaces.maple.Maple method), 150
cputime() (sage.interfaces.maxima_abstract.MaximaAbstract method), 178
cputime() (sage.interfaces.mupad.Mupad method), 212
cputime() (sage.interfaces.qsieve.qsieve_nonblock method), 266
cputime() (sage.interfaces.sage0.Sage method), 287
cputime() (sage.interfaces.singular.Singular method), 300
CubexSolver (class in sage.interfaces.rubik), 283
current ring() (sage.interfaces.singular.Singular method), 301
current ring name() (sage.interfaces.singular.Singular method), 301
D
data to list() (in module sage.interfaces.qsieve), 265
de solve() (sage.interfaces.maxima abstract.MaximaAbstract method), 179
de solve laplace() (sage.interfaces.maxima abstract.MaximaAbstract method), 179
de system plot() (sage.interfaces.octave.Octave method), 222
definition() (sage.interfaces.maxima_abstract.MaximaAbstractElementFunction method), 191
demo() (sage.interfaces.maxima abstract.MaximaAbstract method), 179
derivative() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 186
describe() (sage.interfaces.maxima_abstract.MaximaAbstract method), 180
detach() (sage.interfaces.expect.Expect method), 9
diff() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 186
DikSolver (class in sage.interfaces.rubik), 283
dimension() (sage.interfaces.frobby.Frobby method), 46
directory() (sage.interfaces.four_ti_2.FourTi2 method), 28
display2d() (sage.interfaces.maxima.MaximaElement method), 175
display2d() (sage.interfaces.maxima lib.MaximaLibElement method), 202
done() (sage.interfaces.qsieve_nonblock method), 266
```

```
dot() (sage.interfaces.macaulay2.Macaulay2Element method), 119
dot() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 187
dot product() (sage.interfaces.r.RElement method), 279
dummy integrate() (in module sage.interfaces.maxima lib), 203
F
E() (sage.interfaces.qepcad_qepcad_formula_factory method), 257
ecl() (sage.interfaces.maxima lib.MaximaLibElement method), 203
ECM (class in sage.interfaces.ecm), 21
eval() (sage.interfaces.expect.Expect method), 10
eval() (sage.interfaces.fricas.FriCAS method), 36
eval() (sage.interfaces.gap.Gap generic method), 54
eval() (sage.interfaces.giac.Giac method), 77
eval() (sage.interfaces.interface.Interface method), 3
eval() (sage.interfaces.kash.Kash method), 101
eval() (sage.interfaces.lie.LiE method), 107
eval() (sage.interfaces.lisp.Lisp method), 112
eval() (sage.interfaces.macaulay2.Macaulay2 method), 116
eval() (sage.interfaces.magma.Magma method), 133
eval() (sage.interfaces.magma.MagmaElement method), 138
eval() (sage.interfaces.magma_free.MagmaFree method), 145
eval() (sage.interfaces.mathematica.Mathematica method), 160
eval() (sage.interfaces.maxima lib.MaximaLib method), 196
eval() (sage.interfaces.mupad.Mupad method), 213
eval() (sage.interfaces.mwrank.Mwrank_class method), 215
eval() (sage.interfaces.psage.PSage method), 235
eval() (sage.interfaces.r.R method), 275
eval() (sage.interfaces.sage0.Sage method), 287
eval() (sage.interfaces.scilab.Scilab method), 292
eval() (sage.interfaces.singular.Singular method), 301
evaluate() (sage.interfaces.magma.MagmaElement method), 138
exactly_k() (sage.interfaces.qepcad_formula_factory method), 260
example() (sage.interfaces.maxima_abstract.MaximaAbstract method), 180
execute() (sage.interfaces.interface.Interface method), 3
exists() (sage.interfaces.qepcad.qepcad formula factory method), 260
Expect (class in sage.interfaces.expect), 9
expect() (sage.interfaces.expect.Expect method), 10
expect() (sage.interfaces.giac.Giac method), 77
expect() (sage.interfaces.maple.Maple method), 151
expect() (sage.interfaces.mupad.Mupad method), 213
expect_peek() (sage.interfaces.sagespawn.SageSpawn method), 14
expect_quitall() (in module sage.interfaces.quit), 331
expect_upto() (sage.interfaces.sagespawn.SageSpawn method), 14
ExpectElement (class in sage.interfaces.expect), 11
ExpectFunction (class in sage.interfaces.expect), 12
export image() (sage.interfaces.jmoldata.JmolData method), 93
extcode dir() (in module sage.interfaces.magma), 142
external string() (sage.interfaces.macaulay2.Macaulay2Element method), 119
```

F

```
F() (sage.interfaces.qepcad_qepcad_formula_factory method), 257
factor() (sage.interfaces.ecm.ECM method), 22
final_stats() (sage.interfaces.qepcad.Qepcad method), 247
find factor() (sage.interfaces.ecm.ECM method), 23
flush() (sage.interfaces.magma.MagmaGBLogPrettyPrinter method), 142
flush() (sage.interfaces.singular.SingularGBLogPrettyPrinter method), 312
forall() (sage.interfaces.qepcad_qepcad_formula_factory method), 260
format_cube() (sage.interfaces.rubik.CubexSolver method), 283
format cube() (sage.interfaces.rubik.DikSolver method), 283
format cube() (sage.interfaces.rubik.OptimalSolver method), 283
formula() (sage.interfaces.gepcad.gepcad formula factory method), 261
FourTi2 (class in sage.interfaces.four_ti_2), 27
FriCAS (class in sage.interfaces.fricas), 36
fricas console() (in module sage.interfaces.fricas), 42
FriCASElement (class in sage.interfaces.fricas), 39
FriCASExpectFunction (class in sage.interfaces.fricas), 42
FriCASFunctionElement (class in sage.interfaces.fricas), 42
Frobby (class in sage.interfaces.frobby), 45
function() (sage.interfaces.maxima_abstract.MaximaAbstract method), 180
function call() (sage.interfaces.gap.Gap generic method), 54
function call() (sage.interfaces.interface.Interface method), 3
function call() (sage.interfaces.lie.LiE method), 108
function_call() (sage.interfaces.lisp.Lisp method), 112
function call() (sage.interfaces.magma.Magma method), 133
function call() (sage.interfaces.r.R method), 275
FunctionElement (class in sage.interfaces.expect), 12
G
G() (sage.interfaces.qepcad_qepcad_formula_factory method), 257
Gap (class in sage.interfaces.gap), 51
Gap3 (class in sage.interfaces.gap3), 66
gap3 console() (in module sage.interfaces.gap3), 68
gap3_version() (in module sage.interfaces.gap3), 68
GAP3Element (class in sage.interfaces.gap3), 64
GAP3Record (class in sage.interfaces.gap3), 65
gap_command() (in module sage.interfaces.gap), 56
gap console() (in module sage.interfaces.gap), 56
Gap generic (class in sage.interfaces.gap), 53
gap_reset_workspace() (in module sage.interfaces.gap), 57
GapElement (class in sage.interfaces.gap), 53
GapElement generic (class in sage.interfaces.gap), 53
GapFunction (class in sage.interfaces.gap), 53
GapFunctionElement (class in sage.interfaces.gap), 53
gc_disabled (class in sage.interfaces.expect), 12
gen() (sage.interfaces.fricas.FriCASElement method), 42
gen() (sage.interfaces.interface.InterfaceElement method), 5
gen() (sage.interfaces.magma.MagmaElement method), 138
gen_names() (sage.interfaces.magma.MagmaElement method), 139
generate docstring dictionary() (in module sage.interfaces.singular), 313
```

```
genfiles mintides() (in module sage.interfaces.tides), 321
genfiles_mpfr() (in module sage.interfaces.tides), 323
gens() (sage.interfaces.magma.MagmaElement method), 139
get() (sage.interfaces.axiom.PanAxiom method), 18
get() (sage.interfaces.fricas.FriCAS method), 37
get() (sage.interfaces.gap.Gap method), 52
get() (sage.interfaces.giac.Giac method), 77
get() (sage.interfaces.gp.Gp method), 86
get() (sage.interfaces.interface.Interface method), 3
get() (sage.interfaces.kash.Kash method), 101
get() (sage.interfaces.lie.LiE method), 108
get() (sage.interfaces.lisp.Lisp method), 112
get() (sage.interfaces.macaulay2.Macaulay2 method), 117
get() (sage.interfaces.magma.Magma method), 134
get() (sage.interfaces.maple.Maple method), 151
get() (sage.interfaces.mathematica.Mathematica method), 160
get() (sage.interfaces.matlab.Matlab method), 165
get() (sage.interfaces.maxima.Maxima method), 174
get() (sage.interfaces.maxima lib.MaximaLib method), 197
get() (sage.interfaces.mupad.Mupad method), 213
get() (sage.interfaces.octave.Octave method), 222
get() (sage.interfaces.psage.PSage method), 235
get() (sage.interfaces.r.R method), 275
get() (sage.interfaces.sage0.Sage method), 287
get() (sage.interfaces.scilab.Scilab method), 292
get() (sage.interfaces.singular.Singular method), 302
get_boolean() (sage.interfaces.fricas.FriCAS method), 37
get_classified_solution_dicts() (in module sage.interfaces.phc), 230
get default() (sage.interfaces.gp.Gp method), 86
get docstring() (in module sage.interfaces.singular), 313
get_gap_memory_pool_size() (in module sage.interfaces.gap), 58
get_integer() (sage.interfaces.fricas.FriCAS method), 37
get last params() (sage.interfaces.ecm.ECM method), 24
get_magma_attribute() (sage.interfaces.magma.MagmaElement method), 139
get_precision() (sage.interfaces.gp.Gp method), 86
get_real_precision() (sage.interfaces.gp.Gp method), 86
get_record_element() (sage.interfaces.gap.Gap_generic method), 55
get seed() (sage.interfaces.interface.Interface method), 3
get_series_precision() (sage.interfaces.gp.Gp method), 86
get_solution_dicts() (in module sage.interfaces.phc), 230
get string() (sage.interfaces.fricas.FriCAS method), 38
get_unparsed_InputForm() (sage.interfaces.fricas.FriCAS method), 38
get_using_file() (sage.interfaces.interface.Interface method), 4
get using file() (sage.interfaces.interface.InterfaceElement method), 5
get_using_file() (sage.interfaces.lie.LiE method), 108
get variable list() (in module sage.interfaces.phc), 230
get_verbose() (sage.interfaces.magma.Magma method), 134
GetVerbose() (sage.interfaces.magma.Magma method), 129
Gfan (class in sage.interfaces.gfan), 71
gfq_gap_to_sage() (in module sage.interfaces.gap), 58
```

```
Giac (class in sage.interfaces.giac), 75
giac_console() (in module sage.interfaces.giac), 80
GiacElement (class in sage.interfaces.giac), 78
GiacFunction (class in sage.interfaces.giac), 79
GiacFunctionElement (class in sage.interfaces.giac), 79
Gnuplot (class in sage.interfaces.gnuplot), 81
gnuplot() (sage.interfaces.gnuplot.Gnuplot method), 81
gnuplot console() (in module sage.interfaces.gnuplot), 82
Gp (class in sage.interfaces.gp), 85
gp_console() (in module sage.interfaces.gp), 90
gp version() (in module sage.interfaces.gp), 90
GpElement (class in sage.interfaces.gp), 89
GpFunction (class in sage.interfaces.gp), 90
GpFunctionElement (class in sage.interfaces.gp), 90
graver() (sage.interfaces.four ti 2.FourTi2 method), 28
groebner() (sage.interfaces.four_ti_2.FourTi2 method), 28
Н
hasattr() (sage.interfaces.interface.InterfaceElement method), 6
help() (sage.interfaces.gap.Gap method), 52
help() (sage.interfaces.gap3.Gap3 method), 67
help() (sage.interfaces.giac.Giac method), 78
help() (sage.interfaces.gp.Gp method), 87
help() (sage.interfaces.interface.Interface method), 4
help() (sage.interfaces.interface.InterfaceFunctionElement method), 7
help() (sage.interfaces.kash.Kash method), 101
help() (sage.interfaces.lie.LiE method), 108
help() (sage.interfaces.lisp.Lisp method), 112
help() (sage.interfaces.macaulay2.Macaulay2 method), 117
help() (sage.interfaces.magma.Magma method), 134
help() (sage.interfaces.maple.Maple method), 151
help() (sage.interfaces.mathematica.Mathematica method), 160
help() (sage.interfaces.maxima_abstract.MaximaAbstract method), 181
help() (sage.interfaces.r.R method), 275
help() (sage.interfaces.tachyon.TachyonRT method), 318
help_search() (sage.interfaces.kash.Kash method), 102
HelpExpression (class in sage.interfaces.r), 273
hilbert() (sage.interfaces.four ti 2.FourTi2 method), 28
hilbert() (sage.interfaces.frobby.Frobby method), 46
ideal() (sage.interfaces.macaulay2.Macaulay2 method), 117
ideal() (sage.interfaces.magma.Magma method), 135
ideal() (sage.interfaces.magma.MagmaElement method), 140
ideal() (sage.interfaces.singular.Singular method), 302
iff() (sage.interfaces.qepcad_formula_factory method), 261
imag() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 187
implies() (sage.interfaces.qepcad_qepcad_formula_factory method), 261
index() (sage.interfaces.qepcad.QepcadCell method), 249
infinitely many() (sage.interfaces.qepcad.qepcad formula factory method), 262
```

```
install packages() (sage.interfaces.r.R method), 275
integral() (sage.interfaces.giac.GiacElement method), 78
integral() (sage.interfaces.maxima abstract.MaximaAbstractElement method), 187
integral() (sage.interfaces.maxima abstract.MaximaAbstractElementFunction method), 192
integrate() (sage.interfaces.giac.GiacElement method), 79
integrate() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 188
integrate() (sage.interfaces.maxima abstract.MaximaAbstractElementFunction method), 192
interact() (sage.interfaces.ecm.ECM method), 24
interact() (sage.interfaces.gnuplot.Gnuplot method), 81
interact() (sage.interfaces.interface.Interface method), 4
Interface (class in sage.interfaces.interface), 3
InterfaceElement (class in sage.interfaces.interface), 5
InterfaceFunction (class in sage.interfaces.interface), 7
InterfaceFunctionElement (class in sage.interfaces.interface), 7
interrupt() (sage.interfaces.expect.Expect method), 10
interrupt() (sage.interfaces.gap.Gap_generic method), 55
intmod_gap_to_sage() (in module sage.interfaces.gap), 58
invalidate all() (in module sage.interfaces.quit), 331
irreducible decomposition() (sage.interfaces.frobby.Frobby method), 47
is_AxiomElement() (in module sage.interfaces.axiom), 20
is_ExpectElement() (in module sage.interfaces.expect), 12
is FriCASElement() (in module sage.interfaces.fricas), 43
is_GapElement() (in module sage.interfaces.gap), 59
is GpElement() (in module sage.interfaces.gp), 90
is_InterfaceElement() (in module sage.interfaces.interface), 7
is jvm available() (sage.interfaces.jmoldata.JmolData method), 94
is_KashElement() (in module sage.interfaces.kash), 102
is_LiEElement() (in module sage.interfaces.lie), 109
is LispElement() (in module sage.interfaces.lisp), 113
is local() (sage.interfaces.expect.Expect method), 10
is locked() (sage.interfaces.psage.PSage method), 236
is locked() (sage.interfaces.psage.PSageElement method), 236
is Macaulay2Element() (in module sage.interfaces.macaulay2), 122
is_MagmaElement() (in module sage.interfaces.magma), 142
is MaximaElement() (in module sage.interfaces.maxima), 176
is_MaximaLibElement() (in module sage.interfaces.maxima_lib), 204
is_RElement() (in module sage.interfaces.r), 280
is remote() (sage.interfaces.expect.Expect method), 10
is_running() (in module sage.interfaces.quit), 331
is running() (sage.interfaces.expect.Expect method), 10
is SingularElement() (in module sage.interfaces.singular), 313
is_string() (sage.interfaces.gap.GapElement_generic method), 53
is_string() (sage.interfaces.gp.GpElement method), 90
is string() (sage.interfaces.interface.InterfaceElement method), 6
is_string() (sage.interfaces.singular.SingularElement method), 308
J
```

JmolData (class in sage.interfaces.jmoldata), 93

Κ

```
Kash (class in sage.interfaces.kash), 101
kash console() (in module sage.interfaces.kash), 102
kash_version() (in module sage.interfaces.kash), 102
KashDocumentation (class in sage.interfaces.kash), 102
KashElement (class in sage.interfaces.kash), 102
kill() (sage.interfaces.gp.Gp method), 87
kill() (sage.interfaces.lisp.Lisp method), 112
kill_spawned_jobs() (in module sage.interfaces.quit), 331
level() (sage.interfaces.qepcad.QepcadCell method), 250
LIB() (sage.interfaces.singular.Singular method), 300
lib() (sage.interfaces.singular.Singular method), 303
library() (sage.interfaces.r.R method), 276
LiE (class in sage.interfaces.lie), 107
lie console() (in module sage.interfaces.lie), 109
lie_version() (in module sage.interfaces.lie), 109
LiEElement (class in sage.interfaces.lie), 109
LiEFunction (class in sage.interfaces.lie), 109
LiEFunctionElement (class in sage.interfaces.lie), 109
Lisp (class in sage.interfaces.lisp), 111
lisp() (sage.interfaces.maxima.Maxima method), 174
lisp() (sage.interfaces.maxima lib.MaximaLib method), 197
lisp console() (in module sage.interfaces.lisp), 113
LispElement (class in sage.interfaces.lisp), 113
LispFunction (class in sage.interfaces.lisp), 113
LispFunctionElement (class in sage.interfaces.lisp), 113
list() (sage.interfaces.qsieve_nonblock method), 266
list() (sage.interfaces.singular.Singular method), 303
list_attributes() (sage.interfaces.magma.MagmaElement method), 140
load() (sage.interfaces.magma.Magma method), 135
load() (sage.interfaces.maple.Maple method), 151
load() (sage.interfaces.singular.Singular method), 303
load package() (sage.interfaces.gap.Gap generic method), 56
log() (sage.interfaces.qsieve.qsieve nonblock method), 266
M
Macaulay2 (class in sage.interfaces.macaulay2), 116
macaulay2_console() (in module sage.interfaces.macaulay2), 122
Macaulay2Element (class in sage.interfaces.macaulay2), 119
Macaulay2Function (class in sage.interfaces.macaulay2), 122
Magma (class in sage.interfaces.magma), 128
magma_console() (in module sage.interfaces.magma), 142
magma free eval() (in module sage.interfaces.magma free), 145
magma gb standard options() (in module sage.interfaces.magma), 143
magma version() (in module sage.interfaces.magma), 143
MagmaElement (class in sage.interfaces.magma), 137
MagmaExpr (class in sage.interfaces.magma_free), 145
MagmaFree (class in sage.interfaces.magma free), 145
```

```
MagmaFunction (class in sage.interfaces.magma), 141
MagmaFunctionElement (class in sage.interfaces.magma), 142
MagmaGBDefaultContext (class in sage.interfaces.magma), 142
MagmaGBLogPrettyPrinter (class in sage.interfaces.magma), 142
make_cells() (sage.interfaces.qepcad.Qepcad method), 247
Maple (class in sage.interfaces.maple), 150
maple console() (in module sage.interfaces.maple), 153
MapleElement (class in sage.interfaces.maple), 152
MapleFunction (class in sage.interfaces.maple), 153
MapleFunctionElement (class in sage.interfaces.maple), 153
Mathematica (class in sage.interfaces.mathematica), 160
mathematica console() (in module sage.interfaces.mathematica), 162
MathematicaElement (class in sage.interfaces.mathematica), 161
MathematicaFunction (class in sage.interfaces.mathematica), 162
MathematicaFunctionElement (class in sage.interfaces.mathematica), 162
Matlab (class in sage.interfaces.matlab), 165
matlab_console() (in module sage.interfaces.matlab), 166
matlab version() (in module sage.interfaces.matlab), 166
MatlabElement (class in sage.interfaces.matlab), 166
matrix() (sage.interfaces.singular.Singular method), 303
max at to sage() (in module sage.interfaces.maxima lib), 204
max harmonic to sage() (in module sage.interfaces.maxima lib), 204
max_to_sr() (in module sage.interfaces.maxima_lib), 205
max_to_string() (in module sage.interfaces.maxima_lib), 205
Maxima (class in sage.interfaces.maxima), 174
maxima console() (in module sage.interfaces.maxima abstract), 192
maxima_version() (in module sage.interfaces.maxima_abstract), 192
MaximaAbstract (class in sage.interfaces.maxima_abstract), 177
MaximaAbstractElement (class in sage.interfaces.maxima abstract), 185
MaximaAbstractElementFunction (class in sage.interfaces.maxima abstract), 190
MaximaAbstractFunction (class in sage.interfaces.maxima abstract), 192
MaximaAbstractFunctionElement (class in sage.interfaces.maxima abstract), 192
MaximaElement (class in sage.interfaces.maxima), 175
MaximaElementFunction (class in sage.interfaces.maxima), 175
MaximaFunction (class in sage.interfaces.maxima), 175
MaximaFunctionElement (class in sage.interfaces.maxima), 175
MaximaLib (class in sage.interfaces.maxima_lib), 196
MaximaLibElement (class in sage.interfaces.maxima lib), 202
MaximaLibElementFunction (class in sage.interfaces.maxima_lib), 203
MaximaLibFunction (class in sage.interfaces.maxima lib), 203
MaximaLibFunctionElement (class in sage.interfaces.maxima lib), 203
mdiff to sage() (in module sage.interfaces.maxima lib), 205
methods() (sage.interfaces.magma.MagmaElement method), 140
minimize() (sage.interfaces.four ti 2.FourTi2 method), 29
mixed_volume() (sage.interfaces.phc.PHC method), 226
mlist to sage() (in module sage.interfaces.maxima lib), 206
mqapply_to_sage() (in module sage.interfaces.maxima_lib), 206
mrat_to_sage() (in module sage.interfaces.maxima_lib), 206
Mupad (class in sage.interfaces.mupad), 212
mupad_console() (in module sage.interfaces.mupad), 213
```

```
MupadElement (class in sage.interfaces.mupad), 213
MupadFunction (class in sage.interfaces.mupad), 213
MupadFunctionElement (class in sage.interfaces.mupad), 213
Mwrank() (in module sage.interfaces.mwrank), 215
Mwrank_class (class in sage.interfaces.mwrank), 215
mwrank_console() (in module sage.interfaces.mwrank), 216
Ν
N() (sage.interfaces.mathematica.MathematicaElement method), 161
n() (sage.interfaces.mathematica.MathematicaElement method), 161
n() (sage.interfaces.qsieve_nonblock method), 266
na() (sage.interfaces.r.R method), 276
name() (sage.interfaces.interface.Interface method), 4
name() (sage.interfaces.interface.InterfaceElement method), 6
new() (sage.interfaces.interface.Interface method), 4
new() (sage.interfaces.sage0.Sage method), 287
new from() (sage.interfaces.macaulay2.Macaulay2 method), 117
new_with_bits_prec() (sage.interfaces.gp.Gp method), 87
nintegral() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 188
not () (sage.interfaces.qepcad.qepcad formula factory method), 262
number_of_children() (sage.interfaces.qepcad.QepcadCell method), 250
numer() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 189
objgens() (sage.interfaces.magma.Magma method), 136
Octave (class in sage.interfaces.octave), 221
octave_console() (in module sage.interfaces.octave), 224
octave version() (in module sage.interfaces.octave), 224
OctaveElement (class in sage.interfaces.octave), 224
one_curve() (sage.interfaces.ecm.ECM method), 24
operations() (sage.interfaces.gap3.GAP3Record method), 65
OptimalSolver (class in sage.interfaces.rubik), 283
option() (sage.interfaces.singular.Singular method), 304
or_() (sage.interfaces.qepcad_qepcad_formula_factory method), 262
Р
PanAxiom (class in sage.interfaces.axiom), 18
PanAxiomElement (class in sage.interfaces.axiom), 18
PanAxiomExpectFunction (class in sage.interfaces.axiom), 19
PanAxiomFunctionElement (class in sage.interfaces.axiom), 19
parse_max_string() (in module sage.interfaces.maxima_lib), 207
partial fraction decomposition() (sage.interfaces.maxima abstract.MaximaAbstractElement method), 189
path() (sage.interfaces.expect.Expect method), 10
path_track() (sage.interfaces.phc.PHC method), 226
phase() (sage.interfaces.qepcad.Qepcad method), 247
PHC (class in sage.interfaces.phc), 225
PHC Object (class in sage.interfaces.phc), 228
pid() (sage.interfaces.expect.Expect method), 10
pid() (sage.interfaces.qsieve_qsieve_nonblock method), 266
plot() (sage.interfaces.gnuplot.Gnuplot method), 81
```

```
plot() (sage.interfaces.r.R method), 276
plot2d() (sage.interfaces.maxima_abstract.MaximaAbstract method), 181
plot2d parametric() (sage.interfaces.maxima abstract.MaximaAbstract method), 182
plot3d() (sage.interfaces.gnuplot.Gnuplot method), 81
plot3d() (sage.interfaces.maxima_abstract.MaximaAbstract method), 182
plot3d_parametric() (sage.interfaces.gnuplot.Gnuplot method), 81
plot3d parametric() (sage.interfaces.maxima abstract.MaximaAbstract method), 183
plot list() (sage.interfaces.maxima abstract.MaximaAbstract method), 183
plot_multilist() (sage.interfaces.maxima_abstract.MaximaAbstract method), 184
plot_paths_2d() (sage.interfaces.phc.PHC method), 227
png() (sage.interfaces.r.R method), 277
POVRay (class in sage.interfaces.povray), 233
ppi() (sage.interfaces.four_ti_2.FourTi2 method), 29
preparse() (sage.interfaces.sage0.Sage method), 287
PSage (class in sage.interfaces.psage), 235
PSageElement (class in sage.interfaces.psage), 236
pyobject_to_max() (in module sage.interfaces.maxima_lib), 207
Q
Qepcad (class in sage.interfaces.qepcad), 246
qepcad() (in module sage.interfaces.qepcad), 252
qepcad_banner() (in module sage.interfaces.qepcad), 255
qepcad_console() (in module sage.interfaces.qepcad), 256
Qepcad_expect (class in sage.interfaces.qepcad), 252
qepcad_formula_factory (class in sage.interfaces.qepcad), 256
qepcad version() (in module sage.interfaces.qepcad), 263
QepcadCell (class in sage.interfaces.qepcad), 249
QepcadFunction (class in sage.interfaces.qepcad), 251
qformula (class in sage.interfaces.qepcad), 263
qsieve() (in module sage.interfaces.qsieve), 265
gsieve block() (in module sage.interfaces.gsieve), 266
qsieve_nonblock (class in sage.interfaces.qsieve), 266
qsolve() (sage.interfaces.four_ti_2.FourTi2 method), 29
quantifier() (sage.interfaces.qepcad_formula_factory method), 262
quit() (sage.interfaces.expect.Expect method), 11
quit() (sage.interfaces.octave.Octave method), 222
quit() (sage.interfaces.qsieve.qsieve nonblock method), 266
quo() (sage.interfaces.magma.MagmaElement method), 140
R
R (class in sage.interfaces.r), 273
r console() (in module sage.interfaces.r), 281
r version() (in module sage.interfaces.r), 281
rand_seed() (sage.interfaces.interface.Interface method), 4
rays() (sage.interfaces.four ti 2.FourTi2 method), 29
read() (sage.interfaces.interface.Interface method), 4
read() (sage.interfaces.lie.LiE method), 108
read() (sage.interfaces.r.R method), 278
read_data() (in module sage.interfaces.read_data), 333
read_matrix() (sage.interfaces.four_ti_2.FourTi2 method), 29
```

```
ready() (sage.interfaces.rubik.OptimalSolver method), 283
real() (sage.interfaces.maxima_abstract.MaximaAbstractElement method), 190
recfields() (sage.interfaces.gap3.GAP3Record method), 65
recommended B1() (sage.interfaces.ecm.ECM method), 25
reduce_load() (in module sage.interfaces.gap), 59
reduce_load() (in module sage.interfaces.mathematica), 162
reduce load() (in module sage.interfaces.singular), 313
reduce load Axiom() (in module sage.interfaces.axiom), 20
reduce_load_element() (in module sage.interfaces.sage0), 288
reduce load fricas() (in module sage.interfaces.fricas), 43
reduce load GAP() (in module sage.interfaces.gap), 60
reduce load Giac() (in module sage.interfaces.giac), 80
reduce load GP() (in module sage.interfaces.gp), 91
reduce load Kash() (in module sage.interfaces.kash), 102
reduce load lie() (in module sage.interfaces.lie), 109
reduce_load_Lisp() (in module sage.interfaces.lisp), 114
reduce_load_macaulay2() (in module sage.interfaces.macaulay2), 123
reduce load Magma() (in module sage.interfaces.magma), 143
reduce load Maple() (in module sage.interfaces.maple), 153
reduce_load_Matlab() (in module sage.interfaces.matlab), 166
reduce load Maxima() (in module sage.interfaces.maxima), 176
reduce load Maxima function() (in module sage.interfaces.maxima), 176
reduce_load_MaximaAbstract_function() (in module sage.interfaces.maxima_abstract), 193
reduce_load_MaximaLib() (in module sage.interfaces.maxima_lib), 207
reduce_load_mupad() (in module sage.interfaces.mupad), 214
reduce load Octave() (in module sage.interfaces.octave), 224
reduce_load_R() (in module sage.interfaces.r), 281
reduce_load_Sage() (in module sage.interfaces.sage0), 288
reduce load Singular() (in module sage.interfaces.singular), 314
RElement (class in sage.interfaces.r), 279
remove constants() (in module sage.interfaces.tides), 325
remove output labels() (in module sage.interfaces.macaulay2), 123
remove repeated() (in module sage.interfaces.tides), 325
repr() (sage.interfaces.macaulay2.Macaulay2Element method), 120
require() (sage.interfaces.r.R method), 278
restart() (sage.interfaces.macaulay2.Macaulay2 method), 118
RFunction (class in sage.interfaces.r), 280
RFunctionElement (class in sage.interfaces.r), 280
ring() (sage.interfaces.macaulay2.Macaulay2 method), 118
ring() (sage.interfaces.singular.Singular method), 304
Sage (class in sage.interfaces.sage0), 285
sage() (sage.interfaces.interface.InterfaceElement method), 6
sage.interfaces.axiom (module), 15
sage.interfaces.cleaner (module), 329
sage.interfaces.ecm (module), 21
sage.interfaces.expect (module), 9
sage.interfaces.four ti 2 (module), 27
sage.interfaces.fricas (module), 33
```

```
sage.interfaces.frobby (module), 45
sage.interfaces.gap (module), 49
sage.interfaces.gap3 (module), 61
sage.interfaces.gfan (module), 71
sage.interfaces.giac (module), 73
sage.interfaces.gnuplot (module), 81
sage.interfaces.gp (module), 83
sage.interfaces.interface (module), 3
sage.interfaces.jmoldata (module), 93
sage.interfaces.kash (module), 95
sage.interfaces.lie (module), 103
sage.interfaces.lisp (module), 111
sage.interfaces.macaulay2 (module), 115
sage.interfaces.magma (module), 125
sage.interfaces.magma free (module), 145
sage.interfaces.maple (module), 147
sage.interfaces.mathematica (module), 155
sage.interfaces.matlab (module), 163
sage.interfaces.maxima (module), 167
sage.interfaces.maxima_abstract (module), 177
sage.interfaces.maxima_lib (module), 195
sage.interfaces.mupad (module), 211
sage.interfaces.mwrank (module), 215
sage.interfaces.octave (module), 219
sage.interfaces.phc (module), 225
sage.interfaces.povray (module), 233
sage.interfaces.psage (module), 235
sage.interfaces.qepcad (module), 237
sage.interfaces.gsieve (module), 265
sage.interfaces.quit (module), 331
sage.interfaces.r (module), 269
sage.interfaces.read data (module), 333
sage.interfaces.rubik (module), 283
sage.interfaces.sage0 (module), 285
sage.interfaces.sagespawn (module), 13
sage.interfaces.scilab (module), 289
sage.interfaces.singular (module), 295
sage.interfaces.tachyon (module), 317
sage.interfaces.tides (module), 321
sage0 console() (in module sage.interfaces.sage0), 288
sage0 version() (in module sage.interfaces.sage0), 288
sage2matlab matrix string() (sage.interfaces.matlab.Matlab method), 165
sage2octave_matrix_string() (sage.interfaces.octave.Octave method), 222
sage2scilab matrix string() (sage.interfaces.scilab.Scilab method), 292
sage_flattened_str_list() (sage.interfaces.singular.SingularElement method), 308
sage global ring() (sage.interfaces.singular.SingularElement method), 308
sage_matrix() (sage.interfaces.singular.SingularElement method), 309
sage_poly() (sage.interfaces.singular.SingularElement method), 310
sage polystring() (sage.interfaces.macaulay2.Macaulay2Element method), 120
sage_polystring() (sage.interfaces.singular.SingularElement method), 311
```

```
sage rat() (in module sage.interfaces.maxima lib), 207
sage_structured_str_list() (sage.interfaces.singular.SingularElement method), 311
SageElement (class in sage.interfaces.sage0), 288
SageFunction (class in sage.interfaces.sage0), 288
SagePtyProcess (class in sage.interfaces.sagespawn), 13
SageSpawn (class in sage.interfaces.sagespawn), 14
sample point() (sage.interfaces.qepcad.QepcadCell method), 250
sample point dict() (sage.interfaces.qepcad.QepcadCell method), 250
save_as_start() (sage.interfaces.phc.PHC_Object method), 228
save image() (sage.interfaces.mathematica.MathematicaElement method), 161
save workspace() (sage.interfaces.gap.Gap method), 52
Scilab (class in sage.interfaces.scilab), 292
scilab console() (in module sage.interfaces.scilab), 294
scilab version() (in module sage.interfaces.scilab), 294
ScilabElement (class in sage.interfaces.scilab), 293
server() (sage.interfaces.expect.Expect method), 11
set() (sage.interfaces.axiom.PanAxiom method), 18
set() (sage.interfaces.fricas.FriCAS method), 38
set() (sage.interfaces.gap.Gap method), 52
set() (sage.interfaces.giac.Giac method), 78
set() (sage.interfaces.gp.Gp method), 88
set() (sage.interfaces.interface.Interface method), 5
set() (sage.interfaces.kash.Kash method), 102
set() (sage.interfaces.lie.LiE method), 108
set() (sage.interfaces.lisp.Lisp method), 113
set() (sage.interfaces.macaulay2.Macaulay2 method), 118
set() (sage.interfaces.magma.Magma method), 136
set() (sage.interfaces.maple.Maple method), 152
set() (sage.interfaces.mathematica.Mathematica method), 161
set() (sage.interfaces.matlab.Matlab method), 166
set() (sage.interfaces.matlab.MatlabElement method), 166
set() (sage.interfaces.maxima.Maxima method), 174
set() (sage.interfaces.maxima lib.MaximaLib method), 197
set() (sage.interfaces.mupad.Mupad method), 213
set() (sage.interfaces.octave.Octave method), 223
set() (sage.interfaces.psage.PSage method), 236
set() (sage.interfaces.r.R method), 278
set() (sage.interfaces.sage0.Sage method), 287
set() (sage.interfaces.scilab.Scilab method), 293
set() (sage.interfaces.scilab.ScilabElement method), 294
set() (sage.interfaces.singular.Singular method), 305
set_default() (sage.interfaces.gp.Gp method), 88
set_gap_memory_pool_size() (in module sage.interfaces.gap), 60
set magma attribute() (sage.interfaces.magma.MagmaElement method), 141
set_precision() (sage.interfaces.gp.Gp method), 88
set real precision() (sage.interfaces.gp.Gp method), 88
set_ring() (sage.interfaces.singular.Singular method), 306
set ring() (sage.interfaces.singular.SingularElement method), 311
set seed() (sage.interfaces.gap.Gap method), 52
set_seed() (sage.interfaces.gp.Gp method), 89
```

```
set seed() (sage.interfaces.interface.Interface method), 5
set_seed() (sage.interfaces.magma.Magma method), 136
set seed() (sage.interfaces.maxima.Maxima method), 175
set seed() (sage.interfaces.octave.Octave method), 223
set_seed() (sage.interfaces.r.R method), 278
set_seed() (sage.interfaces.scilab.Scilab method), 293
set seed() (sage.interfaces.singular.Singular method), 306
set_series_precision() (sage.interfaces.gp.Gp method), 89
set_server_and_command() (sage.interfaces.expect.Expect method), 11
set truth() (sage.interfaces.gepcad.OepcadCell method), 251
set truth value() (sage.interfaces.qepcad.Qepcad method), 248
set verbose() (sage.interfaces.magma.Magma method), 136
setring() (sage.interfaces.singular.Singular method), 306
SetVerbose() (sage.interfaces.magma.Magma method), 130
sharp() (sage.interfaces.macaulay2.Macaulay2Element method), 120
show() (sage.interfaces.mathematica.MathematicaElement method), 161
signs() (sage.interfaces.qepcad.QepcadCell method), 251
SingNot (class in sage.interfaces.rubik), 284
Singular (class in sage.interfaces.singular), 299
singular_console() (in module sage.interfaces.singular), 314
singular gb standard options() (in module sage.interfaces.singular), 314
singular version() (in module sage.interfaces.singular), 314
SingularElement (class in sage.interfaces.singular), 307
SingularError, 312
SingularFunction (class in sage.interfaces.singular), 312
SingularFunctionElement (class in sage.interfaces.singular), 312
SingularGBDefaultContext (class in sage.interfaces.singular), 312
SingularGBLogPrettyPrinter (class in sage.interfaces.singular), 312
solution dicts() (sage.interfaces.phc.PHC Object method), 229
solution extension() (sage.interfaces.gepcad.Qepcad method), 248
solutions() (sage.interfaces.phc.PHC Object method), 229
solve() (sage.interfaces.rubik.CubexSolver method), 283
solve() (sage.interfaces.rubik.DikSolver method), 283
solve() (sage.interfaces.rubik.OptimalSolver method), 283
solve linear() (sage.interfaces.maxima abstract.MaximaAbstract method), 184
solve_linear_system() (sage.interfaces.octave.Octave method), 223
source() (sage.interfaces.maple.Maple method), 152
source() (sage.interfaces.r.R method), 279
sr_integral() (sage.interfaces.maxima_lib.MaximaLib method), 198
sr limit() (sage.interfaces.maxima lib.MaximaLib method), 200
sr sum() (sage.interfaces.maxima lib.MaximaLib method), 201
sr tlimit() (sage.interfaces.maxima lib.MaximaLib method), 202
sr_to_max() (in module sage.interfaces.maxima_lib), 208
starstar() (sage.interfaces.macaulay2.Macaulay2Element method), 120
start() (sage.interfaces.rubik.OptimalSolver method), 284
start cleaner() (in module sage.interfaces.cleaner), 329
start_from() (sage.interfaces.phc.PHC method), 227
stat_model() (sage.interfaces.r.RElement method), 280
stdout to string() (in module sage.interfaces.maxima lib), 208
StdOutContext (class in sage.interfaces.expect), 12
```

```
stop() (sage.interfaces.rubik.OptimalSolver method), 284
str() (sage.interfaces.gap.GapElement method), 53
str() (sage.interfaces.mathematica.MathematicaElement method), 162
str() (sage.interfaces.maxima abstract.MaximaAbstractElement method), 190
string() (sage.interfaces.singular.Singular method), 307
strip_answer() (sage.interfaces.matlab.Matlab method), 166
structure sheaf() (sage.interfaces.macaulay2.Macaulay2Element method), 120
sub() (sage.interfaces.magma.MagmaElement method), 141
subexpressions_list() (in module sage.interfaces.tides), 325
subs() (sage.interfaces.macaulay2.Macaulay2Element method), 121
subst() (sage.interfaces.maxima abstract.MaximaAbstractElement method), 190
substitute() (sage.interfaces.macaulay2.Macaulay2Element method), 121
sum() (sage.interfaces.giac.GiacElement method), 79
Т
TachyonRT (class in sage.interfaces.tachyon), 317
temp_project() (sage.interfaces.four_ti_2.FourTi2 method), 30
terminate_async() (sage.interfaces.sagespawn.SagePtyProcess method), 13
tilde() (sage.interfaces.r.RElement method), 280
time() (sage.interfaces.ecm.ECM method), 25
time() (sage.interfaces.qsieve.qsieve nonblock method), 267
tmpdir() (in module sage.interfaces.qsieve), 267
to complex() (in module sage.interfaces.octave), 224
to poly solve() (sage.interfaces.maxima lib.MaximaLibElement method), 203
to sage() (sage.interfaces.macaulay2.Macaulay2Element method), 121
type() (sage.interfaces.axiom.PanAxiomElement method), 19
type() (sage.interfaces.lie.LiEElement method), 109
type() (sage.interfaces.singular.SingularElement method), 312
U
unapply() (sage.interfaces.giac.GiacElement method), 79
unbind() (sage.interfaces.gap.Gap_generic method), 56
underscore() (sage.interfaces.macaulay2.Macaulay2Element method), 122
unit quadratic integer() (sage.interfaces.maxima abstract.MaximaAbstract method), 185
unparsed_input_form() (sage.interfaces.axiom.PanAxiomElement method), 19
usage() (sage.interfaces.povray.POVRay method), 233
usage() (sage.interfaces.tachyon.TachyonRT method), 318
use() (sage.interfaces.macaulay2.Macaulay2 method), 119
user_dir() (sage.interfaces.expect.Expect method), 11
V
validate mwrank input() (in module sage.interfaces.mwrank), 216
variable_list() (sage.interfaces.phc.PHC_Object method), 229
version() (sage.interfaces.gap.Gap_generic method), 56
version() (sage.interfaces.giac.Giac method), 78
version() (sage.interfaces.gp.Gp method), 89
version() (sage.interfaces.kash.Kash method), 102
version() (sage.interfaces.lie.LiE method), 109
version() (sage.interfaces.lisp.Lisp method), 113
version() (sage.interfaces.macaulay2.Macaulay2 method), 119
```

```
version() (sage.interfaces.magma.Magma method), 137
version() (sage.interfaces.matlab.Matlab method), 166
version() (sage.interfaces.maxima_abstract.MaximaAbstract method), 185
version() (sage.interfaces.octave.Octave method), 223
version() (sage.interfaces.r.R method), 279
version() (sage.interfaces.sage0.Sage method), 288
version() (sage.interfaces.scilab.Scilab method), 293
version() (sage.interfaces.singular.Singular method), 307
W
whos() (sage.interfaces.matlab.Matlab method), 166
whos() (sage.interfaces.scilab.Scilab method), 293
with_package() (sage.interfaces.maple.Maple method), 152
write() (sage.interfaces.magma.MagmaGBLogPrettyPrinter method), 142
write() (sage.interfaces.singular.SingularGBLogPrettyPrinter method), 312
write_array() (sage.interfaces.four_ti_2.FourTi2 method), 30
write matrix() (sage.interfaces.four ti 2.FourTi2 method), 30
write_single_row() (sage.interfaces.four_ti_2.FourTi2 method), 30
X
X() (sage.interfaces.qepcad_qepcad_formula_factory method), 258
Ζ
zsolve() (sage.interfaces.four_ti_2.FourTi2 method), 30
```