## CONTENTS

1. Built-in Functions .............................................. 1
2. Indices and Tables ........................................... 257
   Python Module Index ........................................ 259
   Index ................................................................... 261
1.1 Logarithmic functions

AUTHORS:

• Yoora Yi Tenen (2012-11-16): Add documentation for \( \log() \) (Issue #12113)
• Tomas Kalvoda (2015-04-01): Add \exp_{\text{polar}}() (Issue #18085)

\begin{verbatim}
class sage.functions.log.Function_dilog
    Bases: GinacFunction
    The dilogarithm function \( \text{Li}_2(z) = \sum_{k=1}^{\infty} z^k/k^2 \).
    This is simply an alias for \text{polylog}(2, z).

EXAMPLES:

sage: # needs sage.symbolic
sage: dilog(1)
1/6*pi^2
sage: dilog(1/2)
1/12*pi^2 - 1/2*log(2)^2
sage: dilog(x^2+1)
dilog(x^2 + 1)
sage: dilog(-1)
-1/12*pi^2
sage: dilog(-1.0)
-0.822467033424113
sage: dilog(-1.1)
-0.890838090262283
sage: dilog(1/2)
1/12*pi^2 - 1/2*log(2)^2
sage: dilog(.5)
0.582240526465012
sage: dilog(1/2).n()
0.582240526465012
sage: var('z')
z
sage: dilog(z).diff(z, 2)
log(-z + 1)/z^2 - 1/((z - 1)*z)

sage: dilog(z).series(z==1/2, 3)
(1/12*pi^2 - 1/2*log(2)^2) + (-2*log(1/2))*(z - 1/2) + (2*log(1/2) + 2)*(z - 1/2)^2 + Order(1/8*(2*z - 1)^3)

sage: latex(dilog(z))
\end{verbatim}

(continues on next page)
Dilog has a branch point at 1. Sage's floating point libraries may handle this differently from the symbolic package:
class sage.functions.log.Function_exp

Bases: GinacFunction

The exponential function, \( \exp(x) = e^x \).

EXAMPLES:

```sage
sage: # needs sage.symbolic
sage: exp(-1)
e^(-1)
sage: exp(2)
e^2
sage: exp(2).n(100)
7.3890560989306502272304274606
sage: exp(x^2 + log(x))
e^(x^2 + \log(x))
sage: exp(x^2 + log(x)).simplify()
x*e^(x^2)
sage: exp(2.5)
12.1824939607035
sage: exp(I*pi/12)
(1/4*I + 1/4)*sqrt(6) - (1/4*I - 1/4)*sqrt(2)
```

To prevent automatic evaluation, use the `hold` parameter:

```sage
>>> from sage.all import *
>>> # needs sage.symbolic
>>> exp(-Integer(1))
e^(-1)
>>> exp(Integer(2))
e^2
>>> exp(Integer(2)).n(Integer(100))
7.3890560989306502272304274606
>>> exp(x**Integer(2) + log(x))
e^(x^2 + \log(x))
```

```sage
>>> exp(x**Integer(2) + log(x)).simplify()
x*e^(x^2)
```

```sage
>>> exp(RealNumber('2.5'))
12.1824939607035
>>> exp(I*pi/Integer(12))
(1/4*I + 1/4)*sqrt(6) - (1/4*I - 1/4)*sqrt(2)
```
To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```
sage: exp(0, hold=True).simplify()  #...
→ needs sage.symbolic
1
```

For the sake of simplification, the argument is reduced modulo the period of the complex exponential function, $2\pi i$:

```
sage: k = var('k', domain='integer')  #...
→ needs sage.symbolic
sage: exp(2*k*pi*I)  #...
→ needs sage.symbolic
1
```
The precision for the result is deduced from the precision of the input. Convert the input to a higher precision explicitly if a result with higher precision is desired:

```python
sage: t = exp(RealField(100)(2)); t
# needs sage.rings.real_mpfr
7.3890560989306502272304274606
sage: t.prec()
# needs sage.rings.real_mpfr
100
sage: exp(2).n(100)
# needs sage.symbolic
7.3890560989306502272304274606
```

```python
>>> from sage.all import *

>>> t = exp(RealField(Integer(100))(Integer(2)));
# needs sage.rings.real_mpfr
7.3890560989306502272304274606

>>> t.prec()
# needs sage.rings.real_mpfr
100

>>> exp(Integer(2)).n(Integer(100))
# needs sage.symbolic
7.3890560989306502272304274606
```

```python
class sage.functions.log.Function_exp_polar
Bases: BuiltinFunction

Representation of a complex number in a polar form.

INPUT:

• z – a complex number \( z = a + ib \).

OUTPUT:

A complex number with modulus \( \exp(a) \) and argument \( b \).

If \(-\pi < b \leq \pi\) then \( \exp\text{\_polar}(z) = \exp(z) \). For other values of \( b \) the function is left unevaluated.

EXAMPLES:

The following expressions are evaluated using the exponential function:
```
The function is left unevaluated when the imaginary part of the input $z$ does not satisfy $-\pi < \Im(z) \leq \pi$:

```sage
sage: exp_polar(2*pi*I)  # needs sage.symbolic
exp_polar(2*I*pi)
sage: exp_polar(-4*pi*I)  # needs sage.symbolic
exp_polar(-4*I*pi)
```

This fixes Issue #18085:

```sage
sage: integrate(1/sqrt(1+x^3), x, algorithm='sympy')  # needs sage.symbolic
1/3*x*gamma(1/3)*hypergeometric((1/3, 1/2), (4/3,), -x^3)/gamma(4/3)
```

See also:

Examples in Sympy documentation, Sympy source code of `exp_polar`

REFERENCES:

Wikipedia article Complex_number#Polar_form

class sage.functions.log.Function_harmonic_number
Bases: BuiltinFunction
Harmonic number function, defined by:

\[ H_n = H_{n,1} = \sum_{k=1}^{n} \frac{1}{k} \]

\[ H_s = \int_0^1 \frac{1 - x^s}{1 - x} \]

See the docstring for `Function_harmonic_number_generalized()`. This class exists as callback for `harmonic_number` returned by Maxima.

```python
class sage.functions.log.Function_harmonic_number_generalized
    Bases: BuiltinFunction

Harmonic and generalized harmonic number functions, defined by:

\[ H_n = H_{n,1} = \sum_{k=1}^{n} \frac{1}{k} \]

\[ H_{n,m} = \sum_{k=1}^{n} \frac{1}{km} \]

They are also well-defined for complex argument, through:

\[ H_s = \int_0^1 \frac{1 - x^s}{1 - x} \]

\[ H_{s,m} = \zeta(m) - \zeta(m, s - 1) \]

If called with a single argument, that argument is \( s \) and \( m \) is assumed to be 1 (the normal harmonic numbers \( H_s \)).

ALGORITHM:

Numerical evaluation is handled using the mpmath and FLINT libraries.

REFERENCES:

• Wikipedia article Harmonic_number

EXAMPLES:

Evaluation of integer, rational, or complex argument:

```python
sage: harmonic_number(5)  # needs mpmath
137/60
sage: harmonic_number(3, 3)  # needs sage.symbolic
251/216
sage: harmonic_number(5/2)
-2*log(2) + 46/15
sage: harmonic_number(3., 3.)
zeta(3) - 0.0400198661225573
sage: harmonic_number(3., 3.).n(200)
1.16203703703703703703703...
```

1.1. Logarithmic functions
>>> from sage.all import *
>>> harmonic_number(Integer(5))        # needs mpmath
137/60
>>> # needs sage.symbolic
>>> harmonic_number(Integer(3), Integer(3))
251/216
>>> harmonic_number(Integer(5)/Integer(2))
-2*log(2) + 46/15
>>> harmonic_number(RealNumber(3.), Integer(3))
zeta(3) ~ 0.0400198661225573
>>> harmonic_number(RealNumber(3.), RealNumber(3.))
1.16203703703703703703703...
>>> harmonic_number(Integer(3), Integer(3)).n(Integer(200))
1.57436810798989 - 1.06194728851357*I

Solutions to certain sums are returned in terms of harmonic numbers:

    sage: k = var('k')                #...
    → needs sage.symbolic
    sage: sum(1/k^7,k,1,x)            #...
    → needs sage.symbolic
    harmonic_number(x, 7)

Check the defining integral at a random integer:

    sage: n = randint(10,100)
    sage: bool(SR(integrate((1-x^n)/(1-x),x,0,1)) == harmonic_number(n))  #...
    → needs sage.symbolic
    True

There are several special values which are automatically simplified:

    sage: harmonic_number(0)         #...
    → mpmath
    0
    sage: harmonic_number(1)         #...
    → mpmath
    1

(continues on next page)
Functions, Release 10.4

sage: harmonic_number(x, 1)  # needs sage.symbolic
harmonic_number(x)

>>> from sage.all import *
>>> harmonic_number(Integer(0))  # needs mpmath
0
>>> harmonic_number(Integer(1))  # needs mpmath
1
>>> harmonic_number(x, Integer(1))  # needs sage.symbolic
harmonic_number(x)

class sage.functions.log.Function_lambert_w

Bases: BuiltinFunction

The integral branches of the Lambert W function $W_n(z)$.

This function satisfies the equation

$$z = W_n(z)e^{W_n(z)}$$

INPUT:

- $n$ – an integer. $n = 0$ corresponds to the principal branch.
- $z$ – a complex number

If called with a single argument, that argument is $z$ and the branch $n$ is assumed to be 0 (the principal branch).

ALGORITHM:

Numerical evaluation is handled using the mpmath and SciPy libraries.

REFERENCES:

- Wikipedia article Lambert_W_function

EXAMPLES:

Evaluation of the principal branch:

sage: lambert_w(1.0)  # needs scipy
0.567143290409784
sage: lambert_w(-1).n()  # needs mpmath
-0.318131505204764 + 1.33723570143069*I
sage: lambert_w(-1.5 + 5*I)  # needs mpmath sage.symbolic
1.17418016254171 + 1.10651494102011*I

>>> from sage.all import *
>>> lambert_w(RealNumber('1.0'))  # needs scipy
0.567143290409784
>>> lambert_w(-Integer(1)).n()  # needs mpmath sage.symbolic
(continues on next page)
Evaluation of other branches:

```python
sage: lambert_w(2, 1.0)  # needs scipy
-2.40158510486800 + 10.7762995161151*I
```

Solutions to certain exponential equations are returned in terms of lambert_w:

```python
sage: S = solve(e**(5*x) + x == 0, x, to_poly_solve=True)  # needs sage.symbolic
sage: z = S[0].rhs(); z  # needs sage.symbolic
-1/5*lambert_w(5)
sage: N(z)  # needs sage.symbolic
-0.265344933048440
```

Check the defining equation numerically at \( z = 5 \):

```python
sage: N(lambert_w(5)*exp(lambert_w(5)) - 5)  # needs mpmath
0.000000000000000
```

There are several special values of the principal branch which are automatically simplified:

```python
sage: lambert_w(0)  # needs mpmath
0
sage: lambert_w(e)  # needs mpmath
```

(continues on next page)
\begin{verbatim}
needs sage.symbolic
1
sage: lambert_w(-1/e) # ...
needs sage.symbolic
-1

>>> from sage.all import *
>>> lambert_w(Integer(0)) ... # needs mpmath
0
>>> lambert_w(e) ... # needs sage.symbolic
1
>>> lambert_w(-Integer(1)/e) ... # needs sage.symbolic
-1

Integration (of the principal branch) is evaluated using Maxima:

\begin{verbatim}
sage: integrate(lambert_w(x), x) ... # needs sage.symbolic
(lambert_w(x)^2 - lambert_w(x) + 1)*x/lambert_w(x)
sage: integrate(lambert_w(x), x, 0, 1) ... # needs sage.symbolic
(lambert_w(1)^2 - lambert_w(1) + 1)/lambert_w(1) - 1
sage: integrate(lambert_w(x), x, 0, RealNumber('1.0')) ... # needs sage.symbolic
0.3303661247616807

>>> from sage.all import *
>>> integrate(lambert_w(x), x) ... # needs sage.symbolic
(lambert_w(x)^2 - lambert_w(x) + 1)*x/lambert_w(x)

\end{verbatim}

Warning: The integral of a non-principal branch is not implemented, neither is numerical integration using GSL. The \texttt{numerical_integral()} function does work if you pass a lambda function:

\begin{verbatim}
sage: numerical_integral(lambda x: lambert_w(x), 0, 1) ... # needs sage.modules
(0.33036612476168054, 3.667800782666048e-15)

>>> from sage.all import *
>>> numerical_integral(lambda x: lambert_w(x), Integer(0), Integer(1)) ... # needs sage.modules
(0.33036612476168054, 3.667800782666048e-15)
\end{verbatim}

\texttt{class sage.functions.log.Function_log1}

\begin{verbatim}
Bases: GinacFunction

The natural logarithm of \texttt{x}.
\end{verbatim}
\end{verbatim}

1.1. Logarithmic functions
See \texttt{log()} for extensive documentation.

**EXAMPLES:**

```python
sage: \ln(e^2) #... 
\rightarrow \text{needs sage.symbolic}
2
sage: \ln(2) #... 
\rightarrow \text{needs sage.symbolic}
log(2)
sage: \ln(10) #... 
\rightarrow \text{needs sage.symbolic}
log(10)
```

```python
>>> \texttt{from sage.all import *}
>>> \ln(e**Integer(2)) # needs sage.symbolic
2
>>> \ln(Integer(2)) # needs sage.symbolic
log(2)
>>> \ln(Integer(10)) # needs sage.symbolic
log(10)
```

class \texttt{sage.functions.log.Function_log2}

Bases: \texttt{GinacFunction}

Return the logarithm of \(x\) to the given base.

See \texttt{log()} for extensive documentation.

**EXAMPLES:**

```python
sage: \texttt{from sage.functions.log import logb}
sage: \logb(1000, 10) #...
\rightarrow \text{needs sage.symbolic}
3
```

```python
>>> \texttt{from sage.all import *}
>>> \texttt{from sage.functions.log import logb}
>>> \logb(Integer(1000), Integer(10)) # needs sage.symbolic
3
```

class \texttt{sage.functions.log.Function_polylog}

Bases: \texttt{GinacFunction}

The polylog function \(\text{Li}_s(z) = \sum_{k=1}^{\infty} z^k/k^s\).

The first argument is \(s\) (usually an integer called the weight) and the second argument is \(z\): \texttt{polylog}(s, z).

This definition is valid for arbitrary complex numbers \(s\) and \(z\) with \(|z| < 1\). It can be extended to \(|z| \geq 1\) by the process of analytic continuation, with a branch cut along the positive real axis from 1 to \(+\infty\). A \texttt{NaN} value may be returned for floating point arguments that are on the branch cut.

**EXAMPLES:**
sage: # needs sage.symbolic
sage: polylog(2.7, 0)
0.000000000000000
sage: polylog(2, 1)
1/6*pi^2
sage: polylog(2, -1)
-1/12*pi^2
sage: polylog(3, -1)
-3/4*zeta(3)
sage: polylog(2, I)
I*catalan - 1/48*pi^2
sage: polylog(4, 1/2)
0.517479061673899

sage: polylog(4, 0.5)
0.517479061673899

sage: # needs sage.symbolic
sage: polylog(1, x)
-log(-x + 1)
sage: polylog(2, x^2 + 1)
dilog(x^2 + 1)

sage: f = polylog(4, 1); f
1/90*pi^4
sage: f.n()
1.08232323371114

sage: polylog(4, 2).n()
2.42786280675470 - 0.174371300025453*I
sage: complex(polylog(4, 2))
(2.4278628067547032-0.17437130002545306j)
sage: float(polylog(4, 0.5))
0.5174790616738993

sage: z = var('z')
sage: polylog(2, z).series(z==0, 5)
1*z + 1/4*z^2 + 1/9*z^3 + 1/16*z^4 + Order(z^5)

sage: loads(dumps(polylog))
polylog

sage: latex(polylog(5, x))
\mathrm{Li}_5(x)

from sage.all import *
>>> polylog(RealNumber('2.7'), Integer(0))
0.000000000000000
>>> polylog(Integer(2), Integer(1))
1/6*pi^2
>>> polylog(Integer(2), -Integer(1))
-1/12*pi^2
>>> polylog(Integer(3), -Integer(1))
-3/4*zeta(3)
>>> polylog(Integer(2), I)
I*catalan - 1/48*pi^2

(continues on next page)
1.2 Trigonometric functions

class sage.functions.trig.Function_arccos

Bases: GinacFunction

The arccosine function.

EXAMPLES:

sage: arccos(0.5) #-
1.04719755119660
sage: arccos(1/2) #-
1.04719755119660
sage: arccos(1 + 1.0*I) #-
1.04719755119660
sage: arccos(3/4).n(100) #-
0.7227342478134156117837735264
```python
>>> from sage.all import *
>>> arccos(RealNumber('0.5'))
# needs sage.rings.real_mpfr
1.04719755119660
>>> arccos(Integer(1)/Integer(2))
# needs sage.symbolic
1/3*pi
>>> arccos(Integer(1) + RealNumber('1.0')*I)
# needs sage.symbolic
0.904556894302381 - 1.06127506190504*I
>>> arccos(Integer(3)/Integer(4)).n(Integer(100))
# needs sage.symbolic
0.72273424781341561117837735264
```

We can delay evaluation using the `hold` parameter:

```python
sage: arccos(0, hold=True)  #...

```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: a = arccos(0, hold=True); a.simplify()  #...
1/2*pi
```

\[
\text{conjugate}(\text{arccos}(x)) = \text{arccos}(\text{conjugate}(x)), \text{ unless on the branch cuts, which run along the real axis outside the interval } [-1, +1].: 
```

```python
sage: # needs sage.symbolic
sage: y
sage: y
sage: y
sage: y
sage: y
```

1.2. Trigonometric functions 15
```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> conjugate(arccos(x))
conjugate(arccos(x))
>>> var('y', domain='positive')
y
>>> conjugate(arccos(y))
conjugate(arccos(y))
>>> conjugate(arccos(y+I))
conjugate(arccos(y + I))
>>> conjugate(arccos(Integer(1)/Integer(16)))
arccos(1/16)
>>> conjugate(arccos(Integer(2))]
conjugate(arccos(2))
>>> conjugate(arccos(-Integer(2))]
pi - conjugate(arccos(2))
```

```python
class sage.functions.trig.Function_arccot
Bases: GinacFunction

The arccotangent function.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: arccot(1/2)
arccot(1/2)
```
```python
sage: RDF(arccot(1/2))  # abs tol 2e-16
1.1071487177940906
```
```python
sage: arccot(1 + I)
arccot(I + 1)
```
```python
sage: arccot(1/2).n(100)
1.1071487177940905030170654602
```
```python
sage: float(arccot(1/2))  # abs tol 2e-16
1.1071487177940906
```
```python
sage: bool(diff(acot(x), x) == -diff(atan(x), x))
True
```
```python
sage: diff(acot(x), x)
-1/(x^2 + 1)
```
```python
```
We can delay evaluation using the hold parameter:
```
sage: arccot(1, hold=True)  # needs sage.symbolic
arccot(1)

From sage.all import *
>>> arccot(Integer(1), hold=True)  ...
    -> # needs sage.symbolic
    arccot(1)

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():

sage: a = arccot(1, hold=True); a.simplify()  # needs sage.symbolic
1/4*pi

class sage.functions.trig.Function_arccsc

Bases: GinacFunction

The arccosecant function.

EXAMPLES:

sage: # needs sage.symbolic
sage: arccsc(2)
arccsc(2)
sage: RDF(arccsc(2))  # rel tol 1e-15
0.5235987755982988
sage: arccsc(2).n(100)
0.52359877559829887307710723055
sage: float(arccsc(2))
0.52359877559829...

>>> from sage.all import *

>>> a = arccot(Integer(1), hold=True); a.simplify()  ...
    -> # needs sage.symbolic
    1/4*pi

>>> from sage.all import *

>>> # needs sage.symbolic
>>> arccsc(Integer(2))
arccsc(2)

>>> RDF(arccsc(Integer(2)))  # rel tol 1e-15
0.5235987755982988

(continues on next page)
We can delay evaluation using the `hold` parameter:

```python
sage: arccsc(1, hold=True)  # needs sage.symbolic
arccsc(1)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: a = arccsc(1, hold=True); a.simplify()  # needs sage.symbolic
1/2*pi
```

```python
sage: from sage.all import *

sage: a = arccsc(Integer(1), hold=True); a.simplify()  # needs sage.symbolic
1/2*pi
```

### class `sage.functions.trig.Function_arccsc`

**Bases:** `GinacFunction`

The arcsecant function.

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: arccsc(2)
```

```python
sage: arccsc(2.0)
1.04719755119660
```

```python
sage: arccsc(2.0).n(100)
```

```python
sage: arccsc(1/2).n(100)
```

```python
sage: arccsc(1/2).n(100)
```

```python
sage: 1.3169578969248167086250463473*I
```

```python
sage: RDF(arccsc(2))  # abs tol 1e-15
```

```python
sage: arccsc(1 + I)
```

```python
sage: diff(asec(x), x)
```

```python
sage: asec(x)._sympy_()  # needs sympy
```

```python
sage: asec(x)
```
We can delay evaluation using the hold parameter:

```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> arcsin(Integer(1), hold=True)  # needs sage.symbolic
arcsin(1)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```
>>> from sage.all import *
>>> a = arcsin(Integer(1), hold=True); a.simplify()  # needs sage.symbolic
0
```

```python
class sage.functions.trig.Function_arcsin
Bases: GinacFunction

The arcsine function.

EXAMPLES:
```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> arcsin(0.5)  # needs sage.rings.real_mpfr
0.523598775598299
>>> arcsin(1/2)  # needs sage.symbolic
1/6*pi
>>> arcsin(1 + 1.0*I)  # needs sage.symbolic
```
(continues on next page)
We can delay evaluation using the `hold` parameter:

```python
sage: arcsin(0, hold=True)  # needs sage.symbolic
arcsin(0)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: a = arcsin(0, hold=True); a.simplify()  # needs sage.symbolic
0
```

In this case, we have:

\[
\text{conjugate}(\text{arcsin}(x)) = \text{arcsin}(\text{conjugate}(x)),
\]

unless on the branch cuts which run along the real axis outside the interval \([-1, +1]\).:
```python
>>> from sage.all import *
>>> # needs sage.symbolic
conjugate(arcsin(x))
conjugate(arcsin(x))
>>> var('y', domain='positive')
y
>>> conjugate(arcsin(y))
conjugate(arcsin(y))
>>> conjugate(arcsin(y+I))
conjugate(arcsin(y + I))
>>> conjugate(arcsin(Integer(1)/Integer(16)))
arcsin(1/16)
>>> conjugate(arcsin(Integer(2)))
conjugate(arcsin(2))
>>> conjugate(arcsin(-Integer(2)))
-conjugate(arcsin(2))
```

class `sage.functions.trig.Function_arctan`

Bases: `GinacFunction`

The arctangent function.

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: arctan(1/2)
arctan(1/2)
sage: RDF(arctan(1/2))  # rel tol 1e-15
0.46364760900080615
sage: arctan(1 + I)
arctan(I + 1)
sage: arctan(1/2).n(100)
0.46364760900080611621425623146
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
arctan(Integer(1)/Integer(2))  # needs sage.symbolic
```

We can delay evaluation using the `hold` parameter:

```python
sage: arctan(0, hold=True)  # needs sage.symbolic
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

### 1.2. Trigonometric functions
Functions, Release 10.4

```
sage: a = arctan(0, hold=True); a.simplify()  # needs sage.symbolic
0
```

```bash
>>> from sage.all import *

>>> a = arctan(Integer(0), hold=True); a.simplify()  # needs sage.symbolic
0
```

\( \text{conjugate}(\arctan(x)) = \arctan(\text{conjugate}(x)) \), unless on the branch cuts which run along the imaginary axis outside the interval \([-I, +I]\).

```
sage: # needs sage.symbolic
sage: conjugate(arctan(x))
conjugate(arctan(x))
sage: var('y', domain='positive')
y
sage: conjugate(arctan(y))
arctan(y)
sage: conjugate(arctan(y+I))
conjugate(arctan(y + I))
sage: conjugate(arctan(1/16))
arctan(1/16)
sage: conjugate(arctan(-2*I))
conjugate(arctan(-2*I))
sage: conjugate(arctan(2*I))
conjugate(arctan(2*I))
sage: conjugate(arctan(1/2))
arctan(-1/2*I)
```

```
>>> from sage.all import *

>>> # needs sage.symbolic

>>> conjugate(arctan(x))
conjugate(arctan(x))

>>> var('y', domain='positive')
y

>>> conjugate(arctan(y))
arctan(y)

>>> conjugate(arctan(y+I))
conjugate(arctan(y + I))

>>> conjugate(arctan(Integer(1)/Integer(16)))
arctan(1/16)

>>> conjugate(arctan(-Integer(2)*I))
conjugate(arctan(-2*I))

>>> conjugate(arctan(Integer(2)*I))
conjugate(arctan(2*I))

>>> conjugate(arctan(Integer(1)/Integer(2)))
arctan(-1/2*I)
```

```python
class sage.functions.trig.Function_arctan2

    Bases: GinacFunction

    The modified arctangent function.

    Returns the arc tangent (measured in radians) of \( y/x \), where unlike \( \arctan(y/x) \), the signs of both \( x \) and \( y \) are considered. In particular, this function measures the angle of a ray through the origin and \((x, y)\), with the positive \( x \)-axis the zero mark, and with output angle \( \theta \) being between \(-\pi < \theta \leq \pi\).
```

22 Chapter 1. Built-in Functions
Hence, \( \text{arctan2}(y, x) = \text{arctan}(y/x) \) only for \( x > 0 \). One may consider the usual \text{arctan} to measure angles of lines through the origin, while the modified function measures rays through the origin.

Note that the \( y \)-coordinate is by convention the first input.

**EXAMPLES:**

Note the difference between the two functions:

<table>
<thead>
<tr>
<th>Sage</th>
<th>Arctan2(1, -1)</th>
<th># needs sage.symbolic</th>
<th>3/4*pi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sage</td>
<td>Arctan(1/-1)</td>
<td># needs sage.symbolic</td>
<td>-1/4*pi</td>
</tr>
</tbody>
</table>

```python
>>> from sage.all import *

sage: arctan2(Integer(1), -Integer(1)) \# needs sage.symbolic
(3*%pi)/4

sage: arctan(Integer(1)/-Integer(1)) \# needs sage.symbolic
-1/4*pi
```

This is consistent with Python and Maxima:

<table>
<thead>
<tr>
<th>Sage</th>
<th>Maxima atan2(1, -1)</th>
<th># needs sage.symbolic</th>
<th>(3*%pi)/4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sage</td>
<td>Math.atan2(1, -1)</td>
<td># needs sage.symbolic</td>
<td>2.356194490192345</td>
</tr>
</tbody>
</table>

```python
>>> from sage.all import *

sage: maxima.atan2(Integer(1), -Integer(1)) \# needs sage.symbolic
(3*%pi)/4

sage: math.atan2(Integer(1), -Integer(1))
2.356194490192345
```

More examples:

<table>
<thead>
<tr>
<th>Sage</th>
<th>Arctan2(1, 0)</th>
<th># needs sage.symbolic</th>
<th>1/2*pi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sage</td>
<td>Arctan2(2, 3)</td>
<td># needs sage.symbolic</td>
<td>arctan(2/3)</td>
</tr>
<tr>
<td>Sage</td>
<td>Arctan2(-1, -1)</td>
<td># needs sage.symbolic</td>
<td>-3/4*pi</td>
</tr>
</tbody>
</table>

```python
>>> from sage.all import *

sage: arctan2(Integer(1), Integer(0)) \# needs sage.symbolic
1/2*pi

sage: arctan2(Integer(2), Integer(3)) \# needs sage.symbolic
arctan(2/3)
```

(continues on next page)
Of course we can approximate as well:

```python
sage: arctan2(-1/2, 1).n(100)  # needs sage.symbolic
-0.46364760900080611621425623146
sage: arctan2(2, 3).n(100)  # needs sage.symbolic
0.58800260354756755124561108063
```

We can delay evaluation using the `hold` parameter:

```python
sage: arctan2(-1/2, 1, hold=True)  # needs sage.symbolic
arctan2(-1/2, 1)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: arctan2(-1/2, 1, hold=True).simplify()  # needs sage.symbolic
-arctan(1/2)
```

The function also works with numpy arrays as input:

```python
sage: # needs numpy
sage: import numpy
sage: a = numpy.linspace(1, 3, 3)
sage: b = numpy.linspace(3, 6, 3)
sage: atan2(a, b)
array([0.32175055, 0.41822433, 0.46364761])
sage: atan2(1,a)  # needs numpy
```

(continues on next page)
array([0.78539816, 0.46364761, 0.32175055])

sage: atan2(a, 1) # needs numpy
array([0.78539816, 1.10714872, 1.24904577])

>>> from sage.all import *
>>> # needs numpy
>>> import numpy
>>> a = numpy.linspace(Integer(1), Integer(3), Integer(3))
>>> b = numpy.linspace(Integer(3), Integer(6), Integer(3))
>>> atan2(a, b)
array([0.32175055, 0.41822433, 0.46364761])

>>> atan2(Integer(1),a) # needs numpy
array([0.78539816, 0.46364761, 0.32175055])

>>> atan2(a, Integer(1)) # needs numpy
array([0.78539816, 1.10714872, 1.24904577])

class sage.functions.trig.Function_cos

Bases: GinacFunction

The cosine function.

EXAMPLES:

sage: # needs sage.symbolic
cos(pi)
-1

sage: cos(x).subs(x==pi)
-1

sage: cos(2)._sympy_() # needs sympy

>>> from sage.all import *
>>> # needs sage.symbolic
>>> cos(pi)
-1

>>> cos(x).subs(x==pi)
-1

>>> cos(Integer(2))._sympy_() # needs...

We can prevent evaluation using the hold parameter:

sage: cos(0, hold=True) # needs sage.symbolic

1.2. Trigonometric functions
To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: a = cos(0, hold=True); a.simplify()  # needs sage.symbolic
1
```

If possible, the argument is also reduced modulo the period length $2\pi$, and well-known identities are directly evaluated:

```python
sage: # needs sage.symbolic
sage: k = var('k', domain='integer')
sage: cos(1 + 2*k*pi)  
cos(1)  
sage: cos(k*pi)  
cos(pi*k)  
sage: cos(pi/3 + 2*k*pi)  
1/2
```

```
>>> from sage.all import *  
>>> # needs sage.symbolic

```

```python
class sage.functions.trig.Function_cot

Bases: GinacFunction

The cotangent function.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: cot(pi/4)  
1
```
We can prevent evaluation using the hold parameter:

```sage
sage: cot(pi/4, hold=True)
1
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```sage
sage: a = cot(pi/4, hold=True); a.simplify()
1
```

**EXAMPLES:**

```sage
sage: # needs sage.symbolic
sage: cot(pi/4)
1
sage: cot(x).subs(x==pi/4)
1
sage: cot(pi/7)
```

(continues on next page)
Functions, Release 10.4

cot(1/7*pi)
sage: cot(x)
cot(x)

sage: # needs sage.symbolic
case: n(cot(pi/4), 100)
1.000000000000000000000000000000
sage: float(cot(1))
0.64209261593433...

sage: bool(diff(cot(x), x) == diff(1/tan(x), x))
True

sage: diff(cot(x), x)
-cot(x)^2 - 1

>>> from sage.all import *
>>> # needs sage.symbolic
>>> cot(pi/Integer(4))
1
>>> cot(x).subs(x==pi/Integer(4))
1
>>> cot(pi/Integer(7))
cot(1/7*pi)
>>> cot(x)
cot(x)

>>> # needs sage.symbolic
>>> n(cot(pi/Integer(4)), Integer(100))
1.000000000000000000000000000000
>>> float(cot(Integer(1)))
0.64209261593433...
>>> bool(diff(cot(x), x) == diff(Integer(1)/tan(x), x))
True

>>> diff(cot(x), x)
-cot(x)^2 - 1

class sage.functions.trig.Function_csc
Bases: GinacFunction

The cosecant function.

EXAMPLES:

sage: # needs sage.symbolic
sage: csc(pi/4)
sqrt(2)
sage: csc(x).subs(x==pi/4)
sqrt(2)
sage: csc(pi/7)
csc(1/7*pi)
sage: csc(x)
csc(x)
sage: RR(csc(pi/4))
1.41421356237310

sage: n(csc(pi/4), 100)
1.4142135623730950488016887242
sage: float(csc(pi/4))
1.4142135623730951

(continues on next page)
sage: csc(1/2)
csc(1/2)
sage: csc(0.5)
2.08582964293349

sage: # needs sage.symbolic
sage: bool(diff(csc(x), x) == diff(1/sin(x), x))
True
sage: diff(csc(x), x)
-cot(x)*csc(x)
sage: latex(csc(x))
\csc\left(x\right)
sage: csc(x)._sympy_( )

We can prevent evaluation using the hold parameter:

sage: csc(pi/4, hold=True)
\csc\left(\frac{1}{4}\pi\right)

(continues on next page)
To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: a = csc(pi/4, hold=True); a.simplify() # needs sage.symbolic
sqrt(2)
```

```python
>>> from sage.all import *
>>> a = csc(pi/Integer(4), hold=True); a.simplify() # needs sage.symbolic
sqrt(2)
```

```python
class sage.functions.trig.Function_sec

Bases: GinacFunction

The secant function.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: sec(pi/4)
sqrt(2)

sage: sec(x).subs(x==pi/4)
sqrt(2)

sage: sec(pi/7)
sec(1/7*pi)
sage: sec(x)
sec(x)

sage: RR(sec(pi/4))
1.41421356237310

sage: n(sec(pi/4),100)
1.4142135623730950488016887242

sage: float(sec(pi/4))
1.4142135623730951

sage: sec(1/2)
sec(1/2)
sage: sec(0.5)
1.13949392732455

sage: # needs sage.symbolic
sage: bool(diff(sec(x), x) == diff(1/cos(x), x)) # needs sage.symbolic
True

sage: diff(sec(x), x)
sec(x)*tan(x)

sage: latex(sec(x))
\sec\left(x\right)

sage: sec(x)._sympy_() # needs sympy
sec(x)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> sec(pi/Integer(4))
(continues on next page)
```
We can prevent evaluation using the hold parameter:

```
sage: sec(pi/4, hold=True)  # needs sage.symbolic
sec(1/4*pi)
```

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():

```
sage: a = sec(pi/4, hold=True); a.simplify()  # needs sage.symbolic
sqrt(2)
```

```python
class sage.functions.trig.Function_sin
    Bases: GinacFunction

The sine function.
```
EXAMPLES:

```
from sage.all import *

# needs sage.symbolic
sin(Integer(0), hold=True)
# needs sage.symbolic
sin(0)

# needs sage.symbolic
sin(Integer(0), hold=True)  # needs sage.symbolic
sin(0)

# needs sage.symbolic
sin(Integer(0), hold=True)  # needs sage.symbolic
sin(0)
```

We can prevent evaluation using the `hold` parameter:

```
sage: sin(0, hold=True)
# needs sage.symbolic
sin(0)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```
sage: a = sin(0, hold=True); a.simplify()  # needs sage.symbolic
0
```

```
sage: a = sin(Integer(0), hold=True); a.simplify()  # needs sage.symbolic
0
```

If possible, the argument is also reduced modulo the period length $2\pi$, and well-known identities are directly evaluated:

```
sage: x = var('x', domain='integer')  # needs sage.symbolic
```

```
sage: sin(1 + 2*x*pi)  # needs sage.symbolic
sin(1)
```

(continues on next page)
class sage.functions.trig.Function_tan

Bases: GinacFunction

The tangent function.

EXAMPLES:

sage: # needs sage.rings.real_mpfr
sage: tan(3.1415)
-0.0000926535900581913
sage: tan(3.1415/4)
0.999953674278156

sage: # needs sage.symbolic
sage: tan(pi)
0
sage: tan(pi/4)
1
sage: tan(1/2)
tan(1/2)
sage: RR(tan(1/2))
0.546302489843790

>>> from sage.all import *
>>> # needs sage.rings.real_mpfr
>>> tan(RealNumber('3.1415'))
-0.0000926535900581913
>>> tan(RealNumber('3.1415')/Integer(4))
0.999953674278156

>>> # needs sage.symbolic
>>> tan(pi)
0
>>> tan(pi/Integer(4))
1
>>> tan(Integer(1)/Integer(2))
tan(1/2)
>>> RR(tan(Integer(1)/Integer(2)))
0.546302489843790

We can prevent evaluation using the hold parameter:
Functions, Release 10.4

```
sage: tan(pi/4, hold=True)  # needs sage.symbolic
  tan(1/4*pi)

>>> from sage.all import *
>>> tan(pi/Integer(4), hold=True)   # needs sage.symbolic
  tan(pi/4)

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():

```
sage: a = tan(pi/4, hold=True); a.simplify()  # needs sage.symbolic
  1

>>> from sage.all import *
>>> a = tan(pi/Integer(4), hold=True); a.simplify()   # needs sage.symbolic
  1

```

If possible, the argument is also reduced modulo the period length $\pi$, and well-known identities are directly evaluated:

```
sage: k = var('k', domain='integer')  # needs sage.symbolic
sage: tan(1 + 2*k*pi)  # needs sage.symbolic
  tan(1)

>>> from sage.all import *
>>> k = var('k', domain='integer')   # needs sage.symbolic
>>> tan(Integer(1) + Integer(2)*k*pi)  # needs sage.symbolic
  tan(1)

```

1.3 Hyperbolic functions

The full set of hyperbolic and inverse hyperbolic functions is available:

- hyperbolic sine: $\sinh()$
- hyperbolic cosine: $\cosh()$
- hyperbolic tangent: $\tanh()$
- hyperbolic cotangent: $\coth()$
- hyperbolic secant: $\sech()$
• hyperbolic cosecant: \( \text{csch}(x) \)
• inverse hyperbolic sine: \( \text{asinh}(x) \)
• inverse hyperbolic cosine: \( \text{acosh}(x) \)
• inverse hyperbolic tangent: \( \text{atanh}(x) \)
• inverse hyperbolic cotangent: \( \text{acoth}(x) \)
• inverse hyperbolic secant: \( \text{asech}(x) \)
• inverse hyperbolic cosecant: \( \text{acsch}(x) \)

REFERENCES:
• Wikipedia article Hyperbolic function
• Wikipedia article Inverse hyperbolic functions

EXAMPLES:
Inverse hyperbolic functions have logarithmic expressions, so expressions of the form \( \exp(c \cdot f(x)) \) simplify:

```text
sage: # needs sage.symbolic
sage: exp(2*atanh(x))
-(x + 1)/(x - 1)
sage: exp(2*acoth(x))
(x + 1)/(x - 1)
sage: exp(2*asinh(x))
(x + sqrt(x^2 + 1))^2
sage: exp(2*acosh(x))
(x + sqrt(x^2 - 1))^2
sage: exp(2*asech(x))
(sqrt(-x^2 + 1)/x + 1/x)^2
sage: exp(2*acsch(x))
(sqrt(1/x^2 + 1) + 1/x)^2
```

```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> exp(Integer(2)*atanh(x))
-(x + 1)/(x - 1)
>>> exp(Integer(2)*acoth(x))
(x + 1)/(x - 1)
>>> exp(Integer(2)*asinh(x))
(x + sqrt(x^2 + 1))^2
>>> exp(Integer(2)*acosh(x))
(x + sqrt(x^2 - 1))^2
>>> exp(Integer(2)*asech(x))
(sqrt(-x^2 + 1)/x + 1/x)^2
>>> exp(Integer(2)*acsch(x))
(sqrt(1/x^2 + 1) + 1/x)^2
```

```python
class sage.functions.hyperbolic.Function_arccosh
    Bases: GinacFunction

    The inverse of the hyperbolic cosine function.
```

1.3. Hyperbolic functions 35
sage: # needs sage.symbolic
sage: acosh(1/2)
arccosh(1/2)
sage: acosh(1 + I*1.0)
1.06127506190504 + 0.904556894302381*I
sage: float(acosh(2))
1.3169578969248168
sage: cosh(float(acosh(2)))
2.0

sage: acosh(complex(1, 2))  # abs tol 1e-15
                   → needs sage.rings.complex_double
                   (1.5285709194809982+1.1437177404024204j)

>>> from sage.all import *

>>> # needs sage.symbolic
>>> acosh(Integer(1)/Integer(2))
arccosh(1/2)
>>> acosh(Integer(1) + I*RealNumber('1.0'))
1.06127506190504 + 0.904556894302381*I
>>> float(acosh(Integer(2)))
1.3169578969248168
>>> cosh(float(acosh(Integer(2)))))
2.0

>>> acosh(complex(Integer(1), Integer(2)))  # abs tol 1e-15
                   → needs sage.rings.complex_double
                   (1.5285709194809982+1.1437177404024204j)

Warning: If the input is in the complex field or symbolic (which includes rational and integer input), the output will be complex. However, if the input is a real decimal, the output will be real or NaN. See the examples for details.

sage: acosh(CC(0.5))  # needs sage.rings.real_mpfr
1.04719755119660*I

sage: # needs sage.symbolic
sage: acosh(0.5)
NaN
sage: acosh(0)
1/2*I*pi
sage: acosh(-1)
I*pi

>>> from sage.all import *

>>> acosh(CC(RealNumber('0.5')))  → needs sage.rings.real_mpfr
1.04719755119660*I

(continues on next page)
To prevent automatic evaluation use the \texttt{hold} argument:

\begin{verbatim}
 sage: acosh(-1, hold=True) # needs sage.symbolic
 arccosh(-1)
\end{verbatim}

To then evaluate again, use the \texttt{unhold} method:

\begin{verbatim}
 sage: acosh(-1, hold=True).unhold() # needs sage.symbolic
 I*pi
\end{verbatim}

\texttt{conjugate(arccosh(x))==arccosh(conjugate(x))} unless on the branch cut which runs along the real axis from $+1$ to $-\infty$:

\begin{verbatim}
 sage: # needs sage.symbolic
 sage: conjugate(acosh(x))
 conjugate(arccosh(x))
 sage: var('y', domain='positive')
y
 sage: conjugate(acosh(y))
 conjugate(arccosh(y))
 sage: conjugate(acosh(y+I))
 conjugate(arccosh(y + I))
 sage: conjugate(acosh(1/16))
 conjugate(arccosh(1/16))
 sage: conjugate(acosh(2))
 arccosh(2)
 sage: conjugate(acosh(1/2))
 arccosh(-1/2*I)
\end{verbatim}

\begin{verbatim}
 from sage.all import *
>>> from sage.all import *
\end{verbatim}
```python
>>> conjugate(acosh(x))
conjugate(arccosh(x))
>>> var('y', domain='positive')
y
>>> conjugate(acosh(y))
conjugate(arccosh(y))
>>> conjugate(acosh(y+I))
conjugate(arccosh(y + I))
>>> conjugate(acosh(Integer(1)/Integer(16)))
conjugate(arccosh(1/16))
>>> conjugate(acosh(Integer(2)))
arccosh(2)
>>> conjugate(acosh(I/Integer(2)))
arccosh(-1/2*I)
```

class sage.functions.hyperbolic.Function_arccoth

Bases: GinacFunction

The inverse of the hyperbolic cotangent function.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: acoth(2.0)
0.549306144334055
sage: acoth(2)
1/2*log(3)
```

```python
sage: bool(diff(acoth(x), x) == diff(atanh(x), x))
True
sage: diff(acoth(x), x)
-1/(x^2 - 1)
```

```python
sage: float(acoth(2))
0.5493061443340549
```

```python
sage: float(acoth(2).n(53))  # Correct result to 53 bits
0.5493061443340549
```

```python
sage: float(acoth(2).n(100))  # Compute 100 bits and then round to 53
0.5493061443340549
```

```python
>>> from sage.all import *
```
```python
>>> acoth(Integer(2)).n(Integer(200))
0.54930614433405484569762261846126285232374527891137472586735
>>> bool(diff(acoth(x), x) == diff(atanh(x), x))  # needs...
→sage.symbolic
True
>>> diff(acoth(x), x)  # needs...
→sage.symbolic
-1/(x^2 - 1)
```

```python
>>> float(acoth(Integer(2)))  # needs sage.symbolic
0.5493061443340549
```

```python
>>> float(acoth(Integer(2)).n(Integer(53)))  # Correct result to 53 bits  # needs sage.rings.real_mpfr sage.symbolic
0.5493061443340549
```

```python
>>> float(acoth(Integer(2)).n(Integer(100)))  # Compute 100 bits and then round...
→to 53  # needs sage.rings.real_mpfr sage.symbolic
0.5493061443340549
```

```python
class sage.functions.hyperbolic.Function_arccsch
Bases: GinacFunction
The inverse of the hyperbolic cosecant function.

EXAMPLES:
```
```
Functions, Release 10.4

0.881373587019543

```python
>>> diff(acsch(x), x) # needs sage.symbolic
-1/(sqrt(x^2 + 1)*x)
```

```python
>>> latex(acsch(x)) # needs sage.symbolic
\operatorname{arcsch}\left(x\right)
```

```python
class sage.functions.hyperbolic.Function_arcsech
```

Bases: GinacFunction

The inverse of the hyperbolic secant function.

EXAMPLES:

```python
sage: # needs sage.symbolic
1.31695789692482
sage: asech(1/2)
arcsech(1/2)
```

```python
>>> from sage.all import *
```
class sage.functions.hyperbolic.Function_arcsinh

Bases: GinacFunction

The inverse of the hyperbolic sine function.

EXAMPLES:

sage: asinh
arcsinh
sage: asinh(0.5)       # needs sage.rings.real_mpfr
0.481211825059603
sage: asinh(1/2)       # needs sage.symbolic
arcsinh(1/2)

sage: asinh(1 + I*1.0)  # needs sage.symbolic
1.06127506190504 + 0.666239432492515*I

To prevent automatic evaluation use the hold argument:

sage: asinh(-2, hold=True)  # needs sage.symbolic
arcsinh(-2)

To then evaluate again, use the unhold method:

sage: asinh(-Integer(2), hold=True).unhold()  # needs sage.symbolic
-arcsinh(2)

1.3. Hyperbolic functions
conjugate(asinh(x)) == asinh(conjugate(x)) unless on the branch cuts which run along the imaginary axis outside the interval \([-I, +I]\).

```
sage: # needs sage.symbolic
sage: conjugate(asinh(x))
conjugate(arcsinh(x))
sage: var('y', domain='positive')
y
sage: conjugate(asinh(y))
arcsinh(y)
sage: conjugate(asinh(y+I))
conjugate(arcsinh(y + I))
sage: conjugate(asinh(1/16))
arcsinh(1/16)
sage: conjugate(asinh(I/2))
arcsinh(-1/2*I)
sage: conjugate(asinh(2*I))
conjugate(arcsinh(2*I))
```

```python
globals()
>>> from sage.all import *
>>> # needs sage.symbolic
>>> conjugate(asinh(x))
conjugate(arcsinh(x))
>>> var('y', domain='positive')
y
>>> conjugate(asinh(y))
arcsinh(y)
>>> conjugate(asinh(y+I))
conjugate(arcsinh(y + I))
>>> conjugate(asinh(Integer(1)/Integer(16)))
arcsinh(1/16)
>>> conjugate(asinh(I/Integer(2)))
arcsinh(-1/2*I)
>>> conjugate(asinh(Integer(2)*I))
conjugate(arcsinh(2*I))
```

```
class sage.functions.hyperbolic.Function_arctanh

Bases: GinacFunction

The inverse of the hyperbolic tangent function.

EXAMPLES:

```
sage: atanh(0.5)  # needs sage.rings.real_mpfr
0.549306144334055
sage: atanh(1/2)  # needs sage.symbolic
1/2*log(3)
sage: atanh(1 + I*1.0)  # needs sage.symbolic
0.402359478108525 + 1.01722196789785*I
```
>>> from sage.all import *
>>> atanh(RealNumber('0.5'))
˓→ # needs sage.rings.real_mpfr
0.549306144334055
>>> atanh(Integer(1)/Integer(2))
˓→ # needs sage.symbolic
1/2*log(3)
>>> atanh(Integer(1) + I*RealNumber('1.0'))
˓→ # needs sage.symbolic
0.402359478108525 + 1.01722196789785*I

To prevent automatic evaluation use the hold argument:

sage: atanh(-1/2, hold=True)
˓→ needs sage.symbolic
arctanh(-1/2)

To then evaluate again, use the unhold method:

sage: atanh(-1/2, hold=True).unhold()
˓→ needs sage.symbolic
-1/2*log(3)

conjugate(arctanh(x)) == arctanh(conjugate(x)) unless on the branch cuts which run along the real axis outside the interval [-1, +1].

sage: conjugate(arctanh(x)) == arctanh(conjugate(x))

(continues on next page)
>>> var('y', domain='positive')
y
>>> conjugate(atanh(y))
conjugate(arctanh(y))
>>> conjugate(atanh(y + I))
conjugate(arctanh(y + I))
>>> conjugate(atanh(Integer(1)/Integer(16)))
1/2*log(17/15)
>>> conjugate(atanh(I/Integer(2)))
arctanh(-1/2*I)
>>> conjugate(atanh(-Integer(2)*I))
arctanh(2*I)

class sage.functions.hyperbolic.Function_cosh

Bases: GinacFunction

The hyperbolic cosine function.

EXAMPLES:

sage: cosh(3.1415)  
˓→ needs sage.rings.real_mpfr
11.5908832931176

sage: # needs sage.symbolic
sage: cosh(pi)
cosh(pi)
sage: float(cosh(pi))
11.591953275521519
sage: RR(cosh(1/2))
1.12762596520638
sage: latex(cosh(x))
\cosh\left(x\right)
sage: cosh(x)._sympy_()  
˓→ needs sympy
cosh(x)

from sage.all import *

sage: cosh(RealNumber('3.1415'))  
˓→ # needs sage.rings.real_mpfr
11.5908832931176

sage: # needs sage.symbolic
sage: cosh(pi)
cosh(pi)
sage: float(cosh(pi))
11.591953275521519
sage: RR(cosh(Integer(1)/Integer(2)))
1.12762596520638
sage: latex(cosh(x))
\cosh\left(x\right)
sage: cosh(x)._sympy_()  
˓→ needs sympy
cosh(x)

To prevent automatic evaluation, use the hold parameter:
Functions, Release 10.4

sage: cosh(arcsinh(x), hold=True)  # needs sage.symbolic
cosh(arcsinh(x))

>>> from sage.all import *
>>> cosh(arcsinh(x), hold=True)  # needs sage.symbolic
sqrt(x^2 + 1)

To then evaluate again, use the unhold method:

sage: cosh(arcsinh(x), hold=True).unhold()  # needs sage.symbolic
sqrt(x^2 + 1)

class sage.functions.hyperbolic.Function_coth

Bases: GinacFunction

The hyperbolic cotangent function.

EXAMPLES:

sage: coth(3.1415)  # needs sage.rings.real_mpfr
1.0037418731973213

sage: coth(complex(1, 2))  # abs tol 1e-15  # needs sage.rings.complex_double
(0.8213297974938518+0.17138361290918508j)

sage: # needs sage.symbolic
sage: coth(pi)
coth(pi)

sage: # needs sage.symbolic
sage: coth(pi)
coth(pi)

sage: # needs sage.symbolic
sage: coth(0)
Infinity

sage: # needs sage.symbolic
sage: coth(pi*I)
Infinity

sage: # needs sage.symbolic
sage: coth(pi*I/2)
0

sage: # needs sage.symbolic
sage: coth(7*pi*I/2)
0

sage: # needs sage.symbolic
sage: coth(8*pi*I/2)
Infinity

sage: # needs sage.symbolic
sage: coth(7.*pi/2)
-I*cot(3.50000000000000*pi)

sage: float(coth(pi))
1.0037418731973213

sage: RR(coth(pi))
1.00374187319732

(continues on next page)
sage: diff(coth(x), x)
-1/sinh(x)^2
sage: latex(coth(x))
\coth\left(x\right)
sage: coth(x)._sympy_()
# needs sympy
coth(x)

>>> from sage.all import *
>>> coth(RealNumber('3.1415'))
# needs sage.rings.real_mpfr
1.00374256795520
>>> coth(complex(Integer(1), Integer(2)))  # abs tol 1e-15
# needs sage.rings.complex_double
(0.8213297974938518+0.17138361290918508j)

>>> # needs sage.symbolic
>>> coth(pi)
coth(pi)
>>> coth(Integer(0))
Infinity
>>> coth(pi*I)
Infinity
>>> coth(pi*I/Integer(2))
0
>>> coth(Integer(7)*pi*I/Integer(2))
0
>>> coth(Integer(8)*pi*I/Integer(2))
Infinity
>>> coth(RealNumber('7.')*pi*I/Integer(2))
-I*cot(3.50000000000000*pi)
>>> float(coth(pi))
1.0037418731973213
>>> RR(coth(pi))
1.00374187319732

>>> # needs sage.symbolic
>>> bool(diff(coth(x), x) == diff(Integer(1)/tanh(x), x))
True

class sage.functions.hyperbolic.Function_csch
Bases: GinacFunction

The hyperbolic cosecant function.

EXAMPLES:

sage: csch(3.1415)
# needs sage.rings.real_mpfr
0.0865975907592133

(continues on next page)
sage: # needs sage.symbolic
csch(pi)
csch(pi)
sage: float(csch(pi))
0.0865895375300469...
sage: RR(csch(pi))
0.0865895375300470
sage: csch(0)
Infinity
sage: csch(pi*I)
Infinity
sage: csch(pi*I/2)
-I
sage: csch(7*pi*I/2)
I
sage: csch(7.*pi*I/2)
-I*csc(3.50000000000000*pi)

sage: # needs sage.symbolic
sage: bool(diff(csch(x), x) == diff(1/sinh(x), x))
True
sage: diff(csch(x), x)
-coth(x)*csch(x)
sage: latex(csch(x))
\operatorname{csch}\left(x\right)
sage: csch(x)._sympy_()
# needs sympy

>>> from sage.all import *
>>> csch(RealNumber('3.1415'))
... # needs sage.rings.real_mpfr
0.0865975907592133

>>> # needs sage.symbolic
csch(pi)
csch(pi)
>>> float(csch(pi))
0.0865895375300469...
>>> RR(csch(pi))
0.0865895375300470
>>> csch(Integer(0))
Infinity
>>> csch(pi*I)
Infinity
>>> csch(pi*I/Integer(2))
-I
>>> csch(Integer(7)*pi*I/Integer(2))
I
>>> csch(RealNumber('7.1415')*pi*I/Integer(2))
-I*csc(3.50000000000000*pi)

>>> # needs sage.symbolic
>>> bool(diff(csch(x), x) == diff(Integer(1)/sinh(x), x))
True

(continues on next page)
class sage.functions.hyperbolic.Function_sech

    Bases: GinacFunction

    The hyperbolic secant function.

    EXAMPLES:

    sage: sech(3.1415)  # needs sage.rings.real_mpfr
    0.0862747018248192

    sage: # needs sage.symbolic
    sage: sech(pi)
    sech(pi)
    sage: float(sech(pi))
    0.0862667383340544...
    sage: RR(sech(pi))
    0.0862667383340544
    sage: sech(0)
    1
    sage: sech(pi*I)
    -1
    sage: sech(pi*I/2)
    Infinity
    sage: sech(7*pi*I/2)
    Infinity
    sage: sech(8*pi*I/2)
    1
    sage: sech(8.*pi*I/2)
    sec(4.00000000000000*pi)

    sage: # needs sage.symbolic
    sage: bool(diff(sech(x), x) == diff(1/cosh(x), x))
    True
    sage: diff(sech(x), x)
    -sech(x)*tanh(x)
    sage: latex(sech(x))
    \operatorname{sech}\left(x\right)
    sage: sech(x)._sympy_()  # needs sympy
    #...
Functions, Release 10.4

(continued from previous page)

```python
sech(pi)
>>> float(sech(pi))
0.0862667383340544...
>>> RR(sech(pi))
0.0862667383340544
>>> sech(Integer(0))
1
>>> sech(pi*I)
-1
>>> sech(pi*I/Integer(2))
Infinity
>>> sech(Integer(7)*pi*I/Integer(2))
Infinity
>>> sech(Integer(8)*pi*I/Integer(2))
1
>>> sech(RealNumber('8.')*pi*I/Integer(2))
sec(4.00000000000000*pi)

# needs sage.symbolic
>>> bool(diff(sech(x), x) == diff(Integer(1)/cosh(x), x))
True
>>> diff(sech(x), x)
-sech(x)*tanh(x)
>>> latex(sech(x))
\operatorname{sech\left(x\right)}
>>> sech(x)._sympy_()
# needs sympy
sech(x)
```

class sage.functions.hyperbolic.Function_sech

Bases: GinacFunction

The hyperbolicsinefunction.

EXAMPLES:

```python
sage: sinh(3.1415)  # needs sage.rings.real_mpfr
11.5476653707437
sage: # needs sage.symbolic
sage: sinh(pi)
\sinh\left(\pi\right)
sage: sinh(pi)
sinh(pi)
```

```python
>>> from sage.all import *
>>> sinh(RealNumber('3.1415'))  # needs sage.rings.real_mpfr
11.5476653707437
```

(continues on next page)
>>> # needs sage.symbolic
>>> sinh(pi)
sinh(pi)
>>> float(sinh(pi))
11.54873935725774...
>>> RR(sinh(pi))
11.5487393572577
>>> latex(sinh(x))\
\sinh\left(x\right)
>>> sinh(x)._sympy_() # needs...
˓→sympy
sinh(x)

To prevent automatic evaluation, use the hold parameter:

```python
sage: sinh(arccosh(x), hold=True) # needs sage.symbolic
˓→sinh(arccosh(x))
```

To then evaluate again, use the unhold method:

```python
sage: sinh(arccosh(x), hold=True).unhold() # needs sage.symbolic
˓→sqrt(x + 1)*sqrt(x - 1)
```

```python
>>> from sage.all import *
>>> sinh(arccosh(x), hold=True).unhold() # needs sage.symbolic
˓→sqrt(x + 1)*sqrt(x - 1)
```

class sage.functions.hyperbolic.Function_tanh

Bases: GinacFunction

The hyperbolic tangent function.

EXAMPLES:

```python
sage: tanh(3.1415) # needs sage.rings.real_mpfr
˓→0.996271386633702
sage: tan(3.1415/4) # needs sage.rings.real_mpfr
˓→0.999953674278156
sage: # needs sage.symbolic
sage: tanh(pi)
tanh(pi)
sage: float(tanh(pi))
˓→0.99627207622075
sage: tanh(pi/4)
˓→0.999953674278156
```
tanh(1/4*pi)
sage: RR(tanh(1/2))
0.462117157260010

>>> from sage.all import *
>>> tanh(RealNumber('3.1415'))
˓→ # needs sage.rings.real_mpfr
0.996271386633702
>>> tanh(RealNumber('3.1415')/Integer(4))
˓→ # needs sage.rings.real_mpfr
0.99953674278156

>>> # needs sage.symbolic
>>> tanh(pi)
tanh(pi)
>>> float(tanh(pi))
0.99627207622075
>>> tanh(pi/Integer(4))
tanh(1/4*pi)
>>> RR(tanh(Integer(1)/Integer(2)))
0.462117157260010

sage: CC(tanh(pi + I*e))
˓→ # needs sage.rings.real_mpfr sage.symbolic
0.997524731976164 - 0.00279068768100315*I
sage: ComplexField(100)(tanh(pi + I*e))
˓→ # needs sage.rings.real_mpfr sage.symbolic
0.99752473197616361034204366446 - 0.0027906876810031453884245163923*I
sage: CDF(tanh(pi + I*e))
˓→ # rel tol 2e-15
0.9975247319761636 - 0.002790687681003147*I

To prevent automatic evaluation, use the hold parameter:

sage: tanh(arcsinh(x), hold=True)
˓→ # needs sage.symbolic
tanh(arcsinh(x))

To then evaluate again, use the unhold method:
1.4 Number-theoretic functions

class sage.functions.transcendental.DickmanRho

Dickman’s function is the continuous function satisfying the differential equation

\[ x \rho'(x) + \rho(x - 1) = 0 \]

with initial conditions \( \rho(x) = 1 \) for \( 0 \leq x \leq 1 \). It is useful in estimating the frequency of smooth numbers as asymptotically

\[ \Psi(a, a^{1/s}) \sim a \rho(s) \]

where \( \Psi(a, b) \) is the number of \( b \)-smooth numbers less than \( a \).

ALGORITHM:
Dickman’s function is analytic on the interval \([n, n + 1]\) for each integer \( n \). To evaluate at \( n + t, 0 \leq t < 1 \), a power series is recursively computed about \( n + 1/2 \) using the differential equation stated above. As high precision arithmetic may be needed for intermediate results the computed series are cached for later use.

Simple explicit formulas are used for the intervals \([0,1]\) and \([1,2]\).

EXAMPLES:
AUTHORS:

• Robert Bradshaw (2008-09)

REFERENCES:


approximate \((x, parent=None)\)

Approximate using de Bruijn’s formula

\[
\rho(x) \sim \frac{\exp(-x\xi + Ei(\xi))}{\sqrt{2\pi x\xi}}
\]

which is asymptotically equal to Dickman’s function, and is much faster to compute.

REFERENCES:


EXAMPLES:

```
sage: dickman_rho.approximate(10)  # needs sage.rings.real_mpfr
2.41739196365564e-11
sage: dickman_rho(10)  # needs sage.symbolic
2.77017183772596e-11
sage: dickman_rho.approximate(1000)  # needs sage.rings.real_mpfr
4.32938809066403e-3464
```

```
>>> from sage.all import *
>>> dickman_rho.approximate(Integer(10))  # needs sage.rings.real_mpfr
2.41739196365564e-11
>>> dickman_rho(Integer(10))  # needs sage.symbolic
2.77017183772596e-11
>>> dickman_rho.approximate(Integer(1000))  # needs sage.rings.real_mpfr
4.32938809066403e-3464
```

```
power_series \((n, abs_prec)\)

This function returns the power series about \(n + 1/2\) used to evaluate Dickman’s function. It is scaled such that the interval \([n, n + 1]\) corresponds to \(x\) in \([-1, 1]\).

INPUT:

• \(n\) – the lower endpoint of the interval for which this power series holds
• \(abs\_prec\) – the absolute precision of the resulting power series

EXAMPLES:
sage: # needs sage.rings.real_mpfr
sage: f = dickman_rho.power_series(2, 20); f
-9.9376e-8*x^11 + 3.7722e-7*x^10 - 1.4684e-6*x^9 + 5.8783e-6*x^8
  - 0.000024259*x^7 + 0.00010341*x^6 - 0.00045583*x^5 + 0.0020773*x^4
  - 0.0097336*x^3 + 0.045224*x^2 - 0.11891*x + 0.13032
sage: f(-1), f(0), f(1)
(0.30685, 0.13032, 0.048608)
sage: dickman_rho(2), dickman_rho(2.5), dickman_rho(3)
(0.306852819440055, 0.130319561832251, 0.0486083882911316)

>>> from sage.all import *
>>> # needs sage.rings.real_mpfr
>>> f = dickman_rho.power_series(Integer(2), Integer(20)); f
-9.9376e-8*x^11 + 3.7722e-7*x^10 - 1.4684e-6*x^9 + 5.8783e-6*x^8
  - 0.000024259*x^7 + 0.00010341*x^6 - 0.00045583*x^5 + 0.0020773*x^4
  - 0.0097336*x^3 + 0.045224*x^2 - 0.11891*x + 0.13032
>>> f(-Integer(1)), f(Integer(0)), f(Integer(1))
(0.30685, 0.13032, 0.048608)
>>> dickman_rho(Integer(2)), dickman_rho(RealNumber(2.5)), dickman_
  →rho(Integer(3))
(0.306852819440055, 0.130319561832251, 0.0486083882911316)

class sage.functions.transcendental.Function_HurwitzZeta
Bases: BuiltinFunction

class sage.functions.transcendental.Function_stieltjes
Bases: GinacFunction

Stieltjes constant of index n.

stieltjes(0) is identical to the Euler-Mascheroni constant (sage.symbolic.constants.EulerGamma). The Stieltjes constants are used in the series expansions of \( \zeta(s) \).

INPUT:

- n – non-negative integer

EXAMPLES:

sage: # needs sage.symbolic
sage: n = var("n")
sage: stieltjes(n)
stieltjes(n)
sage: stieltjes(0)
euler_gamma
sage: stieltjes(2)
stieltjes(2)
sage: stieltjes(int(2))
stieltjes(2)
sage: stieltjes(2).n(100)
-0.0096903631928723184845303860352
sage: RR = RealField(200)
# needs sage.rings.real_mpfr
sage: stieltjes(RR(2))
# needs sage.rings.real_mpfr
-0.0096903631928723184845303860352125293590658061013407498807014
It is possible to use the hold argument to prevent automatic evaluation:

```python
sage: stieltjes(0, hold=True)  # needs sage.symbolic
stieltjes(0)
```

```python
sage: latex(stieltjes(n))
\gamma_n
```

```python
from sage.all import *

# needs sage.symbolic
stieltjes(n)

# needs sage.symbolic
stieltjes(Integer(0))
euler_gamma

# needs sage.symbolic
stieltjes(Integer(2))

# needs sage.symbolic
stieltjes(int(Integer(2)))

# needs sage.symbolic
stieltjes(Integer(2)).n(Integer(100))
-0.0096903631928723184845303860352125293590658061013407498807014

# needs sage.symbolic
RR = RealField(Integer(200))

# needs sage.rings.real_mpfr

# needs sage.symbolic
RR(Integer(2))
-0.0096903631928723184845303860352125293590658061013407498807014

# needs sage.symbolic
stieltjes(0, hold=True)  # needs sage.symbolic
stieltjes(0)

# needs sage.symbolic
stieltjes(Integer(0), hold=True)  # needs sage.symbolic
stieltjes(0)

# needs sage.symbolic
stieltjes(x).subs(x==0)  # needs sage.symbolic
euler_gamma
```
class sage.functions.transcendental.Function_zeta
Bases: GinacFunction

Riemann zeta function at s with s a real or complex number.

INPUT:

• s – real or complex number

If s is a real number, the computation is done using the MPFR library. When the input is not real, the computation is done using the PARI C library.

EXAMPLES:

```python
sage: RR = RealField(200)  # needs sage.rings.real_mpfr
sage: zeta(RR(2))          # needs sage.rings.real_mpfr
1.6449340668482264364724151666460251892189499012067984377356

sage: # needs sage.symbolic
sage: zeta(x)
zeta(x)

sage: zeta(2)
1/6*pi^2

sage: zeta(2.)
1.64493406684823

sage: zeta(I)
zeta(I)

sage: zeta(I).n()
0.00330022368532410 - 0.418155449141322*I

sage: zeta(sqrt(2))
zeta(sqrt(2))

sage: zeta(sqrt(2)).n()  # rel tol 1e-10
3.02073767948603

from sage.all import *

RR = RealField(Integer(200))  # needs sage.rings.real_mpfr
zeta(RR(Integer(2)))  # needs sage.rings.real_mpfr
1.6449340668482264364724151666460251892189499012067984377356

# needs sage.symbolic
zeta(x)
zeta(2)
1/6*pi^2
zeta(2.)
1.64493406684823
zeta(I)
zeta(I)
0.00330022368532410 - 0.418155449141322*I
zeta(sqrt(2))
zeta(sqrt(2))
zeta(sqrt(2)).n()  # rel tol 1e-10
3.02073767948603
```

It is possible to use the `hold` argument to prevent automatic evaluation:
sage: zeta(2, hold=True)  # needs sage.symbolic
zeta(2)

>>> from sage.all import *
>>> zeta(Integer(2), hold=True)  # needs sage.symbolic
zeta(2)

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():

sage: a = zeta(2, hold=True); a.simplify()  # needs sage.symbolic
1/6*pi^2

>>> from sage.all import *
>>> a = zeta(Integer(2), hold=True); a.simplify()  # needs sage.symbolic
1/6*pi^2

The Laurent expansion of $\zeta(s)$ at $s = 1$ is implemented by means of the Stieltjes constants:

sage: s = SR('s')  # needs sage.symbolic
sage: zeta(s).series(s==1, 2)  # needs sage.symbolic
1*(s - 1)^(-1) + euler_gamma + (-stieltjes(1))*(s - 1) + Order((s - 1)^2)

>>> from sage.all import *
>>> s = SR('s')  # needs sage.symbolic
>>> zeta(s).series(s==Integer(1), Integer(2))  # needs sage.symbolic
1*(s - 1)^(-1) + euler_gamma + (-stieltjes(1))*(s - 1) + Order((s - 1)^2)

Generally, the Stieltjes constants occur in the Laurent expansion of $\zeta$-type singularities:

sage: zeta(2*s/(s+1)).series(s==1, 2)  # needs sage.symbolic
2*(s - 1)^(-1) + (euler_gamma + 1) + (-1/2*stieltjes(1))*(s - 1) + Order((s - 1)^2)

>>> from sage.all import *
>>> zeta(Integer(2)*s/(s+Integer(1))).series(s==Integer(1), Integer(2))  # needs sage.symbolic
2*(s - 1)^(-1) + (euler_gamma + 1) + (-1/2*stieltjes(1))*(s - 1) + Order((s - 1)^2)

class sage.functions.transcendental.Function_zetaderiv

Bases: GinacFunction

Derivatives of the Riemann zeta function.

EXAMPLES:
sage: # needs sage.symbolic
sage: zetaderiv(1, x)
zetaderiv(1, x)
sage: zetaderiv(1, x).diff(x)
zetaderiv(2, x)
sage: var('n')
n
sage: zetaderiv(n, x)
zetaderiv(n, x)
sage: zetaderiv(1, 4).n()
-0.0689112658961254
sage: import mpmath; mpmath.diff(lambda x: mpmath.zeta(x), 4)  # needs mpmath
mpf('-0.068911265896125382')

>>> from sage.all import *
>>> # needs sage.symbolic
>>> zetaderiv(Integer(1), x)
zetaderiv(1, x)
>>> zetaderiv(Integer(1), x).diff(x)
zetaderiv(2, x)
>>> var('n')
n
>>> zetaderiv(n, x)
zetaderiv(n, x)
>>> zetaderiv(Integer(1), Integer(4)).n()
-0.0689112658961254
>>> import mpmath; mpmath.diff(lambda x: mpmath.zeta(x), Integer(4))  # needs mpmath
mpf('-0.068911265896125382')

sage.functions.transcendental.hurwitz_zeta(s, x, **kwargs)
The Hurwitz zeta function \( \zeta(s, x) \), where \( s \) and \( x \) are complex.

The Hurwitz zeta function is one of the many zeta functions. It is defined as

\[
\zeta(s, x) = \sum_{k=0}^{\infty} (k + x)^{-s}.
\]

When \( x = 1 \), this coincides with Riemann’s zeta function. The Dirichlet L-functions may be expressed as linear combinations of Hurwitz zeta functions.

EXAMPLES:

Symbolic evaluations:
Functions, Release 10.4

>>> from sage.all import *

>>> # needs sage.symbolic

>>> hurwitz_zeta(x, Integer(1))

zeta(x)

>>> hurwitz_zeta(Integer(4), Integer(3))

1/90*pi^4 - 17/16

>>> hurwitz_zeta(-Integer(4), x)

-1/5*x^5 + 1/2*x^4 - 1/3*x^3 + 1/30*x

>>> hurwitz_zeta(Integer(7), -Integer(1)/Integer(2))

127*zeta(7) - 128

>>> hurwitz_zeta(-Integer(3), Integer(1))

1/120

Numerical evaluations:

sage: hurwitz_zeta(3, 1/2).n()  # needs mpmath
8.41439832211716

sage: hurwitz_zeta(11/10, 1/2).n()  # needs sage.symbolic
12.1038134956837

sage: hurwitz_zeta(3, x).series(x, 60).subs(x=0.5).n()  # needs sage.symbolic
8.41439832211716

sage: hurwitz_zeta(3, 0.5)  # needs mpmath
8.41439832211716

>>> from sage.all import *

>>> hurwitz_zeta(Integer(3), Integer(1)/Integer(2)).n()  # needs mpmath
8.41439832211716

>>> hurwitz_zeta(Integer(11)/Integer(10), Integer(1)/Integer(2)).n()  # needs sage.symbolic
12.1038134956837

>>> hurwitz_zeta(Integer(3), x).series(x, Integer(60)).subs(x=RealNumber('0.5')).n()  # needs sage.symbolic
8.41439832211716

>>> hurwitz_zeta(Integer(3), RealNumber('0.5'))  # needs mpmath
8.41439832211716

REFERENCES:

- Wikipedia article Hurwitz_zeta_function

sage.functions.transcendental.zeta_symmetric(s)

Completed function $\zeta(s)$ that satisfies $\xi(s) = \xi(1 - s)$ and has zeros at the same points as the Riemann zeta function.

INPUT:

- $s$ – real or complex number

If $s$ is a real number the computation is done using the MPFR library. When the input is not real, the computation is done using the PARI C library.

More precisely,

$$\xi(s) = \gamma(s/2 + 1) * (s - 1) * \pi^{-s/2} * \zeta(s).$$

1.4. Number-theoretic functions 59
EXAMPLES:

```python
sage: # needs sage.rings.real_mpfr
sage: RR = RealField(200)
0.49758041465112690357779107525638385212657443284080589766062
sage: zeta_symmetric(RR(0.7))
0.49758041465112690357779107525638385212657443284080589766062
sage: # needs sage.libs.pari sage.rings.real_mpfr
sage: zeta_symmetric(1 - 0.7)
0.49758041465112690357779107525638385212657443284080589766062
sage: C.<i> = ComplexField()
0.0000868931282620101 + 7.11507675693612e-20*I
sage: zeta_symmetric(0.5 + i*14.0)
0.000201294444235258 + 1.4907798716757e-19*I
sage: zeta_symmetric(0.5 + i*14.1)
0.0000489893483255687 + 4.40457132572236e-20*I
sage: zeta_symmetric(0.5 + i*14.2)
-0.0000868931282620101 + 7.11507675693612e-20*I
```

```python
>>> from sage.all import *

>>> # needs sage.rings.real_mpfr
>>> RR = RealField(Integer(200))
0.49758041465112690357779107525638385212657443284080589766062

>>> # needs sage.libs.pari sage.rings.real_mpfr
>>> zeta_symmetric(RR(RealNumber('0.7')))
0.49758041465112690357779107525638385212657443284080589766062

>>> zeta_symmetric(Integer(1) - RealNumber('0.7'))
0.49758041465112690357779107525638385212657443284080589766062

>>> C = ComplexField(names=('i',)); (i,) = C._first_ngens(1)
0.0000868931282620101 + 7.11507675693612e-20*I

>>> zeta_symmetric(RealNumber('0.5') + i*RealNumber('14.0'))
0.000201294444235258 + 1.4907798716757e-19*I

>>> zeta_symmetric(RealNumber('0.5') + i*RealNumber('14.1'))
0.0000489893483255687 + 4.40457132572236e-20*I

>>> zeta_symmetric(RealNumber('0.5') + i*RealNumber('14.2'))
-0.0000868931282620101 + 7.11507675693612e-20*I
```

REFERENCE:

- I copied the definition of \(x\) from [http://web.viu.ca/pughg/RiemannZeta/RiemannZetaLong.html](http://web.viu.ca/pughg/RiemannZeta/RiemannZetaLong.html)

1.5 Error functions

This module provides symbolic error functions. These functions use the mpmathlibrary for numerical evaluation and Maxima, Pynac for symbols.

The main objects which are exported from this module are:

- `erf` – The error function
- `erfc` – The complementary error function
- `erfi` – The imaginary error function
- `erfinv` – The inverse error function
- `fresnel_sin` – The Fresnel integral \(S(x)\)
• **fresnel_cos** – The Fresnel integral \( C(x) \)

**AUTHORS:**

• Original authors erf/error_fcn (c) 2006-2014: Karl-Dieter Crisman, Benjamin Jones, Mike Hansen, William Stein, Burcin Erocal, Jeroen Demeyer, W. D. Joyner, R. Andrew Ohana

• Reorganisation in new file, addition of erfi/erfinv/erfc (c) 2016: Ralf Stephan

• Fresnel integrals (c) 2017 Marcelo Forets

**REFERENCES:**

• [DLMF-Error]

• [WP-Error]

class sage.functions.error.Function_Fresnel_cos

Bases: BuiltinFunction

The cosine Fresnel integral.

It is defined by the integral

\[
C(x) = \int_0^x \cos \left( \frac{\pi t^2}{2} \right) dt
\]

for real \( x \). Using power series expansions, it can be extended to the domain of complex numbers. See the Wikipedia article Fresnel_integral.

**INPUT:**

• \( x \) – the argument of the function

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: fresnel_cos(0)
0
sage: fresnel_cos(x).subs(x==0)
0
sage: x = var('x')

sage: fresnel_cos(1).n(100)  # needs sympy
fresnelc(x)

0.77989340037682282947420641365
```

```python
>>> from sage.all import *

>>> # needs sage.symbolic
>>> fresnel_cos(Integer(0))
0

>>> fresnel_cos(x).subs(x==Integer(0))
0

>>> x = var('x')

>>> fresnel_cos(Integer(1)).n(Integer(100))
0.77989340037682282282947420641365

>>> fresnel_cos(x)._sympy_()  # needs sympy
fresnelc(x)
```

1.5. Error functions
class sage.functions.error.Function_Fresnel_sin

Bases: BuiltinFunction

The sine Fresnel integral.

It is defined by the integral

\[ S(x) = \int_0^x \sin \left( \frac{\pi t^2}{2} \right) \, dt \]

for real \( x \). Using power series expansions, it can be extended to the domain of complex numbers. See the Wikipedia article Fresnel_integral.

INPUT:

- \( x \) – the argument of the function

EXAMPLES:

```
sage: # needs sage.symbolic
sage: fresnel_sin(0)
0
sage: fresnel_sin(x).subs(x==0)
0
sage: x = var('x')
sage: fresnel_sin(1).n(100)
0.43825914739035476607675669662
sage: fresnel_sin(x)._sympy_()
# needs sympy
fresnels(x)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> fresnel_sin(Integer(0))
0
>>> fresnel_sin(x).subs(x==Integer(0))
0
>>> x = var('x')
>>> fresnel_sin(Integer(1)).n(Integer(100))
0.43825914739035476607675669662
>>> fresnel_sin(x)._sympy_()
# needs...
sympy
fresnels(x)
```

class sage.functions.error.Function_erf

Bases: BuiltinFunction

The error function.

The error function is defined for real values as

\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} \, dt. \]

This function is also defined for complex values, via analytic continuation.

EXAMPLES:

We can evaluate numerically:
sage: erf(2)  # needs sage.symbolic
erf(2)
sage: erf(2).n()  # needs sage.symbolic
0.99532265018953
sage: erf(2).n(100)  # needs sage.symbolic
0.9953226501895273416206925637
sage: erf(ComplexField(100)(2+3j))  # needs sage.rings.real_mpfr
-20.829461427614568389103088452 + 8.6873182714701631444280787545*I

Basic symbolic properties are handled by Sage and Maxima:

sage: x = var("x")  # needs sage.symbolic
sage: diff(erf(x),x)  # needs sage.symbolic
2*e^(-x^2)/sqrt(pi)

ALGORITHM:
Sage implements numerical evaluation of the error function via the \texttt{erf()} function from mpmath. Symbolics are handled by Sage and Maxima.

REFERENCES:
- Wikipedia article Error_function
class sage.functions.error.Function_erfc

Bases: BuiltinFunction

The complementary error function.

The complementary error function is defined by

\[ \frac{2}{\sqrt{\pi}} \int_{1}^{\infty} e^{-x^2} \, dx. \]

EXAMPLES:

```
sage: erfc(6)  # needs sage.symbolic
erfc(6)
sage: erfc(6).n()  # needs sage.symbolic
2.15197367124989e-17
sage: erfc(RealField(100)(1/2))  # needs sage.rings.real_mpfr
0.47950012218695346231725334611
sage: 1 - erfc(0.5)  # needs mpmath
0.520499877813047
sage: erf(0.5)  # needs mpmath
0.520499877813047
```

```python
>>> from sage.all import *
>>> erfc(Integer(6))  # needs sage.symbolic
erfc(6)
>>> erfc(Integer(6)).n()  # needs sage.symbolic
2.15197367124989e-17
>>> erfc(RealField(Integer(100))(Integer(1)/Integer(2)))  # needs sage.rings.real_mpfr
0.47950012218695346231725334611
>>> Integer(1) - erfc(RealNumber('0.5'))  # needs mpmath
0.520499877813047
>>> erf(RealNumber('0.5'))  # needs mpmath
0.520499877813047
```

class sage.functions.error.Function_erfi

Bases: BuiltinFunction

The imaginary error function.

The imaginary error function is defined by

\[ \text{erfi}(x) = -i \text{erf}(ix). \]

class sage.functions.error.Function_erfinv

Bases: BuiltinFunction
The inverse error function.

The inverse error function is defined by:

\[ \text{erfinv}(x) = \text{erf}^{-1}(x). \]

## 1.6 Piecewise functions

This module implements piecewise functions in a single variable. See `sage.sets.real_set` for more information about how to construct subsets of the real line for the domains.

**EXAMPLES:**

```python
sage: f = piecewise([[(0,1), x^3], [(-1,0), -x^2]]); f
piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)
sage: 2*f
2*piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)
sage: f(x=1/2)
1/8
sage: plot(f)  # not tested
```

```python
>>> from sage.all import *

>>> f = piecewise([[(Integer(0),Integer(1)), x**Integer(3)], [(-Integer(1),
-Integer(0)), -x**Integer(2)]]); f
piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)

>>> Integer(2)*f
2*piecewise(x|-->x^3 on (0, 1), x|-->-x^2 on [-1, 0]; x)

>>> f(x=Integer(1)/Integer(2))
1/8

>>> plot(f)  # not tested
```

**Todo:** Implement max/min location and values.

**AUTHORS:**

- David Joyner (2006-04): initial version
- David Joyner (2006-09): added `__eq__`, `extend_by_zero_to`, `unextend`, `convolution`, `trapezoid`, `trapezoid_integral_approximation`, `riemann_sum`, `riemann_sum_integral_approximation`, `tangent_line` fixed bugs in `__mul__`, `__add__`
- David Joyner (2007-03): adding Hann filter for FS, added general FS filter methods for computing and plotting, added options to plotting of FS (e.g., specifying rgb values are now allowed). Fixed bug in documentation reported by Pablo De Napoli.
- David Joyner (2007-09): bug fixes due to behaviour of `SymbolicArithmetic`
- David Joyner (2008-04): fixed doctest bugs reported by J Morrow; added support for Laplace transform of functions with infinite support.
- David Joyner (2008-07): fixed a left multiplication bug reported by C. Boncelet (by defining `__rmul__ = __mul__`).
- Paul Butler (2009-01): added indefinite integration and `default_variable`
- Volker Braun (2013): Complete rewrite
- Ralf Stephan (2015): Rewrite of `convolution()` and other calculus functions; many doctest adaptations

---

1.6. Piecewise functions 65
Eric Gourgoulhon (2017): Improve documentation and user interface of Fourier series

```python
class sage.functions.piecewise.PiecewiseFunction

Bases: BuiltinFunction

Piecewise function

EXAMPLES:

```python
sage: var('x, y')
(x, y)
sage: f = piecewise([[(0,1), x^2*y], [(-1,0), -x*y^2]], var=x); f
piecewise(x|-->x^2*y on (0, 1), x|-->-x*y^2 on [-1, 0]; x)
sage: f(1/2)
1/4*y
sage: f(-1/2)
1/2*y^2
```
x|→-x + 2 on (1, 2]; x) >>> h = f.convolution(g); h

piecewise(x|→1/2*x^2 on (0, 1],
  x|→-x^2 + 3*x - 3/2 on (1, 2],
  x|→1/2*x^2 - 3*x + 9/2 on (2, 3]; x)

Example 1:

sage: f = piecewise([[0,1], 1], [(1,2), 2], [(2,3), 1]])
sage: g = f.convolution(f)
sage: h = f.convolution(g); h

piecewise(x|→1/2*x^2 on (0, 1],
  x|→2*x^2 - 3*x + 3/2 on (1, 3],
  x|→-2*x^2 + 21*x - 69/2 on (3, 4],
  x|→-5*x^2 + 45*x - 165/2 on (4, 5],
  x|→-2*x^2 + 15*x - 15/2 on (5, 6],
  x|→2*x^2 - 33*x + 273/2 on (6, 8],
  x|→1/2*x^2 - 9*x + 81/2 on (8, 9]; x)

Example 2:

sage: f = piecewise([[-1,1], 1]])
sage: g = piecewise([[0,3], x]])
sage: f.convolution(g)
piecewise(x|→x + 1 on (-1, 1],
  x|→2 on (1, 2],
  x|→x on (2, 3],
  x|→x + 6 on (3, 4],
  x|→-2*x + 10 on (4, 5]; x)

sage: g = piecewise([[0,3], 1], [(3,4), 2]])
sage: f.convolution(g)
piecewise(x|→x + 1 on (-1, 1],
  x|→2 on (1, 2],
  x|→x on (2, 3],
  x|→x + 6 on (3, 4],
  x|→-2*x + 10 on (4, 5]; x)

>>> from sage.all import *

>>> f = piecewise([[(Integer(0),Integer(1)), Integer(1)], [(Integer(1), Integer(2)), Integer(2)], [(Integer(2),Integer(3)), Integer(1)]])

>>> g = f.convolution(f)

>>> h = f.convolution(g); h

piecewise(x|→1/2*x^2 on (0, 1],
  x|→2*x^2 - 3*x + 3/2 on (1, 3],
  x|→-2*x^2 + 21*x - 69/2 on (3, 4],
  x|→-5*x^2 + 45*x - 165/2 on (4, 5],
  x|→-2*x^2 + 15*x - 15/2 on (5, 6],
  x|→2*x^2 - 33*x + 273/2 on (6, 8],
  x|→1/2*x^2 - 9*x + 81/2 on (8, 9]; x)
Check that the bugs raised in Issue #12123 are fixed:

\[
\begin{align*}
sage: & f = \text{piecewise}([[(-2, 2), 2]]) \\
sage: & g = \text{piecewise}([[(-1, 1), 1]]) \\
sage: & f = \text{piecewise}([[(-1, 1), 1]]) \\
sage: & g = \text{piecewise}([[(-1, 1), 1]]) \\
sage: & f = \text{piecewise}([[(-2, 2), 2]]) \\
sage: & g = \text{piecewise}([[(-1, 1), 1]]) \\
sage: & f = \text{piecewise}([[(-2, 2), 2]]) \\
sage: & g = \text{piecewise}([[(-1, 1), 1]]) \\
\end{align*}
\]

\[
\begin{align*}
\text{critical_points (parameters, variable)}
\end{align*}
\]

Return the critical points of this piecewise function.

**EXAMPLES:**

\[
\begin{align*}
sage: & R.<x> = \mathbb{Q}[x] \\
sage: & f1 = x^0 \\
sage: & f2 = 10^x - x^2 \\
sage: & f3 = 3^x+4 - 156^x+3 + 3036^x+2 - 2620^x \\
sage: & f = \text{piecewise}([[0,3], f1, [3,10], f2, [10,20], f3]) \\
sage: & \text{expected} = [5, 12, 13, 14] \\
sage: & \text{all(abs(e-a) < 0.001 for e,a in zip(expected, f.critical_points()))} \\
True
\end{align*}
\]
domain (parameters, variable)

Return the domain

OUTPUT:

The union of the domains of the individual pieces as a RealSet.

EXAMPLES:

```
sage: f = piecewise([[([0,0], sin(x)), ((0,2), cos(x))]]; f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.domain()
[0, 2]
```

domains (parameters, variable)

Return the individual domains

See also expressions ()

OUTPUT:

The collection of domains of the component functions as a tuple of RealSet.

EXAMPLES:

```
sage: f = piecewise([[([0,0], sin(x)), ((0,2), cos(x))]]; f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.domains()
((0), (0, 2))
```

end_points (parameters, variable)

Return a list of all interval endpoints for this function.

EXAMPLES:
Functions, Release 10.4

sage: f1(x) = 1
sage: f2(x) = 1 - x
sage: f3(x) = x^2 - 5
sage: f = piecewise([[(0,1), f1], [(1,2), f2], [(2,3), f3]])
sage: f.end_points()
[0, 1, 2, 3]
sage: f = piecewise([[(0,0), sin(x)], ((0,2), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.end_points()
[0, 2]

>>> from sage.all import *
>>> __tmp__=var("x"); f1 = symbolic_expression(Integer(1)).function(x)
>>> __tmp__=var("x"); f2 = symbolic_expression(Integer(1) - x).function(x)
>>> __tmp__=var("x"); f3 = symbolic_expression(x**Integer(2) - Integer(5)).function(x)
>>> f = piecewise([[(Integer(0),Integer(1)), f1], [(Integer(1), Integer(2)), f2], [(Integer(2),Integer(3)), f3]])
>>> f.end_points()
[0, 1, 2, 3]
>>> f = piecewise([[(Integer(0),Integer(0)), sin(x)], ((Integer(0), Integer(2)), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
>>> f.end_points()
[0, 2]

expression_at (parameters, variable, point)

Return the expression defining the piecewise function at value

INPUT:
• point — a real number.

OUTPUT:
The symbolic expression defining the function value at the given point.

EXAMPLES:

sage: f = piecewise([[(0,0), sin(x)], ((0,2), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.expression_at(0)
sin(x)
sage: f.expression_at(1)
cos(x)
sage: f.expression_at(2)
Traceback (most recent call last):
...
ValueError: point is not in the domain

>>> from sage.all import *
>>> f = piecewise([[(Integer(0),Integer(0)), sin(x)], ((Integer(0), Integer(2)), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
>>> f.expression_at(Integer(0))
sin(x)
>>> f.expression_at(Integer(1))
cos(x)
>>> f.expression_at(Integer(2))
(continues on next page)
expressions (parameters, variable)

Return the individual domains

See also domains().

OUTPUT:

The collection of expressions of the component functions.

EXAMPLES:

```python
sage: f = piecewise([[(0,0], sin(x)), ((0,2), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.expressions()
(sin(x), cos(x))
```

```
>>> from sage.all import *

>>> f = piecewise([[(0,0), sin(x)], ((0,2), cos(x))]); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
```

extension (parameters, variable, extension, extension_domain=None)

Extend the function

INPUT:

- extension – a symbolic expression
- extension_domain – a RealSet or None (default). The domain of the extension. By default,
  the entire complement of the current domain.

EXAMPLES:

```python
sage: f = piecewise([((-1,1), x)]); f
piecewise(x|-->x on (-1, 1); x)
sage: f(3)
Traceback (most recent call last):
  ... ValueError: point 3 is not in the domain
```

```
>>> from sage.all import *

>>> f = piecewise([((-Integer(1),Integer(1)), x)]); f
piecewise(x|-->x on (-1, 1); x)
```

(continues on next page)
fourier_series_cosine_coefficient (parameters, variable, n, L=None)

Return the $n$-th cosine coefficient of the Fourier series of the periodic function $f$ extending the piecewise-defined function $self$.

Given an integer $n \geq 0$, the $n$-th cosine coefficient of the Fourier series of $f$ is defined by

$$a_n = \frac{1}{L} \int_{-L}^{L} f(x) \cos \left( \frac{n\pi x}{L} \right) dx,$$

where $L$ is the half-period of $f$. For $n \geq 1$, $a_n$ is the coefficient of $\cos(n\pi x/L)$ in the Fourier series of $f$, while $a_0$ is twice the coefficient of the constant term $\cos(0x)$, i.e. twice the mean value of $f$ over one period (cf. fourier_series_partial_sum()).

INPUT:

- $n$ – a non-negative integer
- $L$ – (default: None) the half-period of $f$; if none is provided, $L$ is assumed to be the half-width of the domain of $self$

OUTPUT:

- the Fourier coefficient $a_n$, as defined above

EXAMPLES:

A triangle wave function of period 2:

```sage
f = piecewise([((0,1), x), ((1,2), 2 - x)])
f.fourier_series_cosine_coefficient(0)
1
sage: f.fourier_series_cosine_coefficient(3)
-4/9/pi^2
```

```python
>>> from sage.all import *

>>> f = piecewise([((Integer(0),Integer(1)), x), ((Integer(1),Integer(2)), -Integer(2) - x)])

>>> f.fourier_series_cosine_coefficient(Integer(0))
1
>>> f.fourier_series_cosine_coefficient(Integer(3))
-4/9/pi^2
```

If the domain of the piecewise-defined function encompasses more than one period, the half-period must be passed as the second argument; for instance:
sage: f2 = piecewise([[(0,1), x], ((1,2), 2 - x),
....:     ((2,3), x - 2), ((3,4), 2 - (x-2))])
sage: bool(f2.restriction((0,2)) == f)  # f2 extends f on (0,4)
True
sage: f2.fourier_series_cosine_coefficient(3, 1)  # half-period = 1
-4/9/pi^2

>>> from sage.all import *
>>> f2 = piecewise([[(Integer(0),Integer(1)), x], ((Integer(1),
˓→Integer(2)), Integer(2) - x),
....:     ((Integer(2),Integer(3)), x - Integer(2)),
˓→((Integer(3),Integer(4)), Integer(2) - (x-Integer(2)))]
>>> bool(f2.restriction((Integer(0),Integer(2))) == f)  # f2 extends f on...
˓→(0,4)
True
>>> f2.fourier_series_cosine_coefficient(Integer(3), Integer(1))  # half-
˓→period = 1
-4/9/pi^2

The default half-period is 2 and one has:

sage: f2.fourier_series_cosine_coefficient(3)  # half-period = 2
0

>>> from sage.all import *
>>> f2.fourier_series_cosine_coefficient(Integer(3))  # half-period = 2
0

The Fourier coefficient $-\frac{4}{9\pi^2}$ obtained above is actually recovered for $n = 6$:

sage: f2.fourier_series_cosine_coefficient(6)
-4/9/pi^2

>>> from sage.all import *
>>> f2.fourier_series_cosine_coefficient(Integer(6))
-4/9/pi^2

Other examples:

sage: f(x) = x^2
sage: f = piecewise([[-1,1], f])
sage: f.fourier_series_cosine_coefficient(2)
pi^(-2)
sage: f1(x) = -1
sage: f2(x) = 2
sage: f = piecewise([[-pi, pi/2], f1], [(pi/2, pi), f2])
sage: f.fourier_series_cosine_coefficient(5, pi)
-3/5/pi

>>> from sage.all import *
>>> __tmp__=var("x"); f = symbolic_expression(x**Integer(2)).function(x)
>>> f = piecewise([[-Integer(1),Integer(1)], f])
>>> f.fourier_series_cosine_coefficient(Integer(2))
pi^(-2)
>>> __tmp__=var("x"); f1 = symbolic_expression(-Integer(1)).function(x)
>>> __tmp__=var("x"); f2 = symbolic_expression(Integer(2)).function(x)

(continues on next page)
>>> f = piecewise([((-pi, pi/Integer(2)), f1), ((pi/Integer(2), pi), f2)])
>>> f.fourier_series_cosine_coefficient(Integer(5), pi)
-3/5/pi

**fourier_series_partial_sum**(*parameters*, *variable*, *N=L=None*)

Returns the partial sum up to a given order of the Fourier series of the periodic function \(f\) extending the piecewise-defined function \(self\).

The Fourier partial sum of order \(N\) is defined as

\[
S_N(x) = \frac{a_0}{2} + \sum_{n=1}^{N} a_n \cos \left(\frac{n \pi x}{L}\right) + b_n \sin \left(\frac{n \pi x}{L}\right),
\]

where \(L\) is the half-period of \(f\) and the \(a_n\)'s and \(b_n\)'s are respectively the cosine coefficients and sine coefficients of the Fourier series of \(f\) (cf. \texttt{fourier_series_cosine_coefficient()} and \texttt{fourier_series_sine_coefficient()}).

**INPUT:**
- \(N\) — a positive integer; the order of the partial sum
- \(L\) — (default: None) the half-period of \(f\); if none is provided, \(L\) is assumed to be the half-width of the domain of \(self\)

**OUTPUT:**
- the partial sum \(S_N(x)\), as a symbolic expression

**EXAMPLES:**

A square wave function of period 2:

```sage
f = piecewise([((-1,0), -1), ((0,1), 1)])
f.fourier_series_partial_sum(5)
```

```sage
4/5*sin(5*pi*x)/pi + 4/3*sin(3*pi*x)/pi + 4*sin(pi*x)/pi
```

If the domain of the piecewise-defined function encompasses more than one period, the half-period must be passed as the second argument; for instance:

```sage
f2 = piecewise([((-Integer(1),Integer(0)), -Integer(1)), ((Integer(0),
Integer(1)), Integer(1))])
f2.fourier_series_partial_sum(Integer(5))
```

```sage
4/5*sin(5*pi*x)/pi + 4/3*sin(3*pi*x)/pi + 4*sin(pi*x)/pi
```

```sage
from sage.all import *
>>> f = piecewise([((-Integer(1),Integer(0)), -Integer(1)), ((Integer(0),
Integer(1)), Integer(1))])
>>> f.fourier_series_partial_sum(Integer(5))
```

```sage
4/5*sin(5*pi*x)/pi + 4/3*sin(3*pi*x)/pi + 4*sin(pi*x)/pi
```

```sage
f2.fourier_series_partial_sum(5, 1) # half-period = 1
```

```sage
4/5*sin(5*pi*x)/pi + 4/3*sin(3*pi*x)/pi + 4*sin(pi*x)/pi
```

```sage
f2.fourier_series_partial_sum(5, 1) == ...
```

```sage
f.fourier_series_partial_sum(5)
```

```sage
True
```
The default half-period is 2, so that skipping the second argument yields a different result:

```
sage: f2.fourier_series_partial_sum(5)  # half-period = 2
4*sin(pi*x)/pi
```

An example of partial sum involving both cosine and sine terms:

```
sage: f = piecewise([((-1,0), 0), ((0,1/2), 2*x),
                   ((1/2,1), 2*(1-x))])
sage: f.fourier_series_partial_sum(5)
-2*cos(2*pi*x)/pi^2 + 4/25*sin(5*pi*x)/pi^2
- 4/9*sin(3*pi*x)/pi^2 + 4*sin(pi*x)/pi^2 + 1/4
```

### `fourier_series_sine_coefficient` (parameters, variable, n, L=None)

Return the $n$-th sine coefficient of the Fourier series of the periodic function $f$ extending the piecewise-defined function `self`.

Given an integer $n \geq 0$, the $n$-th sine coefficient of the Fourier series of $f$ is defined by

$$b_n = \frac{1}{L} \int_{-L}^{L} f(x) \sin \left( \frac{n \pi x}{L} \right) \, dx,$$

where $L$ is the half-period of $f$. The number $b_n$ is the coefficient of $\sin(n \pi x/L)$ in the Fourier series of $f$ (cf. `fourier_series_partial_sum()`).

**INPUT:**
- $n$ – a non-negative integer
- $L$ – (default: None) the half-period of $f$; if none is provided, $L$ is assumed to be the half-width of the domain of `self`

**OUTPUT:**
- the Fourier coefficient $b_n$, as defined above

**EXAMPLES:**
A square wave function of period 2:
If the domain of the piecewise-defined function encompasses more than one period, the half-period must be passed as the second argument; for instance:

```
sage: f2 = piecewise([((-1,0), -1), ((0,1), 1),
                   ...: ((1,2), -1), ((2,3), 1)])
sage: bool(f2.restriction((-1,1)) == f)  # f2 extends f on (-1,3)
True
sage: f2.fourier_series_sine_coefficient(1, 1)  # half-period = 1
4/pi
sage: f2.fourier_series_sine_coefficient(3, 1)  # half-period = 1
4/3/pi
```

The default half-period is 2 and one has:

```
sage: f2.fourier_series_sine_coefficient(1)  # half-period = 2
0
sage: f2.fourier_series_sine_coefficient(3)  # half-period = 2
0
```

(continues on next page)
The Fourier coefficients obtained from $f$ are actually recovered for $n = 2$ and $n = 6$ respectively:

```
sage: f2.fourier_series_sine_coefficient(2)
4/pi
sage: f2.fourier_series_sine_coefficient(6)
4/3/pi
```

```
>>> from sage.all import *

>>> f2.fourier_series_sine_coefficient(Integer(2))
4/pi
>>> f2.fourier_series_sine_coefficient(Integer(6))
4/3/pi
```

```
integral (parameters, variable, x=None, a=None, b=None, definite=False, **kwds)
```

By default, return the indefinite integral of the function.

If `definite=True` is given, returns the definite integral.

AUTHOR:
- Paul Butler

EXAMPLES:

```
sage: f1(x) = 1 - x
sage: f = piecewise([((0,1), f1), ((1,2), f1)])

sage: f.integral(definite=True)
1/2
```

```
>>> from sage.all import *

>>> __tmp__=var("x"); f1 = symbolic_expression(Integer(1) - x).function(x)

>>> f = piecewise([((Integer(0),Integer(1)), f1), ((Integer(1), Integer(2)), f1)])

>>> f.integral(definite=True)
1/2
```

```
sage: f1(x) = -1
sage: f2(x) = 2
sage: f = piecewise([((-2, 0), f1), ((0, 3), f2)])

sage: f.integral()
piecewise(x|-->2*x + 4 on (-2, 0), x|-->-1/2*x^2 + 3*x + 4 on (0, 3); x)
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```python
sage: F = f.integral(y); F
piecewise(y|-->-y - 4 on [-4, -3],
   y|-->1/2*y^2 + 3*y + 7/2 on (-3, -2),
   y|-->-1/2*y^2 - y - 1/2 on [-2, 0],
   y|-->1/3*y^3 - y - 1/2 on (0, 2),
   y|-->3*y - 35/6 on [2, 3]; y)

>>> from sage.all import *
>>> __tmp__=var("x"); f1 = symbolic_expression(-Integer(1)).function(x)  
>>> __tmp__=var("x"); f2 = symbolic_expression(Integer(2)).function(x)  
>>> f = piecewise([[(Integer(0),pi/Integer(2)), f1], ((pi/Integer(2),pi), ...  
                  f2)])
>>> f.integral(definite=True)
1/2*pi

>>> __tmp__=var("x"); f1 = symbolic_expression(Integer(2)).function(x)  
>>> __tmp__=var("x"); f2 = symbolic_expression(Integer(3) - x).function(x)  
>>> f = piecewise([[(Integer(2), Integer(0)), f1], [(Integer(0), ...  
                  f2)])
>>> f.integral()
1/2*pi

>>> __tmp__=var("x"); f1 = symbolic_expression(-Integer(1)).function(x)  
>>> __tmp__=var("x"); f2 = symbolic_expression(Integer(2)).function(x)  
>>> __tmp__=var("y"); f3 = symbolic_expression(-y - Integer(1)).function(y)  
>>> f = piecewise([[(Integer(0),Integer(2)), f1], [(Integer(2), ...  
                  f3)])
>>> F = f.integral(y); F
piecewise(y|-->-y - 4 on [-4, -3],
   y|-->1/2*y^2 + 3*y + 7/2 on (-3, -2),
   y|-->-1/2*y^2 - y - 1/2 on [-2, 0],
   y|-->1/3*y^3 - y - 1/2 on (0, 2),
   y|-->3*y - 35/6 on [2, 3]; y)
```

Ensure results are consistent with FTC:

```python
sage: F(-3) - F(-4)
-1
sage: F(-1) - F(-3)
1
sage: F(2) - F(0)
2/3
sage: f.integral(y, 0, 2)
2/3
sage: F(3) - F(-4)
19/6
sage: f.integral(y, -4, 3)
19/6
sage: f.integral(definite=True)
19/6
```
>>> from sage.all import *

>>> F(-Integer(3)) - F(-Integer(4))
-1

>>> F(-Integer(1)) - F(-Integer(3))
1

>>> F(Integer(2)) - F(Integer(0))
2/3

>>> f.integral(y, Integer(0), Integer(2))
2/3

>>> F(Integer(3)) - F(-Integer(4))
19/6

>>> f.integral(y, -Integer(4), Integer(3))
19/6

>>> f.integral(definite=True)
19/6

sage:

f1(y) = (y+3)^2
f2(y) = y+3
f3(y) = 3

f = piecewise([[(-infinity, -3), f1], [-3, 0), f2],
               [(0, infinity), f3]])

f.integral()

piecewise(y|-->1/3*y^3 + 3*y^2 + 9*y + 9 on (-oo, -3),
y|-->1/2*y^2 + 3*y + 9/2 on (-3, 0),
y|-->3*y + 9/2 on (0, +oo); y)

sage:

f1(x) = e^(-abs(x))

f = piecewise([[-infinity, infinity], f1]])

result = f.integral(definite=True)
...

result
2

f.integral()
piecewise(x|-->-integrate(e^(-abs(x)), x, x, +Infinity) on (-oo, +oo); x)

sage:

__tmp__=var("x"); f1 = symbolic_expression(e**(-abs(x))).function(x)

f = piecewise([[-infinity, infinity], f1]])

result = f.integral(definite=True)
...

result
2

f.integral()
Functions, Release 10.4

(continued from previous page)

piecewise(x|-->-integrate(e^(-abs(x)), x, x, +Infinity) on (-oo, +oo); x)

```
sage: f = piecewise([(0, 5), cos(x)])
sage: f.integral()
```

```
>>> from sage.all import *

>>> f = piecewise([(Integer(0), Integer(5)), cos(x)])

```

```
items(parameters, variable)
```

Iterate over the pieces of the piecewise function

Note: You should probably use `pieces()` instead, which offers a nicer interface.

OUTPUT:

This method iterates over pieces of the piecewise function, each represented by a pair. The first element is the support, and the second the function over that support.

EXAMPLES:

```
sage: f = piecewise([(0,0), sin(x)], ((0,2), cos(x))]
sage: for support, function in f.items():
....:
    print('support is {0}, function is {1}'.format(support, function))
support is {0}, function is sin(x)
support is (0, 2), function is cos(x)
```

```
>>> from sage.all import *

>>> f = piecewise([(Integer(0),Integer(0)), sin(x)], ((Integer(0), Integer(2)), cos(x))]

>>> for support, function in f.items():
...
    print('support is {0}, function is {1}'.format(support, function))
support is {0}, function is sin(x)
support is (0, 2), function is cos(x)
```

```
laplace(parameters, variable, x='x', s='t')
```

Return the Laplace transform of self with respect to the variable var.

INPUT:

• x – variable of self
• s – variable of Laplace transform.

We assume that a piecewise function is 0 outside of its domain and that the left-most endpoint of the domain is 0.

EXAMPLES:

```
sage: x, s, w = var('x, s, w')
sage: f = piecewise([(0,1), 1], [[1,2], 1 - x])
sage: f.laplace(x, s)
```

```
-e^(-s)/s + (s + 1)*e^(-2*s)/s^2 + 1/s - e^(-s)/s^2
```

```
sage: f.laplace(x, w)
```

```
-e^(-w)/w + (w + 1)*e^(-2*w)/w^2 + 1/w - e^(-w)/w^2
```
>>> from sage.all import *
>>> x, s, w = var('x, s, w')
>>> f = piecewise([[Integer(0),Integer(1)], Integer(1) - x]])
>>> f.laplace(x, s)
-e^(-s)/s + (s + 1)*e^(-2*s)/s^2 + 1/s - e^(-s)/s^2

sage: y, t = var('y, t')
sage: f = piecewise([[Integer(1),Integer(2)], Integer(1) - y]])
sage: f.laplace(y, t)
(t + 1)*e^(-2*t)/t^2 - e^(-t)/t^2

sage: s = var('s')
sage: t = var('t')
sage: f1(t) = -t
sage: f2(t) = Integer(2)
>>> f = piecewise([[Integer(0),Integer(1)], f1], [(Integer(1),infinity), f2])
>>> f.laplace(t, s)
(s + 1)*e^(-s)/s^2 + 2*e^(-s)/s - 1/s^2

pieces (parameters, variable)

Return the “pieces”.

OUTPUT:

A tuple of piecewise functions, each having only a single expression.

EXAMPLES:

sage: p = piecewise([((-Integer(1), Integer(0)), -x), ([Integer(0), Integer(1)], x)], var=x)
sage: p.pieces()
(piecewise(x|-->-x on (-1, 0); x),
 piecewise(x|-->x on [0, 1]; x))
piecewise_add (parameters, variable, other)

Return a new piecewise function with domain the union of the original domains and functions summed. Undefined intervals in the union domain get function value 0.

EXAMPLES:

```python
sage: f = piecewise([[(0, 1), 1], ((2, 3), x)])
sage: g = piecewise([[(1/2, 2), x]])
sage: f.piecewise_add(g).unextend_zero()
piecewise(x|-->1 on (0, 1/2],
       x|-->x + 1 on (1/2, 1],
       x|-->x on (1, 2) ∪ (2, 3); x)
```

restriction (parameters, variable, restricted_domain)

Restrict the domain

INPUT:
- restricted_domain — a `RealSet` or something that defines one.

OUTPUT:
A new piecewise function obtained by restricting the domain.

EXAMPLES:

```python
sage: f = piecewise([((-oo, oo), x)])
f
piecewise(x|-->x on (-oo, +oo); x)
sage: f.restriction([[-1, 1], [3, 3]])
piecewise(x|-->x on [-1, 1] ∪ {3}; x)
```

trapezoid (parameters, variable, N)

Return the piecewise line function defined by the trapezoid rule for numerical integration based on a subdivision of each domain interval into N subintervals.

EXAMPLES:

```python
sage: f = piecewise([[[0, 1], x^2],
                  ....: [RealSet.open_closed(1, 2), 5 - x^2]])
sage: f.trapezoid(2)
```
Functions, Release 10.4

piecewise(x|-->1/2*x on (0, 1/2),
    x|-->3/2*x - 1/2 on (1/2, 1),
    x|-->7/2*x - 5/2 on (1, 3/2),
    x|-->-7/2*x + 8 on (3/2, 2); x)
sage: f = piecewise([[-1,1], 1 - x^2])
sage: f.trapezoid(4).integral(definite=True)
5/4
sage: f = piecewise([[-1,1], 1/2 + x - x^3])

## example 3
sage: f.trapezoid(6).integral(definite=True)
1

unextend_zero (parameters, variable)
Remove zero pieces.

EXAMPLES:
sage: f = piecewise([((-1,1), x)]); f
piecewise(x|-->x on (-1, 1); x)
sage: g = f.extension(0); g
piecewise(x|-->x on (-1, 1), x|-->0 on (-oo, -1] ∪ [1, +oo); x)
sage: g(3)
0
sage: h = g.unextend_zero()
sage: bool(h == f)
True

>>> from sage.all import *

>>> f = piecewise([((-Integer(1),Integer(1)), x)])

>>> g = f.extension(Integer(0)); g

>>> g(Integer(3))
0

>>> h = g.unextend_zero()

>>> bool(h == f)
True

which_function (parameters, variable, point)

1.6. Piecewise functions
Return the expression defining the piecewise function at value

INPUT:
• point – a real number.

OUTPUT:
The symbolic expression defining the function value at the given point.

EXAMPLES:

```
sage: f = piecewise(((0, 0), sin(x)), ((0, 2), cos(x))); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
sage: f.expression_at(0)
sin(x)
sage: f.expression_at(1)
cos(x)
sage: f.expression_at(2)
Traceback (most recent call last):
... ValueError: point is not in the domain
```

```
>>> from sage.all import *
>>> f = piecewise(((Integer(0),Integer(0)], sin(x)), ((Integer(0),
    Integer(2)), cos(x)))); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2); x)
>>> f.expression_at(Integer(0))
sin(x)
>>> f.expression_at(Integer(1))
cos(x)
>>> f.expression_at(Integer(2))
Traceback (most recent call last):
... ValueError: point is not in the domain
```

`static in_operands (ex)`

Return whether a symbolic expression contains a piecewise function as operand

INPUT:
• ex – a symbolic expression.

OUTPUT:
Boolean

EXAMPLES:

```
sage: f = piecewise(((0, 0], sin(x)), ((0, 2], cos(x)))); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2]; x)
sage: piecewise.in_operands(f)
True
sage: piecewise.in_operands(1+sin(f))
True
sage: piecewise.in_operands(1+sin(0*f))
False
```

```
>>> from sage.all import *
>>> f = piecewise(((Integer(0),Integer(0]], sin(x)), ((Integer(0),
    Integer(2]), cos(x)))); f
piecewise(x|-->sin(x) on {0}, x|-->cos(x) on (0, 2]; x)
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```python
>>> piecewise.in_operands(f)
True
>>> piecewise.in_operands(Integer(1)+sin(f))
True
>>> piecewise.in_operands(Integer(1)+sin(Integer(0)*f))
False
```

**static simplify**(ex)

Combine piecewise operands into single piecewise function

**OUTPUT:**

A piecewise function whose operands are not piecewiese if possible, that is, as long as the piecewise variable is the same.

**EXAMPLES:**

```python
sage: f = piecewise([[0,0], sin(x)], ((0,2), cos(x)])
sage: piecewise.simplify(f)
Traceback (most recent call last):
 ... Not ImplementedError
```

```python
>>> from sage.all import *
```

```python
>>> f = piecewise([([Integer(0), Integer(0)], sin(x)), ((Integer(0),
˓→ Integer(2)), cos(x))])
```

```python
>>> piecewise.simplify(f)
Traceback (most recent call last):
 ... Not ImplementedError
```

1.7 Spike functions

**AUTHORS:**

• William Stein (2007-07): initial version

• Karl-Dieter Crisman (2009-09): adding documentation and doctests

**class** sage.functions.spike_function.SpikeFunction(v, eps=1e-07)

```
Bases: object
```

Base class for spike functions.

**INPUT:**

• v – list of pairs (x, height)

• eps – parameter that determines approximation to a true spike

**OUTPUT:**

a function with spikes at each point x in v with the given height.

**EXAMPLES:**

```python
sage: spike_function([[-3,4], (-1,1), (2,3)], 0.001)
A spike function with spikes at [-3.0, -1.0, 2.0]
```
>>> from sage.all import *
>>> spike_function([(-Integer(3),Integer(4)), (-Integer(1),Integer(1)),
                  (Integer(2),Integer(3))], RealNumber('0.001'))
A spike function with spikes at [-3.0, -1.0, 2.0]

Putting the spikes too close together may delete some:

sage: spike_function([[1,1], (1.01,4)], 0.1)
Some overlapping spikes have been deleted.
You might want to use a smaller value for eps.
A spike function with spikes at [1.0]

>>> from sage.all import *
>>> spike_function([[Integer(1),Integer(1)], (RealNumber('1.01'),Integer(4))],
                  RealNumber('0.1'))
Some overlapping spikes have been deleted.
You might want to use a smaller value for eps.
A spike function with spikes at [1.0]

Note this should normally be used indirectly via spike_function, but one can use it directly:

sage: from sage.functions.spike_function import SpikeFunction
sage: S = SpikeFunction([[0,1], (1,2), (pi,-5)]); S
A spike function with spikes at [0.0, 1.0, 3.141592653589793]
sage: S.support
[0.0, 1.0, 3.141592653589793]

plot (xmin=None, xmax=None, **kwds)
Special fast plot method for spike functions.

EXAMPLES:

sage: S = spike_function([(-1,1), (1,40)])
sage: P = plot(S)
Line defined by 8 points

>>> from sage.all import *
>>> S = spike_function([(-Integer(1),Integer(1)), (Integer(1),Integer(40))])
>>> P = plot(S)
Line defined by 8 points
**plot_fft_abs** *(samples=4096, xmin=None, xmax=None, **kwds)*

Plot of (absolute values of) Fast Fourier Transform of the spike function with given number of samples.

**EXAMPLES:**

```python
sage: S = spike_function([(-3,4), (-1,1), (2,3)]); S
A spike function with spikes at [-3.0, -1.0, 2.0]
sage: P = S.plot_fft_abs(8) # needs sage.plot
sage: p = P[0]; p.ydata # abs tol 1e-8 # needs sage.plot
[5.0, 5.0, 3.367958691924177, 3.367958691924177, 4.123105625617661, 4.123105625617661, 4.759921664218055, 4.759921664218055]
```

**plot_fft_arg** *(samples=4096, xmin=None, xmax=None, **kwds)*

Plot of (absolute values of) Fast Fourier Transform of the spike function with given number of samples.

**EXAMPLES:**

```python
sage: S = spike_function([(-3,4), (-1,1), (2,3)]); S
A spike function with spikes at [-3.0, -1.0, 2.0]
sage: P = S.plot_fft_arg(8) # needs sage.plot
sage: p = P[0]; p.ydata # abs tol 1e-8 # needs sage.plot
[0.0, 0.0, -0.211524990023434, -0.211524990023434, 0.244978663126864, 0.244978663126864, -0.149106180027477, -0.149106180027477]
```

**vector** *(samples=65536, xmin=None, xmax=None)*

Create a sampling vector of the spike function in question.

**EXAMPLES:**

```python
>> from sage.all import *
>> S = spike_function([(-Integer(3),Integer(4)), (-Integer(1),Integer(1)), (Integer(2),Integer(3))]); S
A spike function with spikes at [-3.0, -1.0, 2.0]
>> P = S.plot_fft_arg(Integer(8)) # needs sage.plot
>> p = P[Integer(0)]; p.ydata # abs tol 1e-8 # needs sage.plot
[0.0, 0.0, -0.211524990023434, -0.211524990023434, 0.244978663126864, 0.244978663126864, -0.149106180027477, -0.149106180027477]
```
Functions, Release 10.4

sage: S = spike_function([(-3,4), (-1,1), (2,3)],0.001); S
A spike function with spikes at [-3.0, -1.0, 2.0]
sage: S.vector(16)  # needs sage.modules
(4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

>>> from sage.all import *
>>> S = spike_function([(Integer(3),Integer(4)), (Integer(1),Integer(1)),
(2,3)],RealNumber('0.001')); S
A spike function with spikes at [-3.0, -1.0, 2.0]
>>> S.vector(Integer(16))  # needs sage.modules
(4.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

sage.functions.spike_function.spike_function
alias of SpikeFunction

1.8 Orthogonal polynomials

1.8.1 Chebyshev polynomials

The Chebyshev polynomial of the first kind arises as a solution to the differential equation

\[(1 - x^2) y'' - xy' + n^2 y = 0\]

and those of the second kind as a solution to

\[(1 - x^2) y'' - 3xy' + n(n + 2) y = 0.\]

The Chebyshev polynomials of the first kind are defined by the recurrence relation

\[T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).\]

The Chebyshev polynomials of the second kind are defined by the recurrence relation

\[U_0(x) = 1, \quad U_1(x) = 2x, \quad U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x).\]

For integers \(m, n\), they satisfy the orthogonality relations

\[\int_{-1}^{1} T_n(x)T_m(x) \frac{dx}{\sqrt{1-x^2}} = \begin{cases}
0 & \text{if } n \neq m, \\
\pi & \text{if } n = m = 0, \\
\pi/2 & \text{if } n = m \neq 0,
\end{cases}\]

and

\[\int_{-1}^{1} U_n(x)U_m(x)\sqrt{1-x^2} \, dx = \frac{\pi}{2} \delta_{m,n}.\]

They are named after Pafnuty Chebyshev (1821-1894, alternative transliterations: Tchebycheff or Tschebyscheff).
1.8.2 Hermite polynomials

The Hermite polynomials are defined either by

\[ H_n(x) = (-1)^n e^{x^2/2} \frac{d^n}{dx^n} e^{-x^2/2} \]

(the “probabilists’ Hermite polynomials”), or by

\[ H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \]

(the “physicists’ Hermite polynomials”). Sage (via Maxima) implements the latter flavor. These satisfy the orthogonality relation

\[ \int_{-\infty}^{\infty} H_n(x) H_m(x) e^{-x^2} dx = \sqrt{\pi} n! 2^n \delta_{nm}. \]

They are named in honor of Charles Hermite (1822-1901), but were first introduced by Laplace in 1810 and also studied by Chebyshev in 1859.

1.8.3 Legendre polynomials

Each Legendre polynomial \( P_n(x) \) is an \( n \)-th degree polynomial. It may be expressed using Rodrigues’ formula:

\[ P_n(x) = (2^n n!)^{-1} \frac{d^n}{dx^n} [(x^2 - 1)^n]. \]

These are solutions to Legendre’s differential equation:

\[ \frac{d}{dx} \left[ (1 - x^2) \frac{d}{dx} P(x) \right] + n(n + 1) P(x) = 0 \]

and satisfy the orthogonality relation

\[ \int_{-1}^{1} P_m(x) P_n(x) dx = \frac{2}{2n + 1} \delta_{mn}. \]

The Legendre function of the second kind \( Q_n(x) \) is another (linearly independent) solution to the Legendre differential equation. It is not an “orthogonal polynomial” however.

The associated Legendre functions of the first kind \( P_{\ell}^m(x) \) can be given in terms of the “usual” Legendre polynomials by

\[ P_{\ell}^m(x) = (-1)^m (1 - x^2)^{m/2} \frac{d^m}{dx^m} P_\ell(x) \]

\[ = \frac{(-1)^m}{2\ell!} (1 - x^2)^{m/2} \frac{d^{\ell+m}}{dx^{\ell+m}} (x^2 - 1)^\ell. \]

Assuming \( 0 \leq m \leq \ell \), they satisfy the orthogonality relation:

\[ \int_{-1}^{1} P_k^m(x) P_\ell^m(x) dx = \frac{2(\ell + m)!}{(2\ell + 1)(\ell - m)!} \delta_{k,\ell}, \]

where \( \delta_{k,\ell} \) is the Kronecker delta.

The associated Legendre functions of the second kind \( Q_{\ell}^m(x) \) can be given in terms of the “usual” Legendre polynomials by

\[ Q_{\ell}^m(x) = (-1)^m (1 - x^2)^{m/2} \frac{d^m}{dx^m} Q_\ell(x). \]

They are named after Adrien-Marie Legendre (1752-1833).
1.8.4 Laguerre polynomials

Laguerre polynomials may be defined by the Rodrigues formula

\[ L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (e^{-x} x^n). \]

They are solutions of Laguerre’s equation:

\[ x y'' + (1 - x) y' + n y = 0 \]

and satisfy the orthogonality relation

\[ \int_0^\infty L_m(x)L_n(x)e^{-x} \, dx = \delta_{mn}. \]

The generalized Laguerre polynomials may be defined by the Rodrigues formula:

\[ L_n^{(\alpha)}(x) = \frac{x^{-\alpha}e^x}{n!} \frac{d^n}{dx^n} (e^{-x} x^{n+\alpha}). \]

(These are also sometimes called the associated Laguerre polynomials.) The simple Laguerre polynomials are recovered from the generalized polynomials by setting \( \alpha = 0 \).

They are named after Edmond Laguerre (1834-1886).

1.8.5 Jacobi polynomials

Jacobi polynomials are a class of orthogonal polynomials. They are obtained from hypergeometric series in cases where the series is in fact finite:

\[ P_n^{(\alpha,\beta)}(z) = \frac{(\alpha + 1)_n}{n!} {}_2F_1 \left( -n, 1 + \alpha + \beta + n; \alpha + 1; \frac{1 - z}{2} \right), \]

where \((\cdot)_n\) is Pochhammer’s symbol (for the rising factorial), (Abramowitz and Stegun p561.) and thus have the explicit expression

\[ P_n^{(\alpha,\beta)}(z) = \frac{\Gamma(\alpha + n + 1)}{n!\Gamma(\alpha + \beta + n + 1)} \sum_{m=0}^n \frac{n!}{m!} \frac{\Gamma(\alpha + \beta + n + m + 1)}{\Gamma(\alpha + m + 1)} \left( \frac{z - 1}{2} \right)^m. \]

They are named after Carl Gustav Jacob Jacobi (1804-1851).

1.8.6 Gegenbauer polynomials

Ultraspherical or Gegenbauer polynomials are given in terms of the Jacobi polynomials \( P_n^{(\alpha,\beta)}(x) \) with \( \alpha = \beta = a - 1/2 \) by

\[ C_n^{(a)}(x) = \frac{\Gamma(a + 1/2)}{\Gamma(2a)} \frac{\Gamma(n + 2a)}{\Gamma(n + a + 1/2)} P_n^{(a-1/2,a-1/2)}(x). \]

They satisfy the orthogonality relation

\[ \int_{-1}^1 (1 - x^2)^{a-1/2} C_m^{(a)}(x)C_n^{(a)}(x) \, dx = \delta_{mn} 2^{1-2a} \frac{\Gamma(n + 2a)}{(n + a)\Gamma^2(a)\Gamma(n + 1)}, \]

for \( a > -1/2 \). They are obtained from hypergeometric series in cases where the series is in fact finite:

\[ C_n^{(a)}(z) = \frac{(2a)_n}{n!} \frac{\Gamma(n + 2a)}{(n + a)\Gamma^2(a)} {}_2F_1 \left( -n, 2a + n; a + \frac{1}{2}; \frac{1 - z}{2} \right) \]

where \( n \) is the falling factorial. (See Abramowitz and Stegun p561.)

They are named for Leopold Gegenbauer (1849-1903).
1.8.7 Krawtchouk polynomials

The Krawtchouk polynomials are discrete orthogonal polynomials that are given by the hypergeometric series

\[ K_j(x; n, p) = (-1)^j \binom{n}{j} p^j 2F_1 (-j, -x; -n; p^{-1}). \]

Since they are discrete orthogonal polynomials, they satisfy an orthogonality relation defined on a discrete (in this case finite) set of points:

\[ \sum_{m=0}^{n} K_i(m; n, p) K_j(m; n, p) \binom{n}{m} p^m q^{n-m} = \binom{n}{j} (pq)^j \delta_{ij}, \]

where \( q = 1 - p \). They can also be described by the recurrence relation

\[ jK_j(x; n, p) = (x - (n - j + 1)p - (j - 1)q)K_{j-1}(x; n, p) - pq(n - j + 2)K_{j-2}(x; n, p), \]

where \( K_0(x; n, p) = 1 \) and \( K_1(x; n, p) = x - np \).

They are named for Mykhailo Krawtchouk (1892-1942).

1.8.8 Meixner polynomials

The Meixner polynomials are discrete orthogonal polynomials that are given by the hypergeometric series

\[ M_n(x; n, p) = (-1)^j \binom{n}{j} p^j 2F_1 (-j, -x; -n; p^{-1}). \]

They satisfy an orthogonality relation:

\[ \sum_{k=0}^{\infty} \tilde{M}_n(k; b, c) \tilde{M}_m(k; b, c) \frac{(b)_k}{k!} c^k = \frac{c^{-n} n!}{(b)_n(1 - c)^b} \delta_{mn}, \]

where \( \tilde{M}_n(x; b, c) = M_n(x; b, c)/(b)_x \), for \( b > 0 \) and \( 0 < c < 1 \). They can also be described by the recurrence relation

\[ c(n - 1 + b)M_n(x; b, c) = ((c - 1)x + n - 1 + c(n - 1 + b))(b + n - 1)M_{n-1}(x; b, c) - (b + n - 1)(b + n - 2)(n - 1)M_{n-2}(x; b, c), \]

where \( M_0(x; b, c) = 0 \) and \( M_1(x; b, c) = (1 - c^{-1})x + b \).

They are named for Josef Meixner (1908-1994).

1.8.9 Hahn polynomials

The Hahn polynomials are discrete orthogonal polynomials that are given by the hypergeometric series

\[ Q_k(x; a, b, n) = 3F_2 (-k, k + a + b + 1, -x; a + 1, -n; 1). \]

They satisfy an orthogonality relation:

\[ \sum_{k=0}^{n-1} Q_i(k; a, b, n) Q_j(k; a, b, n) \rho(k) = \frac{\delta_{ij}}{\pi_i}, \]
Functions, Release 10.4

where

\[ \rho(k) = \binom{a + k}{k} \binom{b + n - k}{n - k}, \]
\[ \pi_i = \delta_{ij} (-1)^i (b + 1) (i + a + b + 1)_{n+1} \frac{n!(2i + a + b + 1)(-n)_i(a + 1)_i}{n!}. \]

They can also be described by the recurrence relation

\[ A Q_k(x; a, b, n) = (-x + A + C) Q_{k-1}(x; a, b, n) - C Q_{k-2}(x; a, b, n), \]

where \( Q_0(x; a, b, n) = 1 \) and \( Q_1(x; a, b, n) = 1 - \frac{a + b + 2}{(a + 1)n} x \) and

\[ A = \frac{(k + a + b)(k + a)(n - k + 1)}{(2k + a + b - 1)(2k + a + b)}, \quad C = \frac{(k - 1)(k + b - 1)(k + a + b + n)}{(2k + a + b - 2)(2k + a + b - 1)}. \]

They are named for Wolfgang Hahn (1911-1998), although they were first introduced by Chebyshev in 1875.

### 1.8.10 Pochhammer symbol

For completeness, the Pochhammer symbol, introduced by Leo August Pochhammer, \((x)_n\), is used in the theory of special functions to represent the “rising factorial” or “upper factorial”

\[ (x)_n = x(x + 1)(x + 2) \cdots (x + n - 1) = \frac{(x + n - 1)!}{(x - 1)!}. \]

On the other hand, the falling factorial or lower factorial is

\[ x^n = \frac{x!}{(x - n)!}, \]

in the notation of Ronald L. Graham, Donald E. Knuth and Oren Patashnik in their book Concrete Mathematics.


REFERENCES:

• [AS1964]
• [Wikipedia article Chebyshev_polynomials](https://en.wikipedia.org/wiki/Chebyshev_polynomials)
• [Wikipedia article Legendre_polynomials](https://en.wikipedia.org/wiki/Legendre_polynomials)
• [Wikipedia article Hermite_polynomials](https://en.wikipedia.org/wiki/Hermite_polynomials)
• [Wikipedia article Jacobi_polynomials](https://en.wikipedia.org/wiki/Jacobi_polynomials)
• [Wikipedia article Laguerre_polynomials](https://en.wikipedia.org/wiki/Laguerre_polynomials)
• [Wikipedia article Associated_Legendre_polynomials](https://en.wikipedia.org/wiki/Associated_Legendre_polynomials)
• [Wikipedia article Meixner_polynomials](https://en.wikipedia.org/wiki/Meixner_polynomials)
• [Wikipedia article Hahn_polynomials](https://en.wikipedia.org/wiki/Hahn_polynomials)
• Roelof Koekeek and René F. Swarttouw, [arXiv math/9602214](https://arxiv.org/abs/math/9602214)
• [Koe1999]

AUTHORS:
• David Joyner (2006-06)
• Stefan Reiterer (2010-)
• Ralf Stephan (2015-)

The original module wrapped some of the orthogonal/special functions in the Maxima package “orthopoly” and was written by Barton Willis of the University of Nebraska at Kearney.

```python
class sage.functions.orthogonal_polys.ChebyshevFunction(name, nargs=2,
latex_name=None, conversions=None)
```

Bases: OrthogonalFunction

Abstract base class for Chebyshev polynomials of the first and second kind.

EXAMPLES:

```python
sage: chebyshev_T(3, x)
# needs sage.symbolic
4*x^3 - 3*x
```

```python
>>> from sage.all import *

>>> chebyshev_T(Integer(3), x) # needs sage.symbolic
4*x^3 - 3*x
```

```python
class sage.functions.orthogonal_polys.Func_assoc_legendre_P
```

Bases: BuiltinFunction

Return the Ferrers function \( P_{m n}(x) \) of first kind for \( x \in (-1, 1) \) with general order \( m \) and general degree \( n \).

Ferrers functions of first kind are one of two linearly independent solutions of the associated Legendre differential equation

\[
(1 - x^2) \frac{d^2 w}{dx^2} - 2x \frac{dw}{dx} + \left( n(n+1) - \frac{m^2}{1 - x^2} \right) w = 0
\]

on the interval \( x \in (-1, 1) \) and are usually denoted by \( P_n^m(x) \).

See also:
The other linearly independent solution is called *Ferrers function of second kind* and denoted by \( Q_n^m(x) \), see Func_assoc_legendre_Q.

Warning: Ferrers functions must be carefully distinguished from associated Legendre functions which are defined on \( C \setminus (-\infty, 1] \) and have not yet been implemented.

EXAMPLES:

We give the first Ferrers functions for non-negative integers \( n \) and \( m \) in the interval \(-1 < x < 1\):

```python
sage: for n in range(4):
    # needs sage.symbolic
    ....:     for m in range(n+1):
```

(continues on next page)
These expressions for non-negative integers are computed by the Rodrigues-type given in `eval_gen_poly()`. Negative values for $n$ are obtained by the following identity:

$$P_{m}^{n}(x) = P_{n}^{-1}(x).$$

For $n$ being a non-negative integer, negative values for $m$ are obtained by

$$P_{n}^{-m}(x) = (-1)^{|m|} \frac{(n - |m|)!}{(n + |m|)!} P_{n}^{m}(x),$$

where $|m| \leq n$.

Here are some specific values with negative integers:

```python
sage: # needs sage.symbolic
gen_legendre_P(-2, -1, x)
1/2*sqrt(-x^2 + 1)
sage: gen_legendre_P(2, -2, x)
-1/8*x^2 + 1/8
sage: gen_legendre_P(3, -2, x)
-1/8*(x^2 - 1)*x
sage: gen_legendre_P(1, -2, x)
0
```

```python
>>> from sage.all import *
"from sage.all import *")
# needs sage.symbolic

>>> for m in range(n+Integer(1)): ...
    print(f"P_{n}^{m}(x) = (gen_legendre_P(n, m, x))")

P_0^0(x) = 1
P_1^0(x) = x
P_1^1(x) = -sqrt(-x^2 + 1)
P_2^0(x) = 3/2*x^2 - 1/2
P_2^1(x) = -3*sqrt(-x^2 + 1)*x
P_2^2(x) = -3*x^2 + 3
P_3^0(x) = 5/2*x^3 - 3/2*x
P_3^1(x) = -3/2*(5*x^2 - 1)*sqrt(-x^2 + 1)
P_3^2(x) = -15*(x^2 - 1)*x
P_3^3(x) = -15*(-x^2 + 1)^(3/2)
```

>>> from sage.all import *
"from sage.all import *")
# needs sage.symbolic

>>> for m in range(n+Integer(1)): ...
    print(f"P_{n}^{m}(x) = (gen_legendre_P(n, m, x))")

P_0^0(x) = 1
P_1^0(x) = x
P_1^1(x) = -sqrt(-x^2 + 1)
P_2^0(x) = 3/2*x^2 - 1/2
P_2^1(x) = -3*sqrt(-x^2 + 1)*x
P_2^2(x) = -3*x^2 + 3
P_3^0(x) = 5/2*x^3 - 3/2*x
P_3^1(x) = -3/2*(5*x^2 - 1)*sqrt(-x^2 + 1)
P_3^2(x) = -15*(x^2 - 1)*x
P_3^3(x) = -15*(-x^2 + 1)^(3/2)
```
Here are some other random values with floating numbers:

```python
sage: # needs sage.symbolic
sage: m = var('m'); assume(m, 'integer')
sage: gen_legendre_P(m, m, RealNumber(0.2))
0.960000000000000^(1/2*m)*(-1)^m*factorial(2*m)/(2^m*factorial(m))
sage: gen_legendre_P(RealNumber(0.2), m, Integer(0))
0.757714892929573
sage: gen_legendre_P(RealNumber(0.2), RealNumber(0.2), RealNumber(0.2))
0.757714892929573
```

REFERENCES:

- [DLMF-Legendre](#)

```python
def DeprecatedFunctionAlias(issue_number, func):
    Create an aliased version of a function or a method which raises a deprecation warning message.

    If f is a function or a method, write g = deprecated_function_alias(issue_number, f) to make a deprecated aliased version of f.

    INPUT:

    - issue_number -- integer. The github issue number where the deprecation is introduced.
    - func -- the function or method to be aliased

    EXAMPLES:

    ```python
    sage: from sage.misc.superseded import deprecated_function_alias
    sage: g = deprecated_function_alias(13109, number_of_partitions)  # needs sage.combinat sage.libs.flint
    sage: g(5)  # needs sage.combinat sage.libs.flint
doctest:...: DeprecationWarning: g is deprecated. Please use sage.combinat.partition.number_of_partitions instead. See https://github.com/sagemath/sage/issues/13109 for details.
    7
    ```
```
This also works for methods:

```
sage: class cls():
    ....:     def new_meth(self): return 42
    ....:     old_meth = deprecated_function_alias(13109, new_meth)
sage: cls().old_meth()
```

```
        doctest:...: DeprecationWarning: old_meth is deprecated. Please use new_meth instead.
        See https://github.com/sagemath/sage/issues/13109 for details.
42
```

Issue #11585:

```
sage: def a(): pass
sage: b = deprecated_function_alias(13109, a)
sage: b()
```

```
        doctest:...: DeprecationWarning: b is deprecated. Please use a instead.
        See https://github.com/sagemath/sage/issues/13109 for details.
```

AUTHORS:

- Florent Hivert (2009-11-23), with the help of Mike Hansen.
- Luca De Feo (2011-07-11), printing the full module path when different from old path

\texttt{eval\_gen\_poly}(n, m, arg, **\textit{kwds})

Return the Ferrers function of first kind \( P_n^m(x) \) for integers \( n > -1, m > -1 \) given by the following Rodrigues-type formula:

\[
P_n^m(x) = (-1)^{m+n} \frac{(1 - x^2)^{m/2}}{2^n n!} \frac{d^{m+n}}{dx^{m+n}} (1 - x^2)^n.
\]
INPUT:

- \( n \) – an integer degree
- \( m \) – an integer order
- \( x \) – either an integer or a non-numerical symbolic expression

EXAMPLES:

```python
sage: gen_legendre_P(7, 4, x)  # needs sage.symbolic
3465/2*(13*x^3 - 3*x)*(x^2 - 1)^2
sage: gen_legendre_P(3, 1, sqrt(x))  # needs sage.symbolic
-3/2*(5*x - 1)*sqrt(-x + 1)
```

REFERENCE:

- [DLMF-Legendre], Section 14.7 eq. 10 (https://dlmf.nist.gov/14.7#E10)

```
eval_poly (*args, **kwds)
```


```
class sage.functions.orthogonal_polys.Func_assoc_legendre_Q
```

Bases: BuiltinFunction

EXAMPLES:

```python
sage: loads(dumps(gen_legendre_Q))
gen_legendre_Q
sage: maxima(gen_legendre_Q(2, 1, 3, hold=True))._sage_().simplify_full()  # needs sage.symbolic
1/4*sqrt(2)*(36*pi - 36*I*log(2) + 25*I)
```

```
eval_recursive (n, m, x, **kwds)
```

Return the associated Legendre Q(n, m, arg) function for integers \( n > -1, m > -1 \).

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: gen_legendre_Q(3, 4, x) 48/(x^2 - 1)^2
sage: gen_legendre_Q(4, 5, x)
```
Functions, Release 10.4

(continued from previous page)

```plaintext
-384/((x^2 - 1)^2*sqrt(-x^2 + 1))
sage: gen_legendre_Q(0, 1, x)
-1/sqrt(-x^2 + 1)
sage: gen_legendre_Q(0, 2, x)
-1/2*((x + 1)^2 - (x - 1)^2)/(x^2 - 1)
sage: gen_legendre_Q(2, 2, x).subs(x=2).expand()
9/2*I*pi - 9/2*log(3) + 14/3
```

```plaintext
>>> from sage.all import *
>>> # needs sage.symbolic
>>> gen_legendre_Q(Integer(3), Integer(4), x)
48/(x^2 - 1)^2
>>> gen_legendre_Q(Integer(4), Integer(5), x)
-384/((x^2 - 1)^2*sqrt(-x^2 + 1))
>>> gen_legendre_Q(Integer(0), Integer(1), x)
-1/sqrt(-x^2 + 1)
>>> gen_legendre_Q(Integer(0), Integer(2), x)
-1/2*((x + 1)^2 - (x - 1)^2)/(x^2 - 1)
>>> gen_legendre_Q(Integer(2), Integer(2), x).subs(x=Integer(2)).expand()
9/2*I*pi - 9/2*log(3) + 14/3
```

```plaintext
class sage.functions.orthogonal_polys.Func_chebyshev_T
Bases: ChebyshevFunction
Chebyshev polynomials of the first kind.
REFERENCE:
• [AS1964] 22.5.31 page 778 and 6.1.22 page 256.
EXAMPLES:
```
```
```plaintext
sage: chebyshev_T(5, x)
˓→ # needs sage.symbolic
16*x^5 - 20*x^3 + 5*x
sage: var('k')
˓→ # needs sage.symbolic
k
sage: test = chebyshev_T(k, x); test
˓→ # needs sage.symbolic
chebyshev_T(k, x)
```
```
```
```plaintext
>>> from sage.all import *
>>> # needs sage.symbolic
>>> chebyshev_T(Integer(5), x)
˓→ # needs sage.symbolic
16*x^5 - 20*x^3 + 5*x
>>> var('k')
˓→ # needs sage.symbolic
k
>>> test = chebyshev_T(k, x); test
˓→ # needs sage.symbolic
chebyshev_T(k, x)
```
```
```plaintext
eval_algebraic(n, x)
Evaluate chebyshev_T as polynomial, using a recursive formula.
INPUT:
```
• \( n \) - an integer
• \( x \) - a value to evaluate the polynomial at (this can be any ring element)

EXAMPLES:

```python
sage: chebyshev_T.eval_algebraic(5, x) # needs sage.symbolic
2*(2*(2*x^2 - 1)*x - x)*(2*x^2 - 1) - x
sage: chebyshev_T(-7, x) - chebyshev_T(7, x) # needs sage.symbolic
0
sage: R.<t> = ZZ[]
sage: chebyshev_T.eval_algebraic(-1, t) t
sage: chebyshev_T.eval_algebraic(0, t)
1
sage: chebyshev_T.eval_algebraic(1, t) t
sage: chebyshev_T(7^100, 1/2)
1/2
sage: chebyshev_T(7^100, Mod(2,3))
2
sage: n = 97; x = RIF(pi/2/n) # needs sage.symbolic
sage: chebyshev_T(n, cos(x)) .contains_zero() # needs sage.symbolic
True
```

(continues on next page)
```python
>>> # needs sage.rings.padics
>>> R = Zp(Integer(2), Integer(8), 'capped-abs')['t']; (t,) = R._first_ngens(1)
>>> chebyshev_T(Integer(10)**Integer(6) + Integer(1), t)
(2^7 + O(2^8))*t^5 + O(2^8)*t^4 + (2^6 + O(2^8))*t^3 + O(2^8)*t^2
+ (1 + 2^6 + O(2^8))*t + O(2^8)
```

### eval_formula \((n, x)\)

Evaluate \(\text{chebyshev}_T\) using an explicit formula. See \([\text{AS1964}] 227\) (p. 782) for details for the recursions. See also \([\text{Koe1999}]\) for fast evaluation techniques.

**INPUT:**

- \(n\) – an integer
- \(x\) – a value to evaluate the polynomial at (this can be any ring element)

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: chebyshev_T.eval_formula(-1, x)*
x
sage: chebyshev_T.eval_formula(0, x)
1
sage: chebyshev_T.eval_formula(1, x)*
x
sage: chebyshev_T.eval_formula(10, x)
512*x^10 - 1280*x^8 + 1120*x^6 - 400*x^4 + 50*x^2 - 1
sage: chebyshev_T.eval_algebraic(10, x).expand()
512*x^10 - 1280*x^8 + 1120*x^6 - 400*x^4 + 50*x^2 - 1
sage: chebyshev_T.eval_formula(2, 0.1) == chebyshev_T._evalf_(2, 0.1)  # needs sage.rings.complex_double
True
```
REFERENCES:

- [AS1964] 22.8.3 page 783 and 6.1.22 page 256.

EXAMPLES:

```python
sage: R.<t> = QQ[]
sage: chebyshev_U(2, t)
4*t^2 - 1
sage: chebyshev_U(3, t)
8*t^3 - 4*t
```

```python
>>> from sage.all import *

R = QQ[t]; (t,) = R._first_ngens(1)

>>> chebyshev_U(Integer(2), t)
4*t^2 - 1

>>> chebyshev_U(Integer(3), t)
8*t^3 - 4*t
```

```python
eval_algebraic(n, x)
```

Evaluate `chebyshev_U` as a polynomial, using a recursive formula.

INPUT:

- `n` – an integer
- `x` – a value to evaluate the polynomial at (this can be any ring element)

EXAMPLES:

```python
sage: chebyshev_U.eval_algebraic(5, x) # needs sage.symbolic
-2*((2*x + 1)*(2*x - 1)*x - 4*(2*x^2 - 1)*x)*(2*x + 1)*(2*x - 1)

sage: parent(chebyshev_U(3, Mod(8,9)))
Ring of integers modulo 9

sage: parent(chebyshev_U(3, Mod(1,9)))
Ring of integers modulo 9

sage: chebyshev_U(-3, x) + chebyshev_U(1, x) # needs sage.symbolic
0

sage: chebyshev_U(-1, Mod(5,8))
0

sage: chebyshev_U.eval_algebraic(-2, t)
-1

sage: chebyshev_U.eval_algebraic(-1, t)
0

sage: chebyshev_U.eval_algebraic(0, t)
1

sage: chebyshev_U.eval_algebraic(1, t)
2*t

sage: n = 97; x = RIF(pi/n) # needs sage.symbolic

sage: chebyshev_U(n - 1, cos(x)).contains_zero() # needs sage.symbolic
```

(True)
Functions, Release 10.4

```
sage: # needs sage.rings.padics
sage: R.<t> = Zp(2, 6, 'capped-abs')[

sage: chebyshev_U(10^6 + 1, t)
(2 + O(2^6))*t + O(2^6)
```

```
>>> from sage.all import *

>>> chebyshev_U.eval_algebraic(Integer(5), x) ⨿
˓→ # needs sage.symbolic
-2*((2*x + 1)*(2*x - 1)*x - 4*(2*x^2 - 1)*x)*(2*x + 1)*(2*x - 1)

>>> parent(chebyshev_U(Integer(3), Mod(Integer(8),Integer(9))))
Ring of integers modulo 9

>>> parent(chebyshev_U(Integer(3), Mod(Integer(1),Integer(9))))
Ring of integers modulo 9

>>> chebyshev_U(-Integer(3), x) + chebyshev_U(Integer(1), x) ⨿
˓→ # needs sage.symbolic
0

>>> chebyshev_U(-Integer(1), Mod(Integer(5),Integer(8)))
0

>>> parent(chebyshev_U(-Integer(1), Mod(Integer(5),Integer(8))))
Ring of integers modulo 8

>>> R = ZZ['t']; (t,) = R._first_ngens(1)

>>> chebyshev_U.eval_algebraic(-Integer(2), t)
-1

>>> chebyshev_U.eval_algebraic(-Integer(1), t)
0

>>> chebyshev_U.eval_algebraic(Integer(0), t)
1

>>> chebyshev_U.eval_algebraic(Integer(1), t)
2*t

>>> n = Integer(97); x = RIF(pi/n) ⨿
˓→ # needs sage.symbolic

>>> chebyshev_U(n - Integer(1), cos(x)).contains_zero() ⨿
˓→ # needs sage.symbolic
True

>>> # needs sage.rings.padics

>>> R = Zp(Integer(2), Integer(6), capped-abs)['t']; (t,) = R._first_ngens(1)

>>> chebyshev_U(Integer(10)**Integer(6) + Integer(1), t)
(2 + O(2^6))*t + O(2^6)
```

eval_formula (n, x)

Evaluate chebyshev_U using an explicit formula.

See [AS1964] 227 (p. 782) for details on the recursions. See also [Koe1999] for the recursion formulas.

INPUT:

• n – an integer

• x – a value to evaluate the polynomial at (this can be any ring element)

EXAMPLES:

```
sage: # needs sage.symbolic

sage: chebyshev_U.eval_formula(10, x)
1024*x^10 - 2304*x^8 + 1792*x^6 - 560*x^4 + 60*x^2 - 1

sage: chebyshev_U.eval_formula(-2, x)
```

class sage.functions.orthogonal_polys.Func_gen_laguerre

Bases: OrthogonalFunction

REFERENCE:


class sage.functions.orthogonal_polys.Func_hahn

Bases: OrthogonalFunction

Hahn polynomials $Q_k(x; a, b, n)$.

INPUT:

• $k$ – the degree
• $x$ – the independent variable $x$
• $a, b$ – the parameters $a, b$
• $n$ – the number of discrete points

EXAMPLES:

We verify the orthogonality for $n = 3$:

```python
class sage.functions.orthogonal_polys.Func_hahn

Bases: OrthogonalFunction

Hahn polynomials $Q_k(x; a, b, n)$.

INPUT:

• $k$ – the degree
• $x$ – the independent variable $x$
• $a, b$ – the parameters $a, b$
• $n$ – the number of discrete points

EXAMPLES:

We verify the orthogonality for $n = 3$:

```
....:         for k in range(n + 1)).expand().factor()
....:         for i in range(n+1) for j in range(n+1))
sage: M = M.factor()
sage: P = rising_factorial
sage: def diag(i, a, b, n):
....:         return ((-1)^i * factorial(i) * P(b + 1, i) * P(i + a + b + 1, n + 1)
....:             / (factorial(n) * (2*i + a + b + 1) * P(-n, i) * P(a + 1, i)))
sage: all(M[i,i] == diag(i, a, b, n) for i in range(3))
True
sage: all(M[i,j] == 0 for i in range(3) for j in range(3) if i != j)
True

>>> from sage.all import *
>>> # needs sage.symbolic
>>> n = Integer(2)
>>> a, b = SR.var('a,b')
>>> def rho(k, a, b, n):
...         return binomial(a + k, k) * binomial(b + n - k, n - k)
>>> M = matrix([[sum(rho(k, a, b, n)
...             * hahn(i, k, a, b, n) * hahn(j, k, a, b, n)
...             for k in range(n + Integer(1))).expand().factor()
...             for i in range(n+Integer(1))]
for j in range(n+Integer(1))])
>>> M = M.factor()
>>> P = rising_factorial
>>> def diag(i, a, b, n):
...         return ((-Integer(1))**i * factorial(i) * P(b + Integer(1), i) * P(i + a +
→b + Integer(1), n + Integer(1))
...             / (factorial(n) * (Integer(2)*i + a + b + Integer(1)) * P(-n, i) *
→P(a + Integer(1), i)))
>>> all(M[i,i] == diag(i, a, b, n) for i in range(Integer(3)))
True
>>> all(M[i,j] == Integer(0) for i in range(Integer(3)) for j in
→range(Integer(3)) if i != j)
True

**eval_formula** (*k, x, a, b, n*)

Evaluate self using an explicit formula.

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: k, x, a, b, n = var('k,x,a,b,n')
sage: Q2 = hahn.eval_formula(2, x, a, b, n).simplify_full()
sage: Q2.coefficient(x**2).factor()
(a + b + 4)*(a + b + 3)/((a + 2)*(a + 1)*(n - 1)*n)
sage: Q2.coefficient(x).factor()
-(2*a*n - a + b + 4*n)*(a + b + 3)/((a + 2)*(a + 1)*(n - 1)*n)
sage: Q2(x=0)
1
```

>>> from sage.all import *
>>> # needs sage.symbolic
>>> k, x, a, b, n = var('k,x,a,b,n')
>>> Q2 = hahn.eval_formula(Integer(2), x, a, b, n).simplify_full()
>>> Q2.coefficient(x**Integer(2)).factor()
(a + b + 4)*(a + b + 3)/((a + 2)*(a + 1)*(n - 1)*n)
```
eval_recursive \((k, x, a, b, n, *args, **kwds)\)

Return the Hahn polynomial \(Q_k(x; a, b, n)\) using the recursive formula.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: x, a, b, n = var('x,a,b,n')

sage: hahn.eval_recursive(0, x, a, b, n)
1
sage: hahn.eval_recursive(1, x, a, b, n)
-(a + b + 2)*x/((a + 1)*n) + 1
sage: bool(hahn(2, x, a, b, n) == hahn.eval_recursive(2, x, a, b, n))
True
sage: bool(hahn(3, x, a, b, n) == hahn.eval_recursive(3, x, a, b, n))
True
sage: bool(hahn(4, x, a, b, n) == hahn.eval_recursive(4, x, a, b, n))
True
sage: M = matrix([[Integer(-1)/Integer(2), -1], [Integer(1), 0]])

sage: ret = hahn.eval_recursive(2, M, Integer(1), Integer(2), n) .simplify_full().factor()  # needs sage.modules
sage: ret
```

```python
[1/4*(4*n^2 + 8*n - 19)/((n - 1)*n) 3/2*(4*n + 3)/((n - 1)*n)]
[ -3/2*(4*n + 3)/((n - 1)*n) (n^2 - n - 7)/((n - 1)*n)]
```
Bases: GinacFunction

Return the Hermite polynomial for integers \( n > -1 \).

REFERENCE:

- [AS1964] 22.5.40 and 22.5.41, page 779.

EXAMPLES:

```sage
# needs sage.symbolic
sage: x = PolynomialRing(QQ, 'x').gen()
sage: hermite(2, x)
4*x^2 - 2
sage: hermite(3, x)
8*x^3 - 12*x
sage: hermite(3, 2)
40
sage: S.<y> = PolynomialRing(RR)
sage: hermite(3, y)
8.00000000000000*y^3 - 12.0000000000000*y
sage: R.<x,y> = QQ[]
sage: hermite(3, y^2)
8*y^6 - 12*y^2
sage: w = var('w')
sage: hermite(3, 2*w)
64*w^3 - 24*w
sage: hermite(5, RealField(100)(pi))
5208.6167627118104649470287166

>>> from sage.all import *
>>> # needs sage.symbolic
>>> x = PolynomialRing(QQ, 'x').gen()
>>> hermite(Integer(2), x)
4*x^2 - 2
>>> hermite(Integer(3), x)
8*x^3 - 12*x
>>> hermite(Integer(3), Integer(2))
40
>>> S = PolynomialRing(RR, names=('y',)); (y,) = S._first_ngens(1)
>>> hermite(Integer(3), y)
8.00000000000000*y^3 - 12.00000000000000*y
>>> R = QQ['x, y']; (x, y,) = R._first_ngens(2)
>>> hermite(Integer(3), y**Integer(2))
8*y^6 - 12*y^2
>>> w = var('w')
>>> hermite(Integer(3), Integer(2)*w)
64*w^3 - 24*w
>>> hermite(Integer(5), RealNumber('3.1416'))
5208.69733891963
```

Check that Issue #17192 is fixed:
Functions, Release 10.4

sage: # needs sage.symbolic
sage: x = PolynomialRing(QQ, 'x').gen()
1
sage: hermite(-1, x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
sage: hermite(-7, x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
sage: hermite(0, x)
1
sage: hermite(-1, x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
sage: hermite(-7, x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
sage: m, x = SR.var('m, x')
sage: hermite(m, x).diff(m)
Traceback (most recent call last):
  ... 
RuntimeError: derivative w.r.t. to the index is not supported yet

>>> from sage.all import *
>>> # needs sage.symbolic
>>> x = PolynomialRing(QQ, 'x').gen()
>>> hermite(Integer(0), x)
1
>>> hermite(-Integer(1), x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
>>> hermite(-Integer(7), x)
Traceback (most recent call last):
  ... 
RuntimeError: hermite_eval: The index n must be a nonnegative integer
>>> m, x = SR.var('m, x')
>>> hermite(m, x).diff(m)
Traceback (most recent call last):
  ... 
RuntimeError: derivative w.r.t. to the index is not supported yet

class sage.functions.orthogonal_polys.Func_jacobi_P
Bases: OrthogonalFunction

Return the Jacobi polynomial \( P_n^{(a,b)}(x) \) for integers \( n > -1 \) and \( a \) and \( b \) symbolic or \( a > -1 \) and \( b > -1 \).

The Jacobi polynomials are actually defined for all \( a \) and \( b \). However, the Jacobi polynomial weight \((1-x)^a(1+x)^b\)
is not integrable for \( a \leq -1 \) or \( b \leq -1 \).

REFERENCE:

• Table on page 789 in [AS1964].

EXAMPLES:

sage: x = PolynomialRing(QQ, 'x').gen()
sage: jacobi_P(2, 0, 0, x)  # needs sage.libs.flint sage.symbolic
3/2*x^2 - 1/2
sage: jacobi_P(2, 1, 2, 1.2)  # needs sage.libs.flint
5.01000000000000
Functions, Release 10.4

```python
>>> from sage.all import *
>>> x = PolynomialRing(QQ, 'x').gen()
>>> jacobi_P(Integer(2), Integer(0), Integer(0), x)           # needs sage.libs.flint sage.symbolic
3/2*x^2 - 1/2
>>> jacobi_P(Integer(2), Integer(1), Integer(2), RealNumber('1.2'))  # needs sage.libs.flint
5.01000000000000
```

```python
class sage.functions.orthogonal_polys.Func_krawtchouk

Bases: OrthogonalFunction

Krawtchouk polynomials $K_j(x; n, p)$.

INPUT:

- $j$ – the degree
- $x$ – the independent variable $x$
- $n$ – the number of discrete points
- $p$ – the parameter $p$

See also:

sage.coding.delsarte_bounds.krawtchouk() $K_{n,q}^{m,q}(x)$, which are related by

$$(-q)^j K_{n,q}^{m,q-1}(x) = K_j(x; n, 1 - q).$$

EXAMPLES:

We verify the orthogonality for $n = 4$:

```python
sage: n = 4
sage: p = SR.var('p')  # needs sage.symbolic
sage: matrix([[sum(binomial(n,m) * p**m * (1-p)**(n-m) * krawtchouk(i,m,n,p) * krawtchouk(j,m,n,p)) for m in range(n+1)].expand().factor() for i in range(n+1)] for j in range(n+1)])
[ 1  0  0  0]
[ 0 -4*(p - 1)*p  0  0]
[ 0  0  6*(p - 1)^2*p^2  0]
[ 0  0  0  -4*(p - 1)^3*p^3]
[ 0  0  0  0  (p - 1)^4*p^4]
```
```
```
We verify the relationship between the Krawtchouk implementations:

```
sage: q = SR.var('q')
    # needs sage.symbolic
sage: all(codes.bounds.krawtchouk(n, 1/q, j, x)*(-q)**j
    # needs sage.symbolic
    ...: == krawtchouk(j, x, n, 1-q) for j in range(n+1))
True
```

```
>>> from sage.all import *
>>> q = SR.var('q')
    # needs sage.symbolic
>>> all(codes.bounds.krawtchouk(n, Integer(1)/q, j, x)*(-q)**j
    # needs sage.symbolic
    ... == krawtchouk(j, x, n, Integer(1)-q) for j in range(n+Integer(1)))
True
```

```
eval_formula(k, x, n, p)
```

Evaluate `self` using an explicit formula.

EXAMPLES:

```
sage: x, n, p = var('x,n,p')
    # needs sage.symbolic
sage: krawtchouk.eval_formula(3, x, n, p).expand().collect(x)
    # needs sage.symbolic
-1/6*n^3*p^3 + 1/2*n^2*p^3 - 1/3*n*p^3 - 1/2*(n*p - 2*p + 1)*x^2
+ 1/6*x^3 + 1/6*(3*n^2*p^2 - 9*n*p^2 + 3*n*p + 6*p^2 - 6*p + 2)*x
```

```
eval_recursive(j, x, n, p, *args, **kwds)
```

Return the Krawtchouk polynomial $K_j(x; n, p)$ using the recursive formula.

EXAMPLES:
sage: # needs sage.symbolic
sage: x, n, p = var('x,n,p')
sage: krawtchouk.eval_recursive(0, x, n, p)
1
sage: krawtchouk.eval_recursive(1, x, n, p)
-n*p + x
sage: krawtchouk.eval_recursive(2, x, n, p).collect(x)
1/2*n^2*p^2 + 1/2*n*(p - 1)*p - n*p^2 + 1/2*n*p
- 1/2*(2*n*p - 2*p + 1)*x + 1/2*x^2
sage: bool(krawtchouk.eval_recursive(2, x, n, p) == krawtchouk(2, x, n, p))
True
sage: bool(krawtchouk.eval_recursive(3, x, n, p) == krawtchouk(3, x, n, p))
True
sage: bool(krawtchouk.eval_recursive(4, x, n, p) == krawtchouk(4, x, n, p))
True
sage: M = matrix([[-1/2, -1], [1, 0]])

>>> from sage.all import *
>>> # needs sage.symbolic
>>> x, n, p = var('x,n,p')
>>> krawtchouk.eval_recursive(Integer(0), x, n, p)
1
>>> krawtchouk.eval_recursive(Integer(1), x, n, p)
-n*p + x
>>> krawtchouk.eval_recursive(Integer(2), x, n, p).collect(x)
1/2*n^2*p^2 + 1/2*n*(p - 1)*p - n*p^2 + 1/2*n*p
- 1/2*(2*n*p - 2*p + 1)*x + 1/2*x^2
>>> bool(krawtchouk.eval_recursive(Integer(2), x, n, p) ==
   krawtchouk(Integer(2), x, n, p))
True
>>> bool(krawtchouk.eval_recursive(Integer(3), x, n, p) ==
   krawtchouk(Integer(3), x, n, p))
True
>>> bool(krawtchouk.eval_recursive(Integer(4), x, n, p) ==
   krawtchouk(Integer(4), x, n, p))
True

>>> M = matrix([[-Integer(1)/Integer(2), -Integer(1)], [Integer(1),
   Integer(0)]])
>>> # needs sage.modules
>>> krawtchouk.eval_recursive(Integer(2), M, Integer(3), Integer(1)/
   Integer(2))
# needs sage.modules

class sage.functions.orthogonal_polys.Func_laguerre
    Bases: OrthogonalFunction

REFERENCE:

class sage.functions.orthogonal_polys.Func_legendre_P
Bases: GinacFunction

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: legendre_P(4, 2.0)
55.3750000000000
sage: legendre_P(1, x)
x
sage: legendre_P(4, x + 1)
35/8*(x + 1)^4 - 15/4*(x + 1)^2 + 3/8
sage: legendre_P(1/2, I + 1.)
1.05338240025858 + 0.359890322109665*I
sage: legendre_P(0, SR(1)).parent()
Symbolic Ring
```

```python
sage: legendre_P(0, 0)  # needs sage.symbolic
1
```

```python
sage: legendre_P(1, x)  # needs sage.symbolic
x
```

```python
sage: # needs sage.symbolic
sage: legendre_P(4, 2.)
55.3750000000000
sage: legendre_P(5.5, 1.00001)
1.00017875754114
sage: legendre_P(1/2, I + 1).n()  # needs sage.symbolic
1.0533824002585801 + 0.35989032210966539*I
sage: legendre_P(42, RR(12345678))
2.66314881466753e309
sage: legendre_P(42, Reals(20)(12345678))
2.6632e309
sage: legendre_P(201/2, 0).n()  # needs sage.symbolic
0.056138617863017877699963095883
sage: legendre_P(201/2, 0).n(100)
0.056138617863017877699963095883
```

```python
sage: # needs sage.symbolic
sage: R.<x> = QQ[]
sage: legendre_P(4, x)
35/8*x^4 - 15/4*x^2 + 3/8
sage: legendre_P(10000, x).coefficient(x, 1)
0
sage: var('t,x')
(t, x)
sage: legendre_P(-5, t)
35/8*t^4 - 15/4*t^2 + 3/8
sage: legendre_P(4, x + 1)
35/8*(x + 1)^4 - 15/4*(x + 1)^2 + 3/8
sage: legendre_P(4, sqrt(2))
83/8
sage: legendre_P(4, I*e)
35/8*e^4 + 15/4*e^2 + 3/8
```

(continues on next page)
sage: # needs sage.symbolic
sage: n = var('n')
sage: derivative(legendre_P(n,x), x)
(n*x*legendre_P(n, x) - n*legendre_P(n - 1, x))/(x^2 - 1)
sage: derivative(legendre_P(3,x), x)
15/2*x^2 - 3/2
sage: derivative(legendre_P(n,x), n)
Traceback (most recent call last):
  ...:
RuntimeError: derivative w.r.t. to the index is not supported yet

>>> from sage.all import *

>>> # needs sage.symbolic

>>> legendre_P(Integer(4), RealNumber('2.0'))
55.3750000000000

>>> legendre_P(Integer(1), x)
x

>>> legendre_P(Integer(4), x + Integer(1))
35/8*(x + 1)^4 - 15/4*(x + 1)^2 + 3/8

>>> legendre_P(Integer(1)/Integer(2), I+RealNumber('1.'))
1.05338240025858 + 0.359890322109665*I

>>> legendre_P(Integer(0), SR(Integer(1))).parent()
Symbolic Ring

>>> legendre_P(Integer(0), Integer(0))
1

>>> legendre_P(Integer(1), x)
x

>>> # needs sage.symbolic

>>> legendre_P(Integer(4), RealNumber('2.'))
55.3750000000000

>>> legendre_P(Integer(1)/Integer(2), I + Integer(1)).n()
1.05338240025858 + 0.359890322109665*I

>>> legendre_P(Integer(1)/Integer(2), I + Integer(1)).n(Integer(59))
1.0533824002585801 + 0.35989032210966539*I

>>> legendre_P(Integer(42), RR(Integer(12345678)))
2.6632e309

>>> legendre_P(Integer(42), Reals(Integer(20))(Integer(12345678)))
2.6632e309

>>> legendre_P(Integer(201)/Integer(2), Integer(0)).n()
0.0561386178630179

>>> legendre_P(Integer(201)/Integer(2), Integer(0)).n(Integer(100))
0.056138617863017877699963095883

>>> # needs sage.symbolic

>>> R = QQ['x']; (x,) = R._first_ngens(1)

>>> legendre_P(Integer(4), x)
35/8*x^4 - 15/4*x^2 + 3/8

>>> legendre_P(Integer(10000), x).coefficient(x, Integer(1))
0

>>> var('t,x')
(continued from previous page)

```python
(t, x)
>>> legendre_P(-Integer(5), t)
35/8*t^4 - 15/4*t^2 + 3/8
>>> legendre_P(Integer(4), x + Integer(1))
35/8*(x + 1)^4 - 15/4*(x + 1)^2 + 3/8
>>> legendre_P(Integer(4), sqrt(Integer(2)))
83/8
>>> legendre_P(Integer(4), I*e)
35/8*e^4 + 15/4*e^2 + 3/8
>>> # needs sage.symbolic
>>> n = var('n')
>>> derivative(legendre_P(n,x), x)
(n*x*legendre_P(n, x) - n*legendre_P(n - 1, x))/(x^2 - 1)
>>> derivative(legendre_P(Integer(3),x), x)
15/2*x^2 - 3/2
>>> derivative(legendre_P(n,x), n)
Traceback (most recent call last):
  ... RuntimeError: derivative w.r.t. to the index is not supported yet
```

class sage.functions.orthogonal_polys.Func_legendre_Q
Bases: BuiltinFunction

EXAMPLES:

```python
sage: loads(dumps(legendre_Q))
legendre_Q
sage: maxima(legendre_Q(20, x, hold=True))._sage_().coefficient(x, 10)  # needs sage.symbolic
-29113619535/131072*log(-(x + 1)/(x - 1))
```

```python
>>> from sage.all import *
>>> loads(dumps(legendre_Q))
legendre_Q
>>> maxima(legendre_Q(Integer(20), x, hold=True))._sage_().coefficient(x,Integer(10))  # needs sage.symbolic
-29113619535/131072*log(-(x + 1)/(x - 1))
```

eval_formula (n, arg,**kwds)

Return expanded Legendre Q(n, arg) function expression.

REFERENCE:

• T.M. Dunster, Legendre and Related Functions, https://dlmf.nist.gov/14.7#E2

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: legendre_Q.eval_formula(1, x)
1/2*x*(log(x + 1) - log(-x + 1)) - 1
sage: legendre_Q.eval_formula(2, x).expand().collect(log(1+x)).collect(log(1-x))
1/4*(3*x^2 - 1)*log(x + 1) - 1/4*(3*x^2 - 1)*log(-x + 1) - 3/2*x
sage: legendre_Q.eval_formula(20, x).coefficient(x, 10)
-29113619535/131072*log(x + 1) + 29113619535/131072*log(-x + 1)
sage: legendre_Q(0, 2)
```

(continues on next page)
Functions, Release 10.4

-\frac{1}{2}i\pi + \frac{1}{2}\log(3)

sage: legendre_Q(0, 2.)
\rightarrow \text{needs mpmath}
0.549306144334055 - 1.57079632679490i

>>> from sage.all import *
>>> # needs sage.symbolic
>>> legendre_Q.eval_formula(Integer(1), x)
\rightarrow \text{needs sage.symbolic}
1/2*x*(\log(x + 1) - \log(-x + 1)) - 1

>>> legendre_Q.eval_formula(Integer(2), x).expand().
\rightarrow \text{collect(\log(Integer(1)+x)).collect(\log(Integer(1)-x))}
1/4*(3*x^2 - 1)*\log(x + 1) - 1/4*(3*x^2 - 1)*\log(-x + 1) - 3/2*x

>>> legendre_Q.eval_formula(Integer(20), x).coefficient(x, Integer(10))
29113619535/131072*\log(x + 1) + 29113619535/131072*\log(-x + 1)

>>> legendre_Q.eval_recursive(Integer(0), RealNumber('2.'))
\rightarrow \text{# needs mpmath}
0.549306144334055 - 1.57079632679490i

eval_recursive\ (n, arg, **kwds)

Return expanded Legendre \(Q(n, arg)\) function expression.

EXAMPLES:

sage: legendre_Q.eval_recursive(2, x)
\rightarrow \text{needs sage.symbolic}
3/4*x^2*(\log(x + 1) - \log(-x + 1)) - 3/2*x - 1/4*\log(x + 1) + 1/4*\log(-x + 1)

sage: legendre_Q.eval_recursive(20, x).expand().coefficient(x, 10)
\rightarrow \text{needs sage.symbolic}
-29113619535/131072*\log(x + 1) + 29113619535/131072*\log(-x + 1)

>>> from sage.all import *
>>> legendre_Q.eval_recursive(Integer(2), x)
\rightarrow \text{# needs sage.symbolic}
3/4*x^2*(\log(x + 1) - \log(-x + 1)) - 3/2*x - 1/4*\log(x + 1) + 1/4*\log(-x + 1)

>>> legendre_Q.eval_recursive(Integer(20), x).expand().coefficient(x, Integer(10))
\rightarrow \text{# needs sage.symbolic}
-29113619535/131072*\log(x + 1) + 29113619535/131072*\log(-x + 1)

class sage.functions.orthogonal_polys.Func_meixner

Meixner polynomials \(M_n(x; b, c)\).

INPUT:

- \(n\) – the degree
- \(x\) – the independent variable \(x\)
- \(b, c\) – the parameters \(b, c\)

eval_formula\ (n, x, b, c)

Evaluate \(\text{self}\) using an explicit formula.

EXAMPLES:
Functions, Release 10.4

>>> from sage.all import *
>>> x, b, c = var(’x,b,c’)  # needs sage.symbolic
>>> meixner.eval_recursive(0, x, b, c)
1
>>> meixner.eval_recursive(1, x, b, c)
-x^(1/c - 1) + b
>>> meixner.eval_recursive(2, x, b, c).simplify_full().collect(x)
-x^2*(2/c - 1/c^2 - 1) + b^2 + (2*b - 2*b/c - 1/c^2 + 1)*x + b
>>> bool(meixner(2, x, b, c) == meixner.eval_recursive(2, x, b, c))
True

>>> # needs sage.symbolic
>>> x, b, c = var(’x,b,c’)
>>> meixner.eval_recursive(3, x, b, c)
1
>>> meixner.eval_recursive(4, x, b, c)
-2*b + (2*b - 1)/c - 3/2/c^2 + 1/2 b^2 + b + 2/c - 1/c^2 - 1

1.8. Orthogonal polynomials 115
>>> bool(meixner(Integer(3), x, b, c) == meixner.eval_recursive(Integer(3), x, → b, c))
True
>>> bool(meixner(Integer(4), x, b, c) == meixner.eval_recursive(Integer(4), x, → b, c))
True
>>> M = matrix([[-Integer(1)/Integer(2), -Integer(1)],[Integer(1), → Integer(0)]])
>>> ret = meixner.eval_recursive(Integer(2), M, b, c).simplify_full().factor()
>>> for i in range(Integer(2)):
...     for j in range(Integer(2)):
...         ret[i, j] = ret[i, j].collect(c)
>>> ret
\[
\begin{bmatrix}
\frac{b^2}{c} + \frac{1}{2}(2b + 3)/c - \frac{1}{4}/c^2 - \frac{5}{4} - 2b + (2b - 1)/c + 3/2/c^2 - 1/2 \\
2b - (2b - 1)/c - 3/2/c^2 + 1/2 + b^2 + b + 2/c - 1/c^2 - 1
\end{bmatrix}
\]

\[
class \text{sage.functions.orthogonal_polys.Func_ultraspherical}
Bases: GinacFunction
Return the ultraspherical (or Gegenbauer) polynomial gegenbauer(n, a, x),
\[C_n^a(x) = \sum_{k=0}^{\lfloor n/2 \rfloor} (-1)^k \frac{\Gamma(n - k + a)}{\Gamma(a)k!(n - 2k)!}(2x)^{n-2k}.\]
When \(n\) is a nonnegative integer, this formula gives a polynomial in \(x\) of degree \(n\), but all parameters are permitted to be complex numbers. When \(a = 1/2\), the Gegenbauer polynomial reduces to a Legendre polynomial.
Computed using Pynac.
For numerical evaluation, consider using the \texttt{mpmath library}, as it also allows complex numbers (and negative \(n\) as well); see the examples below.
REFERENCE:
\textbullet{} [AS1964] 22.5.27
EXAMPLES:
\[
sage: \# \text{needs sage.symbolic}
sage: ultraspherical(8, 101/11, x)
795972057547264/214358881*x^8 - 62604543852032/19487171*x^6...
sage: x = PolynomialRing(QQ, 'x').gen()
sage: ultraspherical(2, 3/2, x)
15/2*x^2 - 3/2
sage: ultraspherical(1, 1, x)
2*x
sage: t = PolynomialRing(RationalField(), "t").gen()
sage: gegenbauer(3, 2, t)
32*t^3 - 12*t
sage: x = SR.var('x')
sage: n = ZZ.random_element(5, 5001)
sage: a = QQ.random_element().abs() + 5
sage: s = ( (n + 1)*ultraspherical(n + 1, a, x) 
\quad - 2*x*(n + a)*ultraspherical(n, a, x) 
\quad + (n + 2*a - 1)*ultraspherical(n - 1, a, x) )
sage: s.expand().is_zero()
True
sage: ultraspherical(5, 9/10, 3.1416)
\]
 sage: ultraspherical(5, 9/10, RealField(100)(pi))  
# needs sage.rings.real_mpfr
6949.4695419382702451843080687

 sage: # needs sage.symbolic
 sage: a, n = SR.var('a,n')
 sage: gegenbauer(2, a, x)
2*(a + 1)*a*x^2 - a
 sage: gegenbauer(3, a, x)
4/3*(a + 2)*(a + 1)*a*x^3 - 2*(a + 1)*a*x
 sage: gegenbauer(3, a, x).expand()
4/3*a^3*x^3 + 4*a^2*x^3 + 8/3*a*x^3 - 2*a^2*x - 2*a*x
 sage: gegenbauer(10, a, x).expand().coefficient(x, 2)
1/12*a^6 + 5/4*a^5 + 85/12*a^4 + 75/4*a^3 + 137/6*a^2 + 10*a
 sage: ex = gegenbauer(100, a, x)
 sage: (ex.subs(a==55/98) - gegenbauer(100, 55/98, x)).is_trivial_zero()
True

 sage: # needs sage.symbolic
 sage: gegenbauer(2, -3, x)
12*x^2 + 3
 sage: gegenbauer(120, -99/2, 3
1654502372608570682112687530178328494861923493372493824
 sage: gegenbauer(5, 9/2, x)
21879/8*x^5 - 6435/4*x^3 + 1287/8*x
 sage: gegenbauer(15, 3/2, x)
3903412392243800

 sage: derivative(gegenbauer(n, a, x), x)
# needs sage.symbolic
2*a*gegenbauer(n - 1, a + 1, x)
 sage: derivative(gegenbauer(n, a, x), x)
# needs sage.symbolic
4*(a + 2)*(a + 1)*a*x^2 - 2*(a + 1)*a
 sage: derivative(gegenbauer(n, a, x), a)
# needs sage.symbolic
Traceback (most recent call last):
... 
RuntimeError: derivative w.r.t. to the second index is not supported yet

>>> from sage.all import *
>>> # needs sage.symbolic
>>> ultraspherical(Integer(8), Integer(101)/Integer(11), x)
795972057547264/214358881*x^8 - 62604543852032/19487171*x^6...
>>> x = PolynomialRing(QQ, 'x').gen()
>>> ultraspherical(Integer(2), Integer(3)/Integer(2), x)
15/2*x^2 - 3/2
>>> ultraspherical(Integer(1), Integer(1), x)
2*x
>>> t = PolynomialRing(RationalField(), "t").gen()
>>> gegenbauer(Integer(3), Integer(2), t)
32*t^3 - 12*t
>>> x = SR.var('x')
>>> n = ZZ.random_element(Integer(5), Integer(5001))
>>> a = QQ.random_element().abs() + Integer(5)

```python
>>> s = ( (n + Integer(1))*ultraspherical(n + Integer(1), a, x)...
... - Integer(2)*x*(n + a)*ultraspherical(n, a, x)...
... + (n + Integer(2)*a - Integer(1))*ultraspherical(n - Integer(1), a, x) )
>>> s.expand().is_zero()
True
>>> ultraspherical(Integer(5), Integer(9)/Integer(10), RealNumber('3.1416'))
6949.55439044240
>>> ultraspherical(Integer(5), Integer(9)/Integer(10), RealField(Integer(100))(pi))  # needs sage.rings.
6949.4695419382704830380687
>>> # needs sage.symbolic
>>> a, n = SR.var('a,n')
>>> gegenbauer(Integer(2), a, x)
2*(a + 1)*a*x^2 - a
>>> gegenbauer(Integer(3), a, x)
4/3*(a + 2)*(a + 1)*a*x^3 - 2*(a + 1)*a*x
>>> gegenbauer(Integer(3), a, x).expand()  # needs sage.rings.
4/3*a^3*x^3 + 4*a^2*x^3 + 8*a*x^3 - 2*a^2*x - 2*a*x
>>> gegenbauer(Integer(10), a, x).expand().coefficient(x, Integer(2))
1/12*a^6 + 5/4*a^5 + 85/12*a^4 + 75/4*a^3 + 137/6*a^2 + 10*a
>>> ex = gegenbauer(Integer(100), a, x)
>>> (ex.subs(a==Integer(55)/Integer(98)) - gegenbauer(Integer(100), Integer(55)/Integer(98), x)).is_trivial_zero()  # needs sage.symbolic
True
>>> gegenbauer(Integer(2), -Integer(3), x)
12*x^2 + 3
>>> gegenbauer(Integer(120), -Integer(99)/Integer(2), Integer(3))
165450237608570682112687530178328494861923493372493824
>>> gegenbauer(Integer(5), Integer(9)/Integer(2), x)
21879/8*x^5 - 6435/4*x^3 + 1287/8*x
>>> gegenbauer(Integer(15), Integer(3)/Integer(2), Integer(5))
3903412392243800
>>> derivative(gegenbauer(n, a, x), a)  # needs sage.symbolic
Traceback (most recent call last):
  ... 
RuntimeError: derivative w.r.t. to the second index is not supported yet
```

Numerical evaluation with the mpmath library:

```python
sage: # needs mpmath
sage: from mpmath import gegenbauer as gegenbauer_mp
sage: from mpmath import mp
sage: mp.pretty = True; mp.dps=25
sage: gegenbauer_mp(-7,0.5,0.3)
0.1291811875
```

(continues on next page)
sage: gegenbauer_mp(2+3j, -0.75, -1000j)
(-5038991.358609026523401901 + 9414549.285447104177860806j)

>>> from sage.all import *

>>> from mpmath import gegenbauer as gegenbauer_mp
>>> from mpmath import mp

>>> mp.pretty = True; mp.dps=Integer(25)

>>> gegenbauer_mp(-Integer(7), 0.5, 0.3)
0.1291811875

>>> gegenbauer_mp(Integer(2)+ComplexNumber(0, 3), -0.75, -ComplexNumber(0, 1000))
(-5038991.358609026523401901 + 9414549.285447104177860806j)

class sage.functions.orthogonal_polys.OrthogonalFunction(name, nargs=2, latex_name=None, conversions=None)

Bases: BuiltinFunction

Base class for orthogonal polynomials.

This class is an abstract base class for all orthogonal polynomials since they share similar properties. The evaluation as a polynomial is either done via maxima, or with pynac.

Convention: The first argument is always the order of the polynomial, the others are other values or parameters where the polynomial is evaluated.

eval_formula(*args)

Evaluate this polynomial using an explicit formula.

EXAMPLES:

sage: from sage.functions.orthogonal_polys import OrthogonalFunction
sage: P = OrthogonalFunction('testo_P')

sage: P.eval_formula(1, 2.0)
Traceback (most recent call last):
...
NotImplementedError: no explicit calculation of values implemented

>>> from sage.all import *
>>> from sage.functions.orthogonal_polys import OrthogonalFunction

>>> P = OrthogonalFunction('testo_P')

>>> P.eval_formula(Integer(1), RealNumber('2.0'))
Traceback (most recent call last):
...
NotImplementedError: no explicit calculation of values implemented
1.9 Other functions

```python
class sage.functions.other.Function_Order
    Bases: GinacFunction
    
The order function.
    
This function gives the order of magnitude of some expression, similar to $O$-terms.
    
See also:
    
Order(), big_oh
    
EXAMPLES:

sage: x = SR('x')  # needs sage.symbolic
sage: x.Order()    # needs sage.symbolic
Order(x)

Order(x^2 + x)

>>> from sage.all import *
>>> x = SR('x')  # needs sage.symbolic
>>> x.Order()    # needs sage.symbolic
Order(x)

Order(x^2 + x)
```

```python
class sage.functions.other.Function_abs
    Bases: GinacFunction
    
The absolute value function.
    
EXAMPLES:

sage: abs(-2)
2

sage: # needs sage.symbolic
sage: var('x y')
(x, y)
sage: abs(x)
abs(x)
sage: abs(x^2 + y^2)
abs(x^2 + y^2)
sage: sqrt(x^2)
sqrt(x^2)
sage: abs(sqrt(x))
sqrt(abs(x))
sage: complex(abs(3*I))
(3+0j)
```

(continues on next page)
sage: f = sage.functions.other.Function_abs()  
sage: latex(f)  
\mathrm{abs}  
sage: latex(abs(x))  
\left| x \right|  
sage: abs(x)._sympy_()  
\text{Abs}(x)

>>> from sage.all import *  
>>> abs(-Integer(2))  
2  
>>> # needs sage.symbolic  
>>> var('x y')  
(x, y)  
>>> abs(x)  
abs(x)  
>>> abs(x**Integer(2) + y**Integer(2))  
abs(x^2 + y^2)  
>>> sqrt(x**Integer(2))  
\sqrt{x^2}  
>>> abs(sqrt(x))  
\text{abs}(\sqrt{x})  
>>> complex(abs(Integer(3)*I))  
(3+0j)

>>> f = sage.functions.other.Function_abs()  
>>> latex(f)  
\mathrm{abs}  

Test pickling:

sage: loads(dumps(abs(x)))  
needs sage.symbolic  
abs(x)

>>> from sage.all import *  
>>> loads(dumps(abs(x)))  
needs sage.symbolic  
abs(x)

class sage.functions.other.Function_arg

Bases: BuiltinFunction

The argument function for complex numbers.

EXAMPLES:
Functions, Release 10.4

\[
\begin{align*}
\text{sage: } & \text{# needs sage.symbolic} \\
\text{sage: } & \text{arg}(3+i) \\
\text{arctan}(1/3) \\
\text{sage: } & \text{arg}(-1+i) \\
3/4\pi \\
\text{sage: } & \text{arg}(2+2i) \\
1/4\pi \\
\text{sage: } & \text{arg}(2+x) \\
\text{arg}(x + 2) \\
\text{sage: } & \text{arg}(2.0+i+x) \\
\text{arg}(x + 2.000000000000000 + 1.000000000000000*I) \\
\text{sage: } & \text{arg}(-3) \\
\pi \\
\text{sage: } & \text{arg}(3) \\
0 \\
\text{sage: } & \text{arg}(0) \\
0 \\
\text{sage: } & \text{# needs sage.symbolic} \\
\text{sage: } & \text{latex(arg(x))} \\
\{\text{\texttt{\textbackslash \texttt{arg}}}\left(x\right)\} \\
\text{sage: } & \text{maxima(arg(x))} \\
\text{atan2}(0,_{\text{\texttt{\textbackslash \texttt{SAGE\_VAR\_x}}}}) \\
\text{sage: } & \text{maxima(arg(2+i))} \\
\text{atan}(1/2) \\
\text{sage: } & \text{maxima(arg(sqrt(2)+i))} \\
\text{atan}(1/sqrt(2)) \\
\text{sage: } & \text{arg(x)._sympy_()} \\
\text{# needs sympy} \\
\text{arg}(x) \\
\text{sage: } & \text{arg}(2+i) \\
\text{# needs sage.symbolic} \\
\text{arctan}(1/2) \\
\text{sage: } & \text{arg}(sqrt(2)+i) \\
\text{# needs sage.symbolic} \\
\text{arg}(sqrt(2) + I) \\
\text{sage: } & \text{arg}(sqrt(2)+i).\text{simplify()}} \\
\text{# needs sage.symbolic} \\
\text{arctan}(1/2*sqrt(2)) \\
\end{align*}
\]

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> arg(Integer(3)+i)
arctan(1/3)
>>> arg(-Integer(1)+i)
3/4*pi
>>> arg(Integer(2)+Integer(2)*i)
1/4*pi
>>> arg(Integer(2)+x)
arg(x + 2)
>>> arg(RealNumber('2.0')+i+x)
arg(x + 2.000000000000000 + 1.000000000000000*I)
>>> arg(-Integer(3))
pi
>>> arg(Integer(3))
0
```

(continues on next page)
functions

Functions, Release 10.4

>>> arg(Integer(0))
0

>>> # needs sage.symbolic
>>> latex(arg(x))
(\text{arg}(x))

>>> maxima(arg(Integer(2)+i))
atan(1/2)

>>> maxima(arg(sqrt(Integer(2))+i))
atan(1/sqrt(2))

>>> arg(x)._sympy_()
# needs sympy
arg(x)

>>> arg(Integer(2)+i) ˓→ # needs sage.symbolic
arctan(1/2)

>>> arg(sqrt(Integer(2))+i) ˓→ # needs sage.symbolic
arg(sqrt(2) + I)

>>> arg(sqrt(Integer(2))+i).simplify() ˓→ # needs sage.symbolic
arctan(1/2*sqrt(2))

class sage.functions.other.Function_binomial
Bases: GinacFunction

Return the binomial coefficient

\binom{x}{m} = \frac{x(x-1)\cdots(x-m+1)}{m!}

which is defined for \( m \in \mathbb{Z} \) and any \( x \). We extend this definition to include cases when \( x - m \) is an integer but \( m \) is not by

\binom{x}{m} = \binom{x}{x-m} \quad \text{if} \quad x - m \in \mathbb{Z}

If \( m < 0 \), return 0.

INPUT:

- \( x, m \) – numbers or symbolic expressions. Either \( m \) or \( x-m \) must be an integer, else the output is symbolic.

OUTPUT: number or symbolic expression (if input is symbolic)

EXAMPLES:

sage: # needs sage.symbolic
sage: binomial(5, 2)
10
sage: binomial(2, 0)
1
sage: binomial(1/2, 0) ˓→ # needs sage.libs.pari
1
sage: binomial(3, -1)

(continues on next page)
0
sage: binomial(20, 10)
184756
sage: binomial(-2, 5)
-6
sage: n = var('n'); binomial(n, 2)
1/2*(n - 1)*n
sage: n = var('n'); binomial(n, n)
1
sage: n = var('n'); binomial(n, n - 1)
n
sage: binomial(2^100, 2^100)
1
sage: binomial(RealField()(2.5), 2)
# needs sage.rings.real_mpfr
1.87500000000000
>>> from sage.all import *
>>>
# needs sage.symbolic
>>>
binomial(Integer(5), Integer(2))
10
>>>
binomial(Integer(2), Integer(0))
1
>>>
binomial(Integer(1)/Integer(2), Integer(0))
1/2
>>>
# needs sage.libs.pari
binomial(Integer(3), -Integer(1))
0
>>>
binomial(Integer(20), Integer(10))
184756
>>>
binomial(-Integer(2), Integer(5))
-6
>>>
# needs sage.symbolic
n = var('n'); binomial(n, Integer(2))
1/2*(n - 1)*n
>>>
# needs sage.libs.pari
n = var('n'); binomial(n, n)
1
>>>
# needs sage.libs.pari
n = var('n'); binomial(n, n - Integer(1))
n
>>>
# needs sage.rings.real_mpfr
binomial(Integer(2)**Integer(100), Integer(2)**Integer(100))
1
>>>
# needs sage.rings.real_mpfr
binomial(RealField()(2.5), Integer(2))
1.87500000000000

sage: k, i = var('k,i')
# needs sage.symbolic
sage: binomial(k,i)
# needs sage.symbolic
binomial(k, i)

>>> from sage.all import *
>>> k, i = var('k,i')
# needs sage.symbolic
>>> binomial(k,i)
# needs sage.symbolic
We can use a hold parameter to prevent automatic evaluation:

```python
sage: SR(5).binomial(3, hold=True)  # needs sage.symbolic
binomial(5, 3)
sage: SR(5).binomial(3, hold=True).simplify()  # needs sage.symbolic
10
```

```python
>>> from sage.all import *
>>> SR(Integer(5)).binomial(Integer(3), hold=True)  # needs sage.symbolic
binomial(5, 3)
>>> SR(Integer(5)).binomial(Integer(3), hold=True).simplify()  # needs sage.symbolic
10
```

```python
class sage.functions.other.Function_cases
    Bases: GinacFunction

    Formal function holding (condition, expression) pairs.

    Numbers are considered conditions with zero being False. A true condition marks a default value. The function is not evaluated as long as it contains a relation that cannot be decided by Pynac.

    EXAMPLES:

    ```python
    sage: # needs sage.symbolic
    sage: ex = cases(((x == 0, pi), (True, 0))); ex
    cases(((x == 0, pi), (1, 0)))
    sage: ex.subs(x==0)
    pi
    sage: ex.subs(x==2)
    0
    sage: ex + 1
    cases(((x == 0, pi), (1, 0))) + 1
    sage: _.subs(x==0)
    pi + 1
    >>> from sage.all import *
    >>> # needs sage.symbolic
    >>> ex = cases(((x==Integer(0), pi), (True, Integer(0)))); ex
    cases(((x == 0, pi), (1, 0)))
    >>> ex.subs(x==Integer(0))
    pi
    >>> ex.subs(x=Integer(2))
    0
    >>> ex + Integer(1)
    cases(((x == 0, pi), (1, 0))) + 1
    >>> _.subs(x=Integer(0))
    pi + 1
    ```
```

The first encountered default is used, as well as the first relation that can be trivially decided:

```python
```

1.9. Other functions
```python
sage: cases(((True, pi), (True, 0)))  # needs sage.symbolic
pi

sage: # needs sage.symbolic
sage: _ = var('y')
sage: ex = cases(((x==0, pi), (y==1, 0))); ex
cases(((x == 0, pi), (y == 1, 0)))
sage: ex.subs(x==0)
pi
sage: ex.subs(x==0, y==1)
pi
```

```python
>>> from sage.all import *

>>> cases(((True, pi), (True, Integer(0))))  # needs sage.symbolic
pi

>>> # needs sage.symbolic

>>> _ = var('y')

>>> ex = cases(((x==Integer(0), pi), (y==Integer(1), Integer(0)))); ex
cases(((x == 0, pi), (y == 1, 0)))

>>> ex.subs(x==Integer(0))
pi

>>> ex.subs(x==Integer(0), y==Integer(1))
pi
```

```python
class sage.functions.other.Function_ceil

Bases: BuiltinFunction

The ceiling function.

The ceiling of \( x \) is computed in the following manner.

1. The \( x.ceil() \) method is called and returned if it is there. If it is not, then Sage checks if \( x \) is one of Python's native numeric data types. If so, then it calls and returns \( \text{Integer(math.ceil(x))} \).

2. Sage tries to convert \( x \) into a RealIntervalField with 53 bits of precision. Next, the ceilings of the endpoints are computed. If they are the same, then that value is returned. Otherwise, the precision of the RealIntervalField is increased until they do match up or it reaches bits of precision.

3. If none of the above work, Sage returns a Expression object.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: a = ceil(2/5 + x); a
ceil(x + 2/5)
sage: a(x=4)
5
sage: a(x=4.0)
5
sage: ZZ(a(x=3))
4
sage: a = ceil(x^3 + x + 5/2); a
ceil(x^3 + x + 1/2) + 2
```
(continues on next page)
sage: a(x=2)
13

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> a = ceil(Integer(2)/Integer(5) + x); a
      ceil(x + 2/5)
>>> a(x=Integer(4))
5
>>> a(x=RealNumber('4.0'))
5
>>> ZZ(a(x=Integer(3)))
4
>>> a = ceil(x**Integer(3) + x + Integer(5)/Integer(2)); a
ceil(x^3 + x + 1/2) + 2
>>> a.simplify()
ceil(x^3 + x + 1/2) + 2
>>> a(x=Integer(2))
13
```

```python
sage: ceil(sin(8)/sin(2))  # needs sage.symbolic
# →
2
```  

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> ceil(sin(Integer(8))/sin(Integer(2)))  # needs sage.symbolic
2
```

```python
sage: ceil(5.4)
6
sage: type(ceil(5.4))
<class 'sage.rings.integer.Integer'>
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> ceil(RedNumber('5.4'))
6
>>> type(ceil(RedNumber('5.4')))  # needs sage.symbolic
<class 'sage.rings.integer.Integer'>
```

```python
sage: ceil(factorial(Integer(50))/exp(Integer(1)))  # needs sage.symbolic
11188719610782480504630258070757734324011354208865721592720336801
sage: ceil(SR(10^50 + 10^-50))  # needs sage.symbolic
100000000000000000000000000000000000000000000000000
sage: ceil(SR(10^50 - 10^-50))  # needs sage.symbolic
100000000000000000000000000000000000000000000000000
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
```
Functions, Release 10.4

(continued from previous page)

```python
>>> ceil(SR(Integer(10)**Integer(50) + Integer(10)**(-Integer(50))))
˓→ # needs sage.symbolic
100000000000000000000000000000000000000000000000001

>>> ceil(SR(Integer(10)**Integer(50) - Integer(10)**(-Integer(50))))
˓→ # needs sage.symbolic
100000000000000000000000000000000000000000000000000
```

Small numbers which are extremely close to an integer are hard to deal with:

```python
sage: ceil((33^100 + 1)^(1/100))
˓→ needs sage.symbolic
Traceback (most recent call last):
... ValueError: cannot compute ceil(...) using 256 bits of precision

>>> from sage.all import *

```python
>>> ceil(Integer(33)**Integer(100) + Integer(1))**(Integer(1)/Integer(100))
˓→ # needs sage.symbolic
Traceback (most recent call last):
... ValueError: cannot compute ceil(...) using 256 bits of precision
```

This can be fixed by giving a sufficiently large `bits` argument:

```python
sage: ceil((33^100 + 1)^(1/100), bits=Integer(500))
˓→ needs sage.symbolic
Traceback (most recent call last):
... ValueError: cannot compute ceil(...) using 512 bits of precision

sage: ceil((33^100 + 1)^(1/100), bits=Integer(1000))  #...
34
```

```python
>>> from sage.all import *

```python
```python
>>> ceil((Integer(33)**Integer(100) + Integer(1))**(Integer(1)/Integer(100)),
˓→ bits=Integer(500))
˓→ needs sage.symbolic
Traceback (most recent call last):
... ValueError: cannot compute ceil(...) using 512 bits of precision

>>> ceil((Integer(33)**Integer(100) + Integer(1))**(Integer(1)/Integer(100)),
˓→ bits=Integer(1000))  # needs sage.symbolic
34
```

```python
sage: ceil(sec(e))
˓→ needs sage.symbolic
-1
```

```python
sage: latex(ceil(x))
˓→ needs sage.symbolic
\left \lfloor x \right \rceil
sage: ceiling(x)
˓→ needs sympy sage.symbolic
```

128 Chapter 1. Built-in Functions
>>> from sage.all import *
>>> ceil(sec(e)) # needs sage.symbolic
-1

>>> latex(ceil(x)) # needs sage.symbolic
\left \lceil x \right \rceil

>>> ceiling(x) # needs sympy sage.symbolic

sage: import numpy
# needs numpy
sage: a = numpy.linspace(0,2,6) # needs numpy
sage: ceil(a) # needs numpy
array([0., 1., 1., 2., 2., 2.])

>>> from sage.all import *
>>> import numpy # needs numpy
>>> a = numpy.linspace(Integer(0),Integer(2),Integer(6)) # needs numpy
>>> ceil(a) # needs numpy
array([0., 1., 1., 2., 2., 2.])

Test pickling:

sage: loads(dumps(ceil))
cell

>>> from sage.all import *
>>> loads(dumps(ceil)) # needs

class sage.functions.other.Function_conjugate
Bases: GinacFunction

Returns the complex conjugate of the input.

It is possible to prevent automatic evaluation using the hold parameter:

sage: conjugate(I, hold=True) # needs sage.symbolic
conjugate(I)

>>> from sage.all import *
>>> conjugate(I, hold=True) # needs sage.symbolic
conjugate(I)

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():
Functions, Release 10.4

```python
sage: conjugate(I, hold=True).simplify() # needs sage.symbolic
-I
```

```python
>>> from sage.all import *
>>> conjugate(I, hold=True).simplify() # needs...
sage: symbolic
-I
```

class `sage.functions.other.Function_crootof`  
Bases: `BuiltinFunction`  

Formal function holding `(polynomial, index)` pairs.  
The expression evaluates to a floating point value that is an approximation to a specific complex root of the polynomial. The ordering is fixed so you always get the same root.  
The functionality is imported from SymPy, see [http://docs.sympy.org/latest/_modules/sympy/polys/rootoftools.html](http://docs.sympy.org/latest/_modules/sympy/polys/rootoftools.html)  

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: c = complex_root_of(x^6 + x + 1, 1); c
complex_root_of(x^6 + x + 1, 1)
sage: c.n()
-0.790667188814418 + 0.300506920309552*I
sage: c.n(100)
-0.79066718881441764449859281847 + 0.30050692030955162512001002521*I
sage: (c^6 + c + 1).n(100) < 1e-25
True
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> c = complex_root_of(x**Integer(6) + x + Integer(1), Integer(1)); c
complex_root_of(x^6 + x + 1, 1)
>>> c.n()
-0.790667188814418 + 0.300506920309552*I
>>> c.n(Integer(100))
-0.79066718881441764449859281847 + 0.30050692030955162512001002521*I
>>> (c**Integer(6) + c + Integer(1)).n(Integer(100)) < RealNumber('1e-25')
True
```

class `sage.functions.other.Function_elementof`  
Bases: `BuiltinFunction`  

Formal set membership function that is only accessible internally.  
This function is called to express a set membership statement, usually as part of a solution set returned by `solve()`.  
See `sage.sets.set.Set` and `sage.sets.real_set.RealSet` for possible set arguments.  

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: from sage.functions.other import element_of
sage: element_of(x, SR(ZZ))
element_of(x, Integer Ring)
```

(continues on next page)
element_of(sin(x), Rational Field)
sage: element_of(x, SR(RealSet.open_closed(0,1)))
element_of(x, (0, 1])
sage: element_of(x, SR(Set([4,6,8])))
element_of(x, {8, 4, 6})

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.functions.other import element_of
>>> element_of(x, SR(ZZ))
```

```python
>>> element_of(x, Integer Ring)
```

```python
>>> element_of(sin(x), SR(QQ))
```

```python
>>> element_of(sin(x), Rational Field)
```

```python
>>> element_of(x, SR(RealSet.open_closed(Integer(0),Integer(1))))
```

```python
>>> element_of(x, (0, 1])
```

```python
>>> element_of(x, SR(Set([Integer(4),Integer(6),Integer(8)])))
```

```python
>>> element_of(x, {8, 4, 6})
```

class sage.functions.other.Function_factorial

Bases: GinacFunction

Returns the factorial of \( n \).

INPUT:

- \( n \) – a non-negative integer, a complex number (except negative integers) or any symbolic expression

OUTPUT: an integer or symbolic expression

EXAMPLES:

```python
sage: factorial(0)
1
sage: factorial(4)
24
sage: factorial(10)
3628800
sage: factorial(6) == 6*5*4*3*2
True

sage: # needs sage.symbolic
sage: x = SR.var('x')
sage: f = factorial(x + factorial(x)); f
factorial(x + factorial(x))
sage: f(x=3)
362880
sage: factorial(x)^2
factorial(x)^2
```

```python
>>> from sage.all import *
>>> factorial(Integer(0))
1
>>> factorial(Integer(4))
24
>>> factorial(Integer(10))
3628800
>>> factorial(Integer(6)) ==
```
\begin{align*}
\rightarrow \text{Integer}(6) \times \text{Integer}(5) \times \text{Integer}(4) \times \text{Integer}(3) \times \text{Integer}(2) \\
\text{True}
\end{align*}

>>> # needs sage.symbolic
>>> x = SR.var('x')
>>> f = factorial(x + factorial(x)); f
factorial(x + factorial(x))
>>> f(x=Integer(3))
362880
>>> factorial(x)**Integer(2)
factorial(x)^2

To prevent automatic evaluation use the \texttt{hold} argument:

sage: factorial(5, hold=True) # needs sage.symbolic
\rightarrow factorial(5)

>>> from sage.all import *

sage: factorial(Integer(5), hold=True) # needs sage.symbolic
\rightarrow factorial(5)

To then evaluate again, we currently must use Maxima via \texttt{sage.symbolic.expression.Expression.simplify()}:

sage: factorial(5, hold=True).simplify() # needs sage.symbolic
\rightarrow 120

We can also give input other than nonnegative integers. For other nonnegative numbers, the \texttt{sage.functions.gamma.gamma()} function is used:

sage: factorial(1/2) # needs sage.symbolic
1/2*sqrt(pi)
sage: factorial(3/4) # needs sage.symbolic
gamma(7/4)
sage: factorial(2.3) # needs sage.symbolic
2.68343738195577

>>> from sage.all import *

\rightarrow 1/2*sqrt(pi)
\rightarrow gamma(7/4)
But negative input always fails:

```python
>>> factorial(-32)
Traceback (most recent call last):
... ValueError: factorial only defined for non-negative integers
```
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Code</th>
</tr>
</thead>
</table>
| sage: # needs sage.symbolic | sage: var('x')
|x |
| sage: a = floor(5.25 + x); a | floor(x + 5.250000000000000)
| sage: a.simplify() | floor(x + 0.25) + 5
| sage: a(x=2) | 7 |

```python
ground from sage.all import *
ground floor(RealNumber('5.4'))
5
ground type(floor(RealNumber('5.4')))
<class 'sage.rings.integer.Integer'>

ground # needs sage.symbolic
sage: var('x')
|x |
sage: a = floor(RealNumber('5.25') + x); a
floor(x + 5.250000000000000)
```
```python
ground a.simplify()
floor(x + 0.25) + 5
ground a(x=Integer(2))
7
```

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Code</th>
</tr>
</thead>
</table>
| sage: # needs sage.symbolic | sage: floor(cos(8) / cos(2))
0 |
| sage: floor(log(4) / log(2))
2 | |
| sage: a = floor(5.4 + x); a | floor(x + 5.400000000000000)
| sage: a.subs(x==2) |
7 |
| sage: floor(log(2^(3/2)) / log(2) + 1/2) | 2 |
| sage: floor(log(2^(-3/2)) / log(2) + 1/2) | -1 |

```python
ground from sage.all import *
ground # needs sage.symbolic
sage: floor(cos(Integer(8)) / cos(Integer(2)))
0
ground floor(log(Integer(4)) / log(Integer(2)))
2
ground a = floor(RealNumber('5.4') + x); a
floor(x + 5.400000000000000)
```
```python
ground a.subs(x==Integer(2))
7
ground floor(log(Integer(2)**(Integer(3)/Integer(2))) / log(Integer(2)) + Integer(1)/
Integer(2))
2
ground floor(log(Integer(2)**(-Integer(3)/Integer(2))) / log(Integer(2)) + Integer(1)/
Integer(2))
-1
```
Small numbers which are extremely close to an integer are hard to deal with:

```python
sage: floor((33^100 + 1)^(1/100))  # needs sage.symbolic
Traceback (most recent call last):
  ...
ValueError: cannot compute floor(...) using 256 bits of precision
```

This can be fixed by giving a sufficiently large `bits` argument:

```python
sage: floor((33^100 + 1)^(1/100), bits=500)  # needs sage.symbolic
Traceback (most recent call last):
  ...
ValueError: cannot compute floor(...) using 512 bits of precision
```

```python
sage: floor((33^100 + 1)^(1/100), bits=1000)  # needs sage.symbolic
33
```

```python
sage: floor((33^100 + 1)^(1/100), bits=500)  # needs sage.symbolic
33
```

```python
sage: floor((33^100 + 1)^(1/100), bits=1000)  # needs sage.symbolic
33
```
ValueError: cannot compute floor(...) using 512 bits of precision

```python
>>> floor((Integer(33)**Integer(100) + Integer(1))**(Integer(1)/Integer(100)),
   bits=Integer(1000))
# needs sage.symbolic
```

```python
33
```

```python
sage: import numpy
# needs numpy
```

```python
sage:
   a = numpy.linspace(0,2,6)
# needs numpy
```

```python
sage:
   floor(a)
# needs numpy
```

```python
array([0., 0., 0., 1., 1., 2.])
```

```python
sage:
   floor(x)._sympy_()
# needs sympy sage.symbolic
```

```python
floor(x)
```

```python
>>> from sage.all import *
>>> import numpy
# needs numpy
```

```python
>>> a = numpy.linspace(Integer(0),Integer(2),Integer(6))
# needs numpy
```

```python
>>> floor(a)
# needs numpy
```

```python
array([0., 0., 0., 1., 1., 2.])
```

```python
>>> floor(x)._sympy_()
# needs sympy sage.symbolic
```

Test pickling:

```python
sage: loads(dumps(floor))
floor
```

```python
>>> from sage.all import *
>>> loads(dumps(floor))
floor
```

```python
class sage.functions.other.Function_frac

Bases: BuiltinFunction

The fractional part function \( \{x\} \).

\( \text{frac}(x) \) is defined as \( \{x\} = x - \lfloor x \rfloor \).

EXAMPLES:

```python
sage: frac(5.4)
# needs sage.rings.real_mpfr
```

```python
0.400000000000000
```

```python
sage: type(frac(5.4))
# needs sage.rings.real_mpfr
```

```python
<class 'sage.rings.real_mpfr.RealNumber'>
```

```python
sage: frac(Integer(456)/Integer(123))
```

```python
29/41
```

```python
sage: # needs sage.symbolic
```

(continues on next page)
sage: var('x')
x
sage: a = frac(5.4 + x); a
frac(x + 5.40000000000000)
sage: frac(cos(8)/cos(2))
cos(8)/cos(2)
sage: latex(frac(x))
\operatorname{frac}(x)
sage: frac(x)._sympy_()  # needs sympy
frac(x)

>>> from sage.all import *

Test pickling:

```python
sage: loads(dumps(floor))
floor
```

```python
>>> from sage.all import *

Test pickling:

```python
sage: loads(dumps(floor))
floor
```

class sage.functions.other.Function_imag_part

    Bases: GinacFunction

    Returns the imaginary part of the (possibly complex) input.

    It is possible to prevent automatic evaluation using the hold parameter:

```python
sage: imag_part(I, hold=True)  # needs sage.symbolic
imag_part(I)
```
>>> from sage.all import *
>>> imag_part(I, hold=True)    # needs...
---sage.symbolic
imag_part(I)

To then evaluate again, we currently must use Maxima via sage.symbolic.expression.Expression.simplify():

sage: imag_part(I, hold=True).simplify()    #...
---needs sage.symbolic
1

class sage.functions.other.Function_limit
Bases: BuiltinFunction

Placeholder symbolic limit function that is only accessible internally.

This function is called to create formal wrappers of limits that Maxima can’t compute:

sage: a = lim(exp(x^2)*(1-erf(x)), x=+infinity); a      #...
---needs sage.symbolic
-limit((erf(x) - 1)*e^(x^2), x, +Infinity)

EXAMPLES:

sage: # needs sage.symbolic
sage: from sage.functions.other import symbolic_limit as slimit
sage: slimit(1/x, x, +oo)  # needs sage.symbolic
limit(1/x, x, +Infinity)

---

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```python
limit(1/x, x, +Infinity)
```  
```python
>>> slimit(Integer(1)/x, x, Integer(0), plus)
```  
```python
limit(1/x, x, 0, plus)
```  
```python
>>> slimit(Integer(1)/x, x, Integer(0), minus)
```  
```python
limit(1/x, x, 0, minus)
```

```python
class sage.functions.other.Function_prod
    Bases: BuiltinFunction

    Placeholder symbolic product function that is only accessible internally.

    EXAMPLES:
    ```
sage: from sage.functions.other import symbolic_product as sprod
    sage: r = sprod(x, x, 1, 10); r
    # needs sage.symbolic
    product(x, x, 1, 10)
    sage: r.unhold()
    # needs sage.symbolic
    3628800
    ```
```

```python
>>> from sage.all import *
```  
```python
>>> from sage.functions.other import symbolic_product as sprod
```  
```python
>>> r = sprod(x, x, Integer(1), Integer(10)); r
```  
```python
>>> r.unhold()
```

```python
class sage.functions.other.Function_real_nth_root
    Bases: BuiltinFunction

    Real \(n\)-th root function \(x^\frac{1}{n}\).

    The function assumes positive integer \(n\) and real number \(x\).

    EXAMPLES:
    ```
sage: real_nth_root(2, 3)
    # needs sage.symbolic
    2^{(1/3)}
    sage: real_nth_root(-2, 3)
    # needs sage.symbolic
    -2^{(1/3)}
    sage: real_nth_root(8, 3)
    2
    sage: real_nth_root(-8, 3)
    -2
    sage: real_nth_root(-2, 4)
    Traceback (most recent call last):
    ...
    ValueError: no real nth root of negative real number with even n
    ```
```

```python
>>> from sage.all import *
```  
```python
>>> real_nth_root(Integer(2), Integer(3))
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

\[
\begin{align*}
2^{1/3} \\
\text{real_nth_root(-Integer(2), Integer(3))} \\
-2^{1/3} \\
\text{real_nth_root(Integer(8), Integer(3))} \\
2 \\
\text{real_nth_root(-Integer(8), Integer(3))} \\
-2 \\
\text{real_nth_root(-Integer(2), Integer(4))} \\
\text{Traceback (most recent call last):} \\
\text{...} \\
\text{ValueError: no real nth root of negative real number with even n}
\end{align*}
\]

For numeric input, it gives a numerical approximation.

\[
\begin{align*}
sage: \text{real_nth_root}(2., 3) \\
\text{1.25992104989487} \\
sage: \text{real_nth_root}(-2., 3) \\
\text{1.25992104989487}
\end{align*}
\]

Some symbolic calculus:

\[
\begin{align*}
sage: \text{f = real_nth_root(x, 5)^3; f} \\
\text{real_nth_root(x^3, 5)} \\
sage: \text{f.diff()} \\
3/5*x^2*real_nth_root(x^{-12}, 5) \\
sage: \text{result = f.integrate(x)} \\
\text{...} \\
sage: \text{result} \\
\text{integrate((abs(x)^3)^(1/5)*sgn(x^3), x)} \\
sage: \text{_.diff()} \\
\text{(abs(x)^3)^(1/5)*sgn(x^3)}
\end{align*}
\]

(continues on next page)
class sage.functions.other.Function_real_part

Bases: GinacFunction

Returns the real part of the (possibly complex) input.

It is possible to prevent automatic evaluation using the hold parameter:

```python
sage: real_part(I, hold=True)  # needs sage.symbolic
real_part(I)
```

To then evaluate again, we currently must use Maxima via `sage.symbolic.expression.Expression.simplify()`:

```python
sage: real_part(I, hold=True).simplify()  # needs sage.symbolic
0
```

EXAMPLES:

```python
sage: z = 1+2*I  # needs sage.symbolic
sage: real(z)   # needs sage.symbolic
1
sage: real(5/3)  # needs sage.symbolic
5/3
sage: a = 2.5
sage: real(a)   # needs sage.rings.real_mpfr
2.5
sage: type(real(a))   # needs sage.rings.real_mpfr
<class 'sage.rings.real_mpfr.RealLiteral'>
```

```python
sage: z = Integer(1)+Integer(2)*I  # needs sage.symbolic
sage: real(z)  # needs sage.symbolic
1.9
```
Sage can recognize some expressions as real and accordingly return the identical argument:

```python
sage: # needs sage.symbolic
sage: SR.var('x', domain='integer').real_part()
x
sage: SR.var('x', domain='integer').imag_part()
0
sage: real_part(sin(x)+x)
x + sin(x)
sage: real_part(x*exp(x))
x*e^x
sage: imag_part(sin(x)+x)
0
sage: real_part(real_part(x))
x
sage: forget()
```

```python
>>> from sage.all import *

```
This module provides easy access to many of Maxima and PARI’s special functions.

Maxima’s special functions package (which includes spherical harmonic functions, spherical Bessel functions (of the 1st and 2nd kind), and spherical Hankel functions (of the 1st and 2nd kind)) was written by Barton Willis of the University of Nebraska at Kearney. It is released under the terms of the General Public License (GPL).

Support for elliptic functions and integrals was written by Raymond Toy. It is placed under the terms of the General Public License (GPL) that governs the distribution of Maxima.

Next, we summarize some of the properties of the functions implemented here.

- **Spherical harmonics**: Laplace’s equation in spherical coordinates is:

\[
\frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial f}{\partial r} \right) + \frac{1}{r^2 \sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial f}{\partial \theta} \right) + \frac{1}{r^2 \sin^2 \theta} \frac{\partial^2 f}{\partial \varphi^2} = 0.
\]

Note that the spherical coordinates \( \theta \) and \( \varphi \) are defined here as follows: \( \theta \) is the colatitude or polar angle, ranging from \( 0 \leq \theta \leq \pi \) and \( \varphi \) the azimuth or longitude, ranging from \( 0 \leq \varphi < 2\pi \).

The general solution which remains finite towards infinity is a linear combination of functions of the form

\[
r^{-1-\ell} \cos(m\varphi) P^m_\ell \left( \cos \theta \right)
\]

and

\[
r^{-1-\ell} \sin(m\varphi) P^m_\ell \left( \cos \theta \right)
\]

where \( P^m_\ell \) are the associated Legendre polynomials (cf. \texttt{Func_assoc_legendre_P}), and with integer parameters \( \ell \geq 0 \) and \( -\ell \leq m \leq \ell \). Put in another way, the solutions with integer parameters \( \ell \geq 0 \) and \( -\ell \leq m \leq \ell \), can be written as linear combinations of:
\[ U_{\ell,m}(r, \theta, \varphi) = r^{-1-\ell} Y_{\ell}^m(\theta, \varphi) \]

where the functions \( Y \) are the spherical harmonic functions with parameters \( \ell, m \), which can be written as:

\[
Y_{\ell}^m(\theta, \varphi) = \sqrt{\frac{(2\ell + 1)(\ell - m)!}{4\pi(\ell + m)!}} e^{im\varphi} P_{\ell}^m(\cos \theta).
\]

The spherical harmonics obey the normalisation condition

\[
\int_{\theta=0}^{\pi} \int_{\varphi=0}^{2\pi} Y_{\ell}^m Y_{\ell'}^{m'}* d\Omega = \delta_{\ell\ell'}\delta_{mm'}, \quad d\Omega = \sin \theta \, d\varphi \, d\theta.
\]

- The **incomplete elliptic integrals** (of the first kind, etc.) are:

\[
\int_0^\phi \frac{1}{\sqrt{1 - m \sin(x)^2}} \, dx,
\int_0^\phi \frac{1}{\sqrt{1 - m \sin(x)^2}} \, dx,
\int_0^\phi \frac{1}{\sqrt{1 - m t^2}} \, dx,
\int_0^\phi \frac{1}{\sqrt{1 - n \sin(x)^2}} \, dx
\]

and the complete ones are obtained by taking \( \phi = \pi/2 \).

**Warning:** SciPy’s versions are poorly documented and seem less accurate than the Maxima and PARI versions. Typically they are limited by hardware floats precision.

REFERENCES:

- Abramowitz and Stegun: *Handbook of Mathematical Functions* [AS1964]
- Wikipedia article Spherical_harmonics
- Wikipedia article Helmholtz_equation
- Online Encyclopedia of Special Functions

AUTHORS:

- David Joyner (2006-13-06): initial version
- David Joyner (2006-30-10): bug fixes to pari wrappers of Bessel functions, hypergeometric_U
- David Joyner (2008-02-16): optional calls to scipy and replace all \#random by ...
• Eviatar Bach (2013): making elliptic integrals symbolic
• Eric Gourgoulhon (2022): add Condon-Shortley phase to spherical harmonics

class sage.functions.special.EllipticE
Bases: BuiltinFunction

Return the incomplete elliptic integral of the second kind:

\[ E(\varphi | m) = \int_0^\varphi \sqrt{1 - m \sin^2 x} \, dx. \]

EXAMPLES:

```python
sage: z = var("z")
# needs sage.symbolic
sage: elliptic_e(z, 1)
# needs sage.symbolic
2*round(z/pi) - sin(pi*round(z/pi) - z)
```

```python
sage: elliptic_e(z, 0)
# needs sage.symbolic
z
```

```python
sage: elliptic_e(0.5, 0.1) # abs tol 2e-15
# needs mpmath
0.498011394498832
```

```python
sage: elliptic_e(1/2, 1/10).n(200) # needs sage.symbolic
0.4980113944988315331154610406...
```

See also:

• Taking \( \varphi = \pi/2 \) gives \( \text{elliptic_ec}() \).
• Taking \( \varphi = \arcsin(\text{sn}(u, m)) \) gives \( \text{elliptic_eu}() \).

REFERENCES:

• Wikipedia article Elliptic_integral#Incomplete_elliptic_integral_of_the_second_kind

1.10. Miscellaneous special functions 145
Functions, Release 10.4

- Wikipedia article Jacobi_elliptic_functions

```python
class sage.functions.special.EllipticEC

Bases: BuiltinFunction

Return the complete elliptic integral of the second kind:

\[
E(m) = \int_0^{\pi/2} \sqrt{1 - m \sin(x)^2} \, dx.
\]

EXAMPLES:

```sage`

sage: elliptic_ec(0.1)  
# needs mpmath
1.53075763689776

sage: elliptic_ec(x).diff()  
# needs sage.symbolic
1/2*(elliptic_ec(x) - elliptic_kc(x))/x
```

See also:

- `elliptic_e()`.

REFERENCES:

- Wikipedia article Elliptic_integral#Complete_elliptic_integral_of_the_second_kind

```python
class sage.functions.special.EllipticEU

Bases: BuiltinFunction

Return Jacobi’s form of the incomplete elliptic integral of the second kind:

\[
E(u, m) = \int_0^u \frac{dn(x, m)^2}{dx} \, dx = \int_0^\tau \frac{\sqrt{1 - mx^2}}{\sqrt{1 - x^2}} \, dx.
\]

where \( \tau = \text{sn}(u, m) \).

Also, `elliptic_eu(u, m) = elliptic_e(asin(sn(u,m)),m)`.

EXAMPLES:

```sage`

```sage: from sage.all import *`
```

```sage`

```sage: elliptic_eu(RealNumber('0.5'), RealNumber('0.1'))  
# needs mpmath
0.496054551286597
```

See also:
• `elliptic_e()`.

REFERENCES:
• Wikipedia article Elliptic_integral#Incomplete_elliptic_integral_of_the_second_kind
• Wikipedia article Jacobi_elliptic_functions

class `sage.functions.special.EllipticF`
Bases: `BuiltinFunction`

Return the incomplete elliptic integral of the first kind.

\[ F(\varphi | m) = \int_0^\varphi \frac{dx}{\sqrt{1 - m \sin(x)^2}}. \]

Taking \( \varphi = \pi/2 \) gives `elliptic_kc()`.

EXAMPLES:

```python
sage: z = var("z")
# needs sage.symbolic
sage: elliptic_f(z, 0) # needs sage.symbolic
z
sage: elliptic_f(z, 1).simplify() # needs sage.symbolic
log(tan(1/4*pi + 1/2*z))
```

```python
>>> from sage.all import *
```

```python
>>> z = var("z") # needs sage.symbolic
```

```python
>>> elliptic_f(z, Integer(0)) # needs sage.symbolic
z
```

```python
>>> elliptic_f(z, Integer(1)).simplify() # needs sage.symbolic
log(tan(1/4*pi + 1/2*z))
```

```python
>>> elliptic_f(RealNumber('0.2'), RealNumber('0.1')) # needs mpmath
```

See also:
• `elliptic_e()`.

REFERENCES:
• Wikipedia article Elliptic_integral#Incomplete_elliptic_integral_of_the_first_kind

class `sage.functions.special.EllipticKC`
Bases: `BuiltinFunction`

Return the complete elliptic integral of the first kind:

\[ K(m) = \int_0^{\pi/2} \frac{dx}{\sqrt{1 - m \sin(x)^2}}. \]
EXAMPLES:

```python
sage: elliptic_kc(0.5)  # needs mpmath
1.85407467730137
```

```python
>>> from sage.all import *
>>> elliptic_kc(RealNumber('0.5'))  # needs mpmath
1.85407467730137
```

See also:

- elliptic_f()
- elliptic_ec()

REFERENCES:

- Wikipedia article Elliptic_integral#Complete_elliptic_integral_of_the_first_kind
- Wikipedia article Elliptic_integral#Incomplete_elliptic_integral_of_the_first_kind

```python
class sage.functions.special.EllipticPi
```

Return the incomplete elliptic integral of the third kind:

\[
\Pi(n, t, m) = \int_0^t \frac{dx}{(1 - n \sin^2(x))^{1/2} \sqrt{1 - m \sin^2(x)}}.
\]

INPUT:

- \( n \) – a real number, called the “characteristic”
- \( t \) – a real number, called the “amplitude”
- \( m \) – a real number, called the “parameter”

EXAMPLES:

```python
sage: N(elliptic_pi(1, pi/4, 1))  # needs sage.symbolic
1.14779357469632
```

```python
>>> from sage.all import *
>>> N(elliptic_pi(Integer(1), pi/Integer(4), Integer(1)))  # needs sage.symbolic
1.14779357469632
```

Compare the value computed by Maxima to the definition as a definite integral (using GSL):

```python
sage: elliptic_pi(0.1, 0.2, 0.3)  # needs mpmath
0.200665068220979
```

```python
sage: numerical_integral(1/(1-0.1*sin(x)^2)/sqrt(1-0.3*sin(x)^2), 0.0, 0.2)  # needs sage.symbolic
(0.2006650682209791, 2.227829789769088e-15)
```
>>> from sage.all import *
>>> elliptic_pi(RealNumber('0.1'), RealNumber('0.2'), RealNumber('0.3'))
0.200665068220979
>>> numerical_integral(Integer(1)/(Integer(1)-RealNumber('0.1'))
*(Integer(2)/sqrt(Integer(1)-RealNumber('0.3'))*sin(x)**Integer(2)),
RealNumber('0.0'), RealNumber('0.2'))
(0.200650682209791, 2.227829789769088e-15)

REFERENCES:

• Wikipedia article Elliptic_integral#Incomplete_elliptic_integral_of_the_third_kind

class sage.functions.special.SphericalHarmonic

Bases: BuiltinFunction

Returns the spherical harmonic function $Y_m^n(\theta, \varphi)$.

For integers $n > -1, |m| \leq n$, simplification is done automatically. Numeric evaluation is supported for complex $n$ and $m$.

EXAMPLES:

sage: # needs sage.symbolic
sage: x, y = var('x, y')
sage: spherical_harmonic(3, 2, x, y)
1/8*sqrt(30)*sqrt(7)*cos(x)*e^(2*I*y)*sin(x)^2/sqrt(pi)
sage: spherical_harmonic(3, 2, 1, 2)
1/8*sqrt(30)*sqrt(7)*cos(1)*e^(4*I)*sin(1)^2/sqrt(pi)
sage: spherical_harmonic(Integer(3), Integer(2), Integer(1), Integer(2))
-0.351154337307488 - 0.415562233975369*I
sage: latex(spherical_harmonic(Integer(3), Integer(2), x, y, hold=True))
Y_{3}^{2}\left(x, y\right)
sage: spherical_harmonic(Integer(1), Integer(2), x, y)
0

>>> from sage.all import *
>>> # needs sage.symbolic
>>> x, y = var('x, y')
>>> spherical_harmonic(Integer(3), Integer(2), x, y)
1/8*sqrt(30)*sqrt(7)*cos(x)*e^(2*I*y)*sin(x)^2/sqrt(pi)
>>> spherical_harmonic(Integer(3), Integer(2), Integer(1), Integer(2))
1/8*sqrt(30)*sqrt(7)*cos(1)*e^(4*I)*sin(1)^2/sqrt(pi)
>>> spherical_harmonic(Integer(3) + I, RealNumber('2.'), Integer(1), Integer(2))
-0.351154337307488 - 0.415562233975369*I
>>> latex(spherical_harmonic(Integer(3), Integer(2), x, y, hold=True))
Y_{3}^{2}\left(x, y\right)
>>> spherical_harmonic(Integer(1), Integer(2), x, y)
0

The degree $n$ and the order $m$ can be symbolic:

sage: # needs sage.symbolic
sage: n, m = var('n m')
sage: spherical_harmonic(n, m, x, y)
spherical_harmonic(n, m, x, y)
sage: latex(spherical_harmonic(n, m, x, y))
Y_{n}^{m}\left(x, y\right)
sage: diff(spherical_harmonic(n, m, x, y), x)
m*cot(x)*spherical_harmonic(n, m, x, y)
+ sqrt(-(m + n + 1)*(m - n))*e^(-I*y)*spherical_harmonic(n, m + 1, x, y)
sage: diff(spherical_harmonic(n, m, x, y), y)
I*m*spherical_harmonic(n, m, x, y)

The convention regarding the Condon-Shortley phase $(-1)^m$ is the same as for SymPy’s spherical harmonics and Wikipedia article Spherical_harmonics:

sage: # needs sage.symbolic
sage: spherical_harmonic(1, 1, x, y)
-1/4*sqrt(3)*sqrt(2)*e^(I*y)*sin(x)/sqrt(pi)

sage: from sympy import Ynm
sage: Ynm(1, 1, x, y).expand(func=True)
-sqrt(6)*exp(I*y)*sin(x)/(4*sqrt(pi))

sage: spherical_harmonic(1, 1, x, y) - Ynm(1, 1, x, y)
0

It also agrees with SciPy’s spherical harmonics:

sage: spherical_harmonic(1, 1, pi/2, pi).n()  # abs tol 1e-14
0.345494149471335

sage: from scipy.special import sph_harm  # NB: arguments x and y are swapped
sage: sph_harm(1, 1, pi/2).n()  # abs tol 1e-14
Note that this convention differs from the one in Maxima, as revealed by the sign difference for odd values of \( m \):

\[
\text{sage: maxima.spherical_harmonic(1, 1, x, y).sage() \quad \# \quad \text{needs sage.symbolic}}
\]
\[
\quad 1/2*sqrt(3/2)*e^{i*y}*\sin(x)/sqrt(pi)
\]

It follows that, contrary to Maxima, SageMath uses the same sign convention for spherical harmonics as SymPy, SciPy, Mathematica and Wikipedia article Table of spherical harmonics.

REFERENCES:

- Wikipedia article Spherical harmonics

sage.functions.special.elliptic_eu_f \((u, m)\)

Internal function for numeric evaluation of \( E(\text{am}(u, m)|m) \), where \( E \) is the incomplete elliptic integral of the second kind and \( \text{am} \) is the Jacobi amplitude function.

EXAMPLES:

\[
\text{sage: from sage.functions.special import elliptic_eu_f}
\]
\[
\text{sage: elliptic_eu_f(0.5, 0.1) \quad \# \quad \text{needs mpmath}}
\]
\[
\quad mpf('0.49605455128659691')
\]

\[
\text{sage.functions.special.elliptic_j}(z, \text{prec}=53)
\]

Returns the elliptic modular \( j \)-function evaluated at \( z \).

INPUT:

- \( z \) (complex) – a complex number with positive imaginary part.
- \( \text{prec} \) (default: 53) – precision in bits for the complex field.

OUTPUT:

(complex) The value of \( j(z) \).
ALGORITHM:
Calls the pari function ellj().

AUTHOR:
John Cremona

EXAMPLES:

```python
sage: elliptic_j(CC(i)) #...
1728.00000000000
sage: elliptic_j(sqrt(-2.0)) #...
800.00000000000
sage: z = ComplexField(100)(1, sqrt(11))/2 #...

sage: elliptic_j(z) #...
-32768.000...
sage: elliptic_j(z).real().round() #...
-32768
sage: tau = (1 + sqrt(-163))/2 #...

sage: (-elliptic_j(tau.n(100)).real().round())^(1/3) #...
640320
```

This example shows the need for higher precision than the default one of the ComplexField, see Issue #28355:
1.11 Hypergeometric functions

This module implements manipulation of infinite hypergeometric series represented in standard parametric form (as $\mathbf{pFq}$ functions).

AUTHORS:

- Fredrik Johansson (2010): initial version
- Eviatar Bach (2013): major changes

EXAMPLES:

Examples from Issue #9908:

```python
sage: # needs sage.symbolic
sage: maxima('integrate(bessel_j(2, x), x)').sage()
1/24*x^3*hypergeometric((3/2,), (5/2, 3), -1/4*x^2)
```

```python
sage: sum(((2*I)^x/(x^3 + 1)*(1/4)^x), x, 0, oo)
hypergeometric((1, 1, -1/2*I*sqrt(3) - 1/2, 1/2*I*sqrt(3) - 1/2),... (2, -1/2*I*sqrt(3) + 1/2, 1/2*I*sqrt(3) + 1/2), 1/2*I)
```

```python
res = sum((-1)^x/((2*x + 1)*factorial(2*x + 1)), x, 0, oo)
sage: res
hypergeometric((1/2,), (3/2, 3/2), -1/4)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> -elliptic_j(tau, 75)  # rel tol 1e-2
2.62537412640768000000e17 - 0.0001309913593909879441262*I
>>> -elliptic_j(tau, 100)  # rel tol 1e-2
2.6253741264076799999999999999e17 - 1.3012822400356887122945119790e-12*I
>>> (-elliptic_j(tau, 100).real().round())**(1/3)
640320
```

1.11. Hypergeometric functions
Functions, Release 10.4

(continued from previous page)

```python
>>> res = hypergeometric((1/2,), (3/2, 3/2), -1/4)
>>> res in [hypergeometric((Integer(1)/Integer(2)), (Integer(3)/Integer(2),...
-Integer(3)/Integer(2)), -Integer(1)/Integer(4)), sin_integral(Integer(1))]  
True
```

Simplification (note that `simplify_full` does not yet call `simplify_hypergeometric`):

```python
sage: # needs sage.symbolic
sage: hypergeometric([-2], [], x).simplify_hypergeometric()
x^2 - 2*x + 1
sage: hypergeometric([], [], x).simplify_hypergeometric()
e^x
sage: a = hypergeometric((hypergeometric(()), (), x), (),
    ..., hypergeometric(()), (, x))
sage: a.simplify_hypergeometric()
1/((-e^x + 1)^e^x)
sage: a.simplify_hypergeometric(algorithm='sage')
1/((-e^x + 1)^e^x)
```

```python
>>> from sage.all import *
```

```
>>> # needs sage.symbolic
>>> hypergeometric([-Integer(2)], [], x).simplify_hypergeometric()
x^2 - 2*x + 1
>>> hypergeometric([], [], x).simplify_hypergeometric()
e^x
>>> a = hypergeometric((hypergeometric(()), (), x), (),
    ..., hypergeometric(()), (, x))
>>> a.simplify_hypergeometric()
1/((-e^x + 1)^e^x)
>>> a.simplify_hypergeometric(algorithm='sage')
1/((-e^x + 1)^e^x)
```

Equality testing:

```python
sage: bool(hypergeometric([], [], x).derivative(x) ==
    # needs sage.symbolic
    ..., hypergeometric([], [], x))  # diff(e^x, x) == e^x
True
sage: bool(hypergeometric([], [], x) == hypergeometric([], [1], x))  
    # needs sage.symbolic
   False
```

```python
>>> from sage.all import *
```

```python
>>> # needs sage.symbolic
>>> hypergeometric([], [], x).derivative(x) ==
    # needs sage.symbolic
    ..., hypergeometric([], [], x))  # diff(e^x, x) == e^x
True
>>> bool(hypergeometric([], [Integer(1)], x) ==
    # needs sage.symbolic
   False
```

Computing terms and series:
Functions, Release 10.4

>>> from sage.all import *
>>> # needs sage.symbolic
>>> var('z')

sage: hypergeometric([], [], z).series(z, 0)
Order(1)

sage: hypergeometric([], [], z).series(z, 1)
1 + Order(z)

sage: hypergeometric([], [], z).series(z, 2)
1 + 1*z + Order(z^2)

sage: hypergeometric([], [], z).series(z, 3)
1 + 1*z + 1/2*z^2 + Order(z^3)

sage: # needs sage.symbolic
sage: hypergeometric([-2], [], z).series(z, 3)
1 + (-2)*z + 1*z^2

sage: hypergeometric([-2], [], z).series(z, 6)
1 + (-2)*z + 1*z^2

sage: hypergeometric([-2], [], z).series(z, 6).is_terminating_series()
True

sage: hypergeometric([-2], [], z).series(z, 2)
1 + (-2)*z + Order(z^2)

sage: hypergeometric([-2], [], z).series(z, 2).is_terminating_series()
False

sage: hypergeometric([1], [], z).series(z, 6)
# needs sage.symbolic
1 + 1*z + 1*z^2 + 1*z^3 + 1*z^4 + 1*z^5 + Order(z^6)

sage: hypergeometric([1], [1/2], -z^2/4).series(z, 11)
# needs sage.symbolic
1 + (-1/2)*z^2 + 1/24*z^4 + (-1/720)*z^6 + 1/40320*z^8 +...
(-1/3628800)*z^10 + Order(z^11)

sage: sum(hypergeometric([1, 2], [3], 1/3).terms(6)).n()
# needs sage.symbolic
1.29788359788360

sage: hypergeometric([1, 2], [3], 1/3).n()
# needs sage.symbolic
1.29837194594696

sage: hypergeometric([], [], x).series(x, 20)(x=1).n() == e.n()
# needs sage.symbolic
True

1.11. Hypergeometric functions

(continues on next page)
1 + 1*z + 1/2*z^2 + Order(z^3)

```python
>>> # needs sage.symbolic
>>> hypergeometric([-Integer(2)], [], z).series(z, Integer(3))
```

1 + (-2)*z + 1*z^2

```python
>>> hypergeometric([-Integer(2)], [], z).series(z, Integer(6))
```

1 + (-2)*z + 1*z^2

```python
>>> hypergeometric([-Integer(2)], [], z).series(z, Integer(6)).is_terminating_series()
```

True

```python
>>> hypergeometric([-Integer(2)], [], z).series(z, Integer(2))
```

1 + (-2)*z + Order(z^2)

```python
>>> hypergeometric([-Integer(2)], [], z).series(z, Integer(2)).is_terminating_series()
```

False

```python
>>> hypergeometric([Integer(1)], [], z).series(z, Integer(6))
```

1 + 1*z + 1/2*z^2 + Order(z^3)

```python
>>> # needs sage.symbolic
>>> hypergeometric([Integer(1)], [], z).series(z, Integer(6))
```

1 + 1*z + 1/2*z^2 + 1/24*z^4 + (-1/720)*z^6 + 1/40320*z^8 +...

```python
>>> hypergeometric([1], [Integer(3)], x).series(x, Integer(5))
```

1 + 1/5*x + 1/240*x^3 + 1/6720*x^4 + Order(x^5)

```python
>>> sum(hypergeometric([Integer(1), Integer(2)], [Integer(3), Integer(3), Integer(3)], x).series(x, Integer(6))).n() # needs sage.symbolic
```

1.29788359788360

```python
>>> hypergeometric([1], [Integer(5)], x).series(x, Integer(20))(x=Integer(1)).n() == e.n() # needs sage.symbolic
```

True

Plotting:

```python
sage: # needs sage.symbolic
sage: f(x) = hypergeometric([1, 1], [3, 3, 3], x)
```

```python
sage: plot(f, x, -30, 30) # needs sage.plot
```

Graphics object consisting of 1 graphics primitive

```python
sage: g(x) = hypergeometric([x], [], 2)
```

```python
sage: complex_plot(g, (-1, 1), (-1, 1)) # needs sage.plot
```

Graphics object consisting of 1 graphics primitive

```python
>>> from sage.all import *
```

```python
>>> # needs sage.symbolic
>>> __tmp__=var("x"); f = symbolic_expression(hypergeometric([Integer(1), Integer(1)], [Integer(3), Integer(3), Integer(3)], x)).function(x)
```

```python
>>> plot(f, x, -Integer(30), Integer(30)) # needs sage.plot
```

Graphics object consisting of 1 graphics primitive

```python
>>> __tmp__=var("x"); g = symbolic_expression(hypergeometric([x], [], Integer(2))).
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

→ function(x)

>>> complex_plot(g, (-Integer(1), Integer(1)), (-Integer(1), Integer(1)))
# needs sage.plot

Graphics object consisting of 1 graphics primitive

Numeric evaluation:

sage: # needs sage.symbolic
sage: hypergeometric([1], [], 1/10).n()  # geometric series
1.11111111111111
sage: hypergeometric([1], [], 1).n()  # e
2.71828182845905
sage: hypergeometric([1], [3..], hold=True)
hypergeometric((), (3.00000000000000), 3.00000000000000)
sage: hypergeometric([1, 2, 3], [4, 5, 6], 1/2).n()
1.02573619590134
sage: hypergeometric([1, 2, 3], [4, 5, 6], 1/2).n(digits=30)
1.02573619590133865036584139535
sage: hypergeometric([5 - 3*I], [3/2, 2 + I, sqrt(2)], 4 + I).n()
5.5260511678803 - 7.86331357527540*I
sage: hypergeometric((10, 10), (50,), 2.).n()
-1705.75733163554 - 356.749986056024*I

>>> from sage.all import *

>>> # needs sage.symbolic

>>> hypergeometric([Integer(1)], [], Integer(1)/Integer(10)).n()  # geometric series
1.11111111111111
>>> hypergeometric([], [], Integer(1)).n()  # e
2.71828182845905
>>> hypergeometric([], [], RealNumber('3.')).n()  # geometric series
1.11111111111111
>>> hypergeometric([Integer(1), Integer(2), Integer(3)], [Integer(4), Integer(5), Integer(6)], Integer(1)/Integer(2)).n()  # geometric series
1.02573619590134

>>> hypergeometric([Integer(1), Integer(2), Integer(3)], [Integer(4), Integer(5), Integer(6)], Integer(1)/Integer(2)).n(digits=Integer(30))
1.02573619590133865036584139535

>>> hypergeometric([Integer(5) - Integer(3)*I], [Integer(3)/Integer(2), Integer(2) + Integer(1), sqrt(Integer(2))], Integer(4) + I).n()
5.5260511678803 - 7.86331357527540*I

>>> hypergeometric((Integer(10), Integer(10)), (Integer(50),), RealNumber('2.'))
-1705.75733163554 - 356.749986056024*I

Conversions:

sage: maxima(hypergeometric([1, 1, 1], [3, 3, 3], x))  # needs sage.symbolic
hypergeometric([1,1,1],[3,3,3],_SAGE_VAR_x)
sage: hypergeometric((5, 4), (4, 4), 3)._sympy_()  # needs sympy sage.symbolic
hyber((5, 4), (4, 4), 3)
sage: hypergeometric((5, 4), (4, 4), 3).mathematica_init_()  # needs sage.symbolic
'HypergeometricPFQ[{5,4},{4,4},3]'

1.11. Hypergeometric functions 157
>>> from sage.all import *
>>> maxima(hypergeometric([Integer(1), Integer(1), Integer(1)], [Integer(3), Integer(3), x]))  # needs sage.symbolic
hypergeometric([1, 1, 1], [3, 3, 3], _SAGE_VAR_x)

>>> hypergeometric((Integer(5), Integer(4)), (Integer(4), Integer(4)), Integer(3))._sympy_()  # needs sympy sage.symbolic
hyper((5, 4), (4, 4), 3)

>>> hypergeometric((Integer(5), Integer(4)), (Integer(4), Integer(4)), Integer(3))._mathematica_init_()  # needs sage.symbolic
HypergeometricPFQ[{5, 4}, {4, 4}, 3]

Arbitrary level of nesting for conversions:

sage: maxima(nest(lambda y: hypergeometric([y], [], x), 3, 1))  # needs sage.symbolic
1/(1-_SAGE_VAR_x)^(1/(1-_SAGE_VAR_x)^(1/(1-_SAGE_VAR_x)))

sage: maxima(nest(lambda y: hypergeometric([y], [3], x), 3, 1))._sage_()  # needs sage.symbolic
hypergeometric((hypergeometric((hypergeometric((1,), (3,), x),), (3,), x),), (3,), x)

sage: nest(lambda y: hypergeometric([y], [], x), 3, 1)._mathematica_init_()  # needs sage.symbolic
'HypergeometricPFQ[{HypergeometricPFQ[{HypergeometricPFQ[{1},{},x]},...]

The confluent hypergeometric functions can arise as solutions to second-order differential equations (example from here):

sage: var('m')  # needs sage.symbolic
m
sage: y = function('y')(x)  # needs sage.symbolic
sage: desolve(diff(y, x, 2) + 2*x*diff(y, x) - 4*m*y, y,  # needs sage.symbolic
....: contrib_ode=true, ivar=x)
[y(x) == _K1*hypergeometric_M(-m, 1/2, -x^2) +...
 _K2*hypergeometric_U(-m, 1/2, -x^2)]

(continues on next page)
... contrib_ode=true, ivar=x)
[y(x) == _K1*hypergeometric_M(-m, 1/2, -x^2) + ...
  _K2*hypergeometric_U(-m, 1/2, -x^2)]

Series expansions of confluent hypergeometric functions:

```
sage: hypergeometric_M(2, 2, x).series(x, 3)  # needs sage.symbolic
1 + 1*x + 1/2*x^2 + Order(x^3)

sage: hypergeometric_U(2, 2, x).series(x == 3, 100).subs(x=1).n()  # needs sage.symbolic
0.403652637676806

sage: hypergeometric_U(2, 2, 1).n()  # needs mpmath sage.symbolic
0.403652637676806
```

```python
>>> from sage.all import *

>>> hypergeometric_M(Integer(2), Integer(2), x).series(x, Integer(3))  # needs sage.symbolic
1 + 1*x + 1/2*x^2 + Order(x^3)

>>> hypergeometric_U(Integer(2), Integer(2), x).series(x == Integer(3), Integer(100)).
  subs(x=Integer(1)).n()  # needs sage.symbolic
0.403652637676806

>>> hypergeometric_U(Integer(2), Integer(2), Integer(1)).n()  # needs mpmath sage.symbolic
0.403652637676806
```

```python
class sage.functions.hypergeometric.Hypergeometric

   Bases: BuiltinFunction

   Represent a (formal) generalized infinite hypergeometric series.

   It is defined as

   \[
   \frac{\Gamma(a_1 + \cdots + a_p + b_1 + \cdots + b_q)}{\Gamma(a_1) \cdots \Gamma(a_p) 
   \Gamma(b_1) \cdots \Gamma(b_q) } \sum_{n=0}^{\infty} \frac{(a_1)_n \cdots (a_p)_n \ z^n}{n!}.
   \]

   where \((x)_n\) is the rising factorial.

   class EvaluationMethods

   Bases: object

   deflated(a, b, z)

   Rewrite as a linear combination of functions of strictly lower degree by eliminating all parameters \(a[i]\)
   and \(b[j]\) such that \(a[i] = b[i] + m\) for nonnegative integer \(m\).

   EXAMPLES:
```

```
sage: # needs sage.symbolic
sage: x = hypergeometric([6, 1], [3, 4, 5], 10)
sage: y = x.deflated(); y
1/252*hypergeometric((4,), (7, 8), 10)
  + 1/12*hypergeometric((3,), (6, 7), 10)
  + 1/2*hypergeometric((2,), (5, 6), 10)
  + hypergeometric((1,), (4, 5), 10)
sage: x.n(); y.n()
2.87893612686782
```

(continues on next page)
2.87893612686782

```python
sage: # needs sage.symbolic
sage: x = hypergeometric([6, 7], [3, 4, 5], 10)
sage: y = x.deflated(); y
25/27216*hypergeometric((), (11,), 10) +
25/648*hypergeometric((), (10,), 10) +
265/504*hypergeometric((), (9,), 10) +
181/63*hypergeometric((), (8,), 10) +
19/3*hypergeometric((), (7,), 10) +
5*hypergeometric((), (6,), 10) +
hypergeometric((), (5,), 10)
sage: x.n(); y .n()
63.0734110716969
63.0734110716969
```
is_absolutely_convergent \((a, b, z)\)

Determine whether self converges absolutely as an infinite series. False is returned if not all terms are finite.

EXAMPLES:

Degree giving infinite radius of convergence:

```python
sage: hypergeometric([2, 3], [4, 5], 6).is_absolutely_convergent() # undefined
True
sage: hypergeometric([2, 3], [-4, 5], 6).is_absolutely_convergent() # undefined
False
sage: (hypergeometric([2, 3], [-4, 5], Infinity).is_absolutely_convergent()) # undefined
False
```

Ordinary geometric series (unit radius of convergence):

```python
sage: # needs sage.symbolic
sage: hypergeometric([1], [], 1/2).is_absolutely_convergent()
```
True
sage: hypergeometric([1], [], 2).is_absolutely_convergent()
False
sage: hypergeometric([1], [], 1).is_absolutely_convergent()
False
sage: hypergeometric([1], [], -1).is_absolutely_convergent()
False
sage: hypergeometric([1], [], -1).n()
# Sum still exists
0.500000000000000

Degree $p = q + 1$ (unit radius of convergence):

sage: # needs sage.symbolic
sage: hypergeometric([2, 3], [4], 6).is_absolutely_convergent()
False
sage: hypergeometric([2, 3], [4], 1).is_absolutely_convergent()
False
sage: hypergeometric([2, 3], [5], 1).is_absolutely_convergent()
False
sage: hypergeometric([2, 3], [6], 1).is_absolutely_convergent()
True
sage: hypergeometric([-2, 3], [4], 5).is_absolutely_convergent()
True
sage: hypergeometric([-2, 3], [4], -5).is_absolutely_convergent()
True
sage: hypergeometric([-2, -3], [4], -5).is_absolutely_convergent()
True
sage: hypergeometric([-2, -3], [-1], 5).is_absolutely_convergent()
False

>>> from sage.all import *
>>> # needs sage.symbolic
>>> hypergeometric([Integer(2), Integer(3)], [Integer(4)], [Integer(6)]).is_absolutely_convergent()
False

(continues on next page)
Functions, Release 10.4

(continued from previous page)

>>> hypergeometric([Integer(2), Integer(3)], [Integer(4)], Integer(1)).is_absolutely_convergent()
False

>>> hypergeometric([Integer(2), Integer(3)], [Integer(5)], Integer(1)).is_absolutely_convergent()
False

>>> hypergeometric([Integer(2), Integer(3)], [Integer(6)], Integer(1)).is_absolutely_convergent()
True

>>> hypergeometric([-Integer(2), Integer(3)], [Integer(4)], ... Integer(5)).is_absolutely_convergent()
True

>>> hypergeometric([Integer(2), -Integer(3)], [Integer(4)], ... Integer(5)).is_absolutely_convergent()
True

>>> hypergeometric([Integer(2), -Integer(3)], [-Integer(4)], ... Integer(5)).is_absolutely_convergent()
True

>>> hypergeometric([Integer(2), -Integer(3)], [-Integer(1)], ... Integer(5)).is_absolutely_convergent()
False

Degree giving zero radius of convergence:

sage: hypergeometric([1, 2, 3], [4], # needs sage.symbolic
....: 2).is_absolutely_convergent()
False

sage: hypergeometric([1, 2, 3], [4], # needs sage.symbolic
....: 1/2).is_absolutely_convergent()
False

sage: (hypergeometric([1, 2, -3], [4], 1/2) # needs sage.symbolic
....: .is_absolutely_convergent()) # polynomial
True

>>> from sage.all import *

>>> hypergeometric([Integer(1), Integer(2), Integer(3)], [Integer(4)], ... # needs sage.symbolic
... Integer(2)).is_absolutely_convergent()
False

>>> hypergeometric([Integer(1), Integer(2), Integer(3)], [Integer(4)], ... # needs sage.symbolic
... Integer(1)/Integer(2)).is_absolutely_convergent()
False

>>> (hypergeometric([Integer(1), Integer(2), -Integer(3)], [Integer(4)], ... Integer(1)/Integer(2)) # needs sage.symbolic
... .is_absolutely_convergent()) # polynomial
True

**is_terminating (a, b, z)**

Determine whether the series represented by self terminates after a finite number of terms.

This happens if any of the numerator parameters are nonnegative integers (with no preceding nonnegative denominator parameters), or \( z = 0 \).
If terminating, the series represents a polynomial of $z$.

EXAMPLES:

```
sage: hypergeometric([1, 2], [3, 4], x).is_terminating()  # needs sage.symbolic
False
sage: hypergeometric([1, -2], [3, 4], x).is_terminating()  # needs sage.symbolic
True
sage: hypergeometric([1, -2], [], x).is_terminating()  # needs sage.symbolic
True
```

```
>>> from sage.all import *

>>> hypergeometric([Integer(1), Integer(2)], [Integer(3), Integer(4)], x).is_terminating()  # needs sage.symbolic
False
>>> hypergeometric([Integer(1), -Integer(2)], [Integer(3), Integer(4)], -x).is_terminating()  # needs sage.symbolic
True
>>> hypergeometric([Integer(1), -Integer(2)], [], x).is_terminating()  # needs sage.symbolic
True
```

\textbf{is\_termwise\_finite}(a, b, z)

Determine whether all terms of \texttt{self} are finite.

Any infinite terms or ambiguous terms beyond the first zero, if one exists, are ignored.

Ambiguous cases (where a term is the product of both zero and an infinity) are not considered finite.

EXAMPLES:

```
sage: # needs sage.symbolic
sage: hypergeometric([2], [3, 4], 5).is_termwise_finite()
True
sage: hypergeometric([2], [-3, 4], 5).is_termwise_finite()
False
sage: hypergeometric([-2], [-3, 4], 5).is_termwise_finite()
True
sage: hypergeometric([-3], [-3, 4], 5).is_termwise_finite()  # ambiguous
False
sage: # needs sage.symbolic
sage: hypergeometric([0], [-1], 5).is_termwise_finite()
True
sage: hypergeometric([0], [0], 5).is_termwise_finite()  # ambiguous
False
sage: hypergeometric([1], [2], Infinity).is_termwise_finite()
False
sage: (hypergeometric([0], [0], Infinity)).is_termwise_finite()  # ambiguous
False
```

```
>>> # needs sage.symbolic

>>> hypergeometric([Integer(2)], [Integer(3), Integer(4)], x).is_termwise_finite()
False
>>> hypergeometric([Integer(1), -Integer(2)], [Integer(3), Integer(4)], -x).is_termwise_finite()  # needs sage.symbolic
True
>>> hypergeometric([Integer(1), -Integer(2)], [], x).is_termwise_finite()  # needs sage.symbolic
True
```
Functions, Release 10.4

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> hypergeometric([Integer(2)], [Integer(3), Integer(4)], Integer(5)).is_termwise_finite()
True
>>> hypergeometric([Integer(2)], [-Integer(3), Integer(4)], Integer(5)).is_termwise_finite()
False
>>> hypergeometric([-Integer(2)], [-Integer(3), Integer(4)], Integer(5)).is_termwise_finite()
True
>>> hypergeometric([-Integer(3)], [-Integer(3), Integer(4)], ...
... Integer(5)).is_termwise_finite()  # ambiguous
False
>>> # needs sage.symbolic
>>> hypergeometric([Integer(0)], [-Integer(1)], Integer(5)).is_termwise_finite()
True
>>> hypergeometric([Integer(0)], [Integer(0)], ...
... Integer(5)).is_termwise_finite()  # ambiguous
False
>>> hypergeometric([Integer(1)], [Integer(2)], Infinity).is_termwise_finite()
False
>>> (hypergeometric([Integer(0)], [Integer(0)], Infinity)
... .is_termwise_finite())  # ambiguous
False
>>> (hypergeometric([Integer(0)], [], Infinity)
... .is_termwise_finite())  # ambiguous
False
```

`sorted_parameters (a, b, z)`

Return with parameters sorted in a canonical order.

**EXAMPLES:**

```python
sage: hypergeometric([2, 1, 3], [5, 4],  # needs sage.symbolic
....: 1/2).sorted_parameters()
hypergeometric((1, 2, 3), (4, 5), 1/2)
```

`terms (a, b, z, n=None)`

Generate the terms of `self` (optionally only `n` terms).

**EXAMPLES:**

```python
sage: list(hypergeometric([-2, 1], [3, 4], x).terms())  # needs sage.symbolic
[1, -1/6*x, 1/120*x^2]
sage: list(hypergeometric([-2, 1], [3, 4], x).terms(2))  # needs sage.symbolic
(continues on next page)
```

1.11. Hypergeometric functions 165
class sage.functions.hypergeometric.Hypergeometric_M

Bases: BuiltinFunction

The confluent hypergeometric function of the first kind, \( y = M(a, b, z) \), is defined to be the solution to Kummer’s differential equation

\[
zy'' + (b - z)y' - ay = 0.
\]

This is not the same as Kummer’s \( U \)-hypergeometric function, though it satisfies the same DE that \( M \) does.

Warning: In the literature, both are called “Kummer confluent hypergeometric” functions.

EXAMPLES:

\[
sage: hypergeometric_M(1, 1, 1.)
\]

\[
\quad \Rightarrow \text{needs mpmath}
\]

2.71828182845905

\[
sage: hypergeometric_M(1, 1, 1)
\]

\[
\quad \Rightarrow \text{needs sage.symbolic}
\]

\[
sage: hypergeometric_M(1, 1, 1)
\]

\[
\quad \Rightarrow \text{needs sage.symbolic}
\]

\[
\quad \Rightarrow \text{n}(70)
\]

2.7182818284590452354

\[
sage: hypergeometric_M(1, 1, 1).simplify_hypergeometric()
\]

\[
\quad \Rightarrow \text{e}
\]

\[
sage: hypergeometric_M(1, 1, 1).simplify_hypergeometric()
\]

\[
\quad \Rightarrow 1/2*sqrt(p1)*\text{erf}(1)*e
\]

\[
sage: hypergeometric_M(1, 1, 1, x).simplify_hypergeometric()
\]

\[
(-1*sqrt(p1)*x*\text{erf}(1*sqrt(-x)))*e^x + sqrt(-x))/sqrt(-x)
\]

\[
\quad \Rightarrow \text{from sage.all import *}
\]

\[
\quad \Rightarrow \text{hypergeometric_M(1, 1, RealNumber('1.'))}
\]

\[
\quad \Rightarrow \text{# needs mpmath}
\]

2.71828182845905

\[
\quad \Rightarrow \text{# needs sage.symbolic}
\]

(continues on next page)
class EvaluationMethods
Bases: object

generalized(a, b, z)

Return as a generalized hypergeometric function.

EXAMPLES:

```python
sage: var('a b z')
# needs sage.symbolic
(a, b, z)
sage: hypergeometric_M(a, b, z).generalized()
# needs sage.symbolic
hypergeometric((a,), (b,), z)
```

```python
>>> from sage.all import *

```
```python
sage: # needs mpmath
sage: hypergeometric_U(1, 1, 1)
hypergeometric_U(1, 1, 1)
sage: hypergeometric_U(1, 1, 1.)
0.596347362323194

sage: # needs sage.symbolic
sage: hypergeometric_U(1, 1, 1).n(70) #...
→ mpmath
0.59634736232319407434
sage: hypergeometric_U(10^4, 1/3, 1).n() #...
→ sage.libs.pari
6.60377008885811e-35745

sage: hypergeometric_U(1, 2, 2).simplify_hypergeometric() 1/2

sage: hypergeometric_U(2 + I, 2, 1).n() #...
→ sage.symbolic
0.183481989942099 - 0.458685959185190*I

sage: hypergeometric_U(1, 3, x).simplify_hypergeometric() #...
→ sage.symbolic
(x + 1)/x^2
```
sage.functions.hypergeometric.closed_form(hyp)

Try to evaluate hyp in closed form using elementary (and other simple) functions.

It may be necessary to call Hypergeometric.deflated() first to find some closed forms.

EXAMPLES:

sage: # needs sage.symbolic
sage: from sage.functions.hypergeometric import closed_form
sage: var(’a b c z’)
(a, b, c, z)

sage: closed_form(hypergeometric([1], [], 1 + z))
-1/z
sage: closed_form(hypergeometric([], [], 1 + z))
e^(z + 1)

sage: closed_form(hypergeometric([1/2], [3/2], -5))
1/10*sqrt(5)*sqrt(pi)*erf(sqrt(5))

(continues on next page)
sage: closed_form(hypergeometric([2], [5], 3))
4
sage: closed_form(hypergeometric([2], [5], 5))
48/625*e^5 + 612/625
sage: closed_form(hypergeometric([1/2, 7/2], [3/2], z))
1/5*z^2/(-z + 1)^5/2 + 2/3*z/(-z + 1)^3/2 + 1/sqrt(-z + 1)
sage: closed_form(hypergeometric([1/2, 1], [2], z))
-2*(sqrt(-z + 1) - 1)/z
sage: closed_form(hypergeometric([1, 1], [2], z))
-log(-z + 1)/z
sage: closed_form(hypergeometric([1, 1], [3], z))
-2*((z - 1)*log(-z + 1)/z - 1)/z
sage: closed_form(hypergeometric([1, 1, 1], [2, 2], x))
hypergeometric((1, 1, 1), (2, 2), x)

>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.functions.hypergeometric import closed_form
>>> var('a b c z')
(a, b, c, z)
>>> closed_form(hypergeometric([Integer(1)], [], Integer(1) + z))
-1/z
>>> closed_form(hypergeometric([[], [], Integer(1) + z]))
e^(z + 1)
>>> closed_form(hypergeometric([[], [Integer(1)/Integer(2)], Integer(4)]))
cosh(4)
>>> closed_form(hypergeometric([[], [Integer(3)/Integer(2)], Integer(4)]))
1/4*sinh(4)
>>> closed_form(hypergeometric([[], [Integer(5)/Integer(2)], Integer(4)]))
3/16*cosh(4) - 3/64*sinh(4)
>>> closed_form(hypergeometric([[], [-Integer(3)/Integer(2)], Integer(4)]))
19/3*cosh(4) - 4*sinh(4)
>>> closed_form(hypergeometric([-Integer(3), Integer(1)], [var('a')], z))
-3*z/a + 6*z^2/((a + 1)*a) - 6*z^3/((a + 2)*(a + 1)*a) + 1
>>> closed_form(hypergeometric([-Integer(3), Integer(1)/Integer(3)], [-Integer(4), z]))
7/162*z^3 + 1/9*z^2 + 1/4*z + 1
>>> closed_form(hypergeometric([[], [], z]))
e^z
>>> closed_form(hypergeometric([a], [], z))
1/((-z + 1)^a)
>>> closed_form(hypergeometric([Integer(1), Integer(1), Integer(2)], [Integer(1), -Integer(1)], z))
(z - 1)^(-2)
>>> closed_form(hypergeometric([Integer(2), Integer(3)], [Integer(1)], x))
-1/(x - 1)^3 + 3*x/(x - 1)^4
>>> closed_form(hypergeometric([Integer(1)/Integer(2)], [Integer(3)/Integer(2)], -Integer(5)))
1/10*sqrt(5)*sqrt(pi)*erf(sqrt(5))
>>> closed_form(hypergeometric([Integer(2)], [Integer(5)], Integer(3)))
4
>>> closed_form(hypergeometric([Integer(2)], [Integer(5)], Integer(5)))
48/625*e^5 + 612/625
>>> closed_form(hypergeometric([Integer(1)/Integer(2), Integer(7)/Integer(2)], [Integer(3)/Integer(2)], z))
1/5*z^2/((-z + 1)^5/2 + 2/3*z/(-z + 1)^3/2 + 1/sqrt(-z + 1)
sage.functions.hypergeometric.rational_param_as_tuple(x)

Utility function for converting rational \( \frac{p}{F_q} \) parameters to tuples (which mpmath handles more efficiently).

EXAMPLES:

```python
sage: from sage.functions.hypergeometric import rational_param_as_tuple
sage: rational_param_as_tuple(1/2)
(1, 2)
sage: rational_param_as_tuple(3)
3
sage: rational_param_as_tuple(pi)  # Needs sage.symbolic
pi
```

1.12 Jacobi elliptic functions

This module implements the 12 Jacobi elliptic functions, along with their inverses and the Jacobi amplitude function.

Jacobi elliptic functions can be thought of as generalizations of both ordinary and hyperbolic trig functions. There are twelve Jacobian elliptic functions. Each of the twelve corresponds to an arrow drawn from one corner of a rectangle to another.

```
+-------+-------+-------+
|      n|       d|       |
|      |       |       |
|      c|       |       |
|      |       |       |
|      s|-------|       |
```

Each of the corners of the rectangle are labeled, by convention, \( s, c, d, \) and \( n \). The rectangle is understood to be lying on the complex plane, so that \( s \) is at the origin, \( c \) is on the real axis, and \( n \) is on the imaginary axis. The twelve Jacobian elliptic functions are then \( pq(x) \), where \( p \) and \( q \) are one of the letters \( s, c, d, n \).

The Jacobian elliptic functions are then the unique doubly-periodic, meromorphic functions satisfying the following three properties:
1. There is a simple zero at the corner \( p \), and a simple pole at the corner \( q \).

2. The step from \( p \) to \( q \) is equal to half the period of the function \( pq(x) \); that is, the function \( pq(x) \) is periodic in the direction \( pq \), with the period being twice the distance from \( p \) to \( q \). \( pq(x) \) is periodic in the other two directions as well, with a period such that the distance from \( p \) to one of the other corners is a quarter period.

3. If the function \( pq(x) \) is expanded in terms of \( x \) at one of the corners, the leading term in the expansion has a coefficient of 1. In other words, the leading term of the expansion of \( pq(x) \) at the corner \( p \) is \( x \); the leading term of the expansion at the corner \( q \) is \( 1/x \), and the leading term of an expansion at the other two corners is 1.

We can write

\[
pq(x) = \frac{pr(x)}{qr(x)}
\]

where \( p, q, \) and \( r \) are any of the letters \( s, c, d, n, \) with the understanding that \( ss = cc = dd = nn = 1 \).

Let

\[
u = \int_0^\phi \frac{d\theta}{\sqrt{1 - m \sin^2 \theta}}.
\]

then the Jacobi elliptic function \( sn(u) \) is given by

\[
sn u = \sin \phi
\]

and \( cn(u) \) is given by

\[
\text{cn} u = \cos \phi
\]

and

\[
\text{dn} u = \sqrt{1 - m \sin^2 \phi}.
\]

To emphasize the dependence on \( m \), one can write \( sn(u|m) \) for example (and similarly for \( cn \) and \( dn \)). This is the notation used below.

For a given \( k \) with \( 0 < k < 1 \) they therefore are solutions to the following nonlinear ordinary differential equations:

- \( sn(x; k) \) solves the differential equations

\[
\frac{d^2 y}{dx^2} + (1 + k^2) y - 2 k^2 y^3 = 0 \quad \text{and} \quad \left( \frac{dy}{dx} \right)^2 = (1 - y^2)(1 - k^2 y^2).
\]

- \( cn(x; k) \) solves the differential equations

\[
\frac{d^2 y}{dx^2} + (1 - 2k^2) y + 2k^2 y^3 = 0 \quad \text{and} \quad \left( \frac{dy}{dx} \right)^2 = (1 - y^2)(1 - k^2 + 2k^2 y^2).
\]

- \( dn(x; k) \) solves the differential equations

\[
\frac{d^2 y}{dx^2} - (2 - k^2) y + 2y^3 = 0 \quad \text{and} \quad \left( \frac{dy}{dx} \right)^2 = y^2(1 - k^2 - y^2).
\]

If \( K(m) \) denotes the complete elliptic integral of the first kind (named \texttt{elliptic_kc} in Sage), the elliptic functions \( sn(x|m) \) and \( cn(x|m) \) have real periods \( 4K(m) \), whereas \( dn(x|m) \) has a period \( 2K(m) \). The limit \( m \to 0 \) gives \( K(0) = \pi/2 \) and trigonometric functions: \( sn(x|0) = \sin x, \ cn(x|0) = \cos x, \ dn(x|0) = 1 \). The limit \( m \to 1 \) gives \( K(1) \to \infty \) and hyperbolic functions: \( sn(x|1) = \tanh x, \ cn(x|1) = \sech x, \ dn(x|1) = \sech x \).
REFERENCES:

- Wikipedia article Jacobi%27s_elliptic_functions
- [KS2002]

AUTHORS:

- David Joyner (2006): initial version
- Eviatar Bach (2013): complete rewrite, new numerical evaluation, and addition of the Jacobi amplitude function

```python
class sage.functions.jacobi.InverseJacobi(kind)
    Bases: BuiltinFunction
    Base class for the inverse Jacobi elliptic functions.

class sage.functions.jacobi.Jacobi(kind)
    Bases: BuiltinFunction
    Base class for the Jacobi elliptic functions.

class sage.functions.jacobi.JacobiAmplitude
    Bases: BuiltinFunction
    The Jacobi amplitude function
    \( \text{am}(x|m) = \int_0^x dn(t|m) dt \) for \(-K(m) \leq x \leq K(m)\), \(F(\text{am}(x|m)|m) = x\).

sage.functions.jacobi.inverse_jacobi(kind, x, m, **kwargs)
    The inverses of the 12 Jacobi elliptic functions. They have the property that
    \( pq(\text{arcpq}(x|m)|m) = pq(pq^{-1}(x|m)|m) = x \).

INPUT:

- kind - a string of the form 'pq', where p, q are in c, d, n, s
- x - a real number
- m - a real number; note that \( m = k^2 \), where k is the elliptic modulus

EXAMPLES:

```sage
sage: jacobi('dn', inverse_jacobi('dn', 3, 0.4), 0.4)  #...

→ needs mpmath
3.00000000000000

sage: inverse_jacobi('dn', 10, 1/10).n(digits=50)  #...

→ needs mpmath
2.477736267904273296523691232988240759001423661683*I

sage: inverse_jacobi_dn(x, 1)  # needs sage.symbolic
arcsech(x)

sage: inverse_jacobi_dn(1, 3)  # needs mpmath
0

sage: # needs sage.symbolic
sage: m = var('m')
sage: z = inverse_jacobi_dn(x, m).series(x, 4).subs(x=0.1, m=0.7)
sage: jacobi_dn(z, 0.7)
0.0999892750039819...

sage: inverse_jacobi_nd(x, 1)  # arccosh(x)
```

(continues on next page)
Functions, Release 10.4

sage: # needs mpmath
sage: inverse_jacobi_nd(1, 2)
0
sage: inverse_jacobi_ns(10^-5, 3).n()
5.77350269202456e-6 + 1.17142008414677*I
sage: jacobi('sn', 1/2, 1/2)
jacobi_sn(1/2, 1/2)
sage: jacobi('sn', 1/2, 1/2).n()
0.470750473655657
sage: inverse_jacobi('sn', 0.47, 1/2)
0.49098231322220
sage: inverse_jacobi('sn', 0.4707504, 0.5)
0.49999911466555
sage: P = plot(inverse_jacobi('sn', x, 0.5), 0, 1) # needs sage.plot

>>> from sage.all import *
>>> jacobi('dn', inverse_jacobi('dn', Integer(3), RealNumber('0.4')), RealNumber(→0.4)) # needs mpmath
3.00000000000000
>>> inverse_jacobi('dn', Integer(10), Integer(1)/Integer(10)).
→(digits=Integer(50)) # needs mpmath
2.477736267904273296523691232988240759001423661683*I
>>> inverse_jacobi_dn(x, Integer(1)) # needs sage.symbolic
arcsech(x)
>>> inverse_jacobi_dn(Integer(1), Integer(3)) # needs mpmath
0

>>> # needs sage.symbolic
>>> m = var('m')
>>> z = inverse_jacobi_dn(x, m).series(x, Integer(4)).subs(x=RealNumber('0.1'),←m=RealNumber('0.7'))
0.0999892750039819...
>>> inverse_jacobi_nd(x, Integer(1))
arccosh(x)

>>> # needs mpmath
>>> inverse_jacobi_nd(Integer(1), Integer(2))
0

>>> inverse_jacobi_ns(Integer(10)^-Integer(5), Integer(3)).n()
5.77350269202456e-6 + 1.17142008414677*I
>>> jacobi('sn', Integer(1)/Integer(2), Integer(1)/Integer(2))
jacobi_sn(1/2, 1/2)
>>> jacobi('sn', Integer(1)/Integer(2), Integer(1)/Integer(2)).n()
0.470750473655657
>>> inverse_jacobi('sn', RealNumber('0.47'), Integer(1)/Integer(2))
0.49098231322220
>>> inverse_jacobi('sn', RealNumber('0.4707504'), RealNumber('0.5'))
0.49999911466555
>>> P = plot(inverse_jacobi('sn', x, RealNumber('0.5')), Integer(0), Integer(1)) # needs sage.plot

sage.functions.jacobi.inverse_jacobi_f(kind, x, m)
Internal function for numerical evaluation of a continuous complex branch of each inverse Jacobi function, as described in [Tee1997]. Only accepts real arguments.

`sage.functions.jacobi.jacobi(kind, z, m, **kwargs)`

The 12 Jacobi elliptic functions.

**INPUT:**

- `kind` – a string of the form 'pq', where p, q are in c, d, n, s
- `z` – a complex number
- `m` – a complex number; note that \( m = k^2 \), where \( k \) is the elliptic modulus

**EXAMPLES:**

```python
sage: # needs mpmath
sage: jacobi('sn', 1, 1)
tanh(1)
sage: jacobi('cd', 1, 1/2)
jacobi_cd(1, 1/2)
sage: RDF(jacobi('cd', 1, 1/2))
0.7240097216593705
sage: (RDF(jacobi('cn', 1, 1/2)), RDF(jacobi('dn', 1, 1/2)),
    ...: RDF(jacobi('cn', 1, 1/2) / jacobi('dn', 1, 1/2)))
(0.5959765676721407, 0.8231610016315962, 0.7240097216593705)
sage: jsn = jacobi('sn', x, 1)  # needs sage.symbolic
sage: P = plot(jsn, 0, 1)  # needs sage.plot sage.symbolic
```
1.13 Airy functions

This module implements Airy functions and their generalized derivatives. It supports symbolic functionality through Maxima and numeric evaluation through mpmath and scipy.

Airy functions are solutions to the differential equation $f''(x) - xf(x) = 0$.

Four global function symbols are immediately available, please see

- `airy_ai()`: for the Airy Ai function
- `airy_ai_prime()`: for the first differential of the Airy Ai function
- `airy_bi()`: for the Airy Bi function
- `airy_bi_prime()`: for the first differential of the Airy Bi function

AUTHORS:

- Oscar Gerardo Lazo Arjona (2010): initial version
- Douglas McNeil (2012): rewrite

EXAMPLES:

Verify that the Airy functions are solutions to the differential equation:

```python
sage: diff(airy_ai(x), x, 2) - x * airy_ai(x)  # needs sage.symbolic
0
dsage: diff(airy_bi(x), x, 2) - x * airy_bi(x)  # needs sage.symbolic
0
```

```python
>>> from sage.all import *
>>> diff(airy_ai(x), x, Integer(2)) - x * airy_ai(x)  # needs sage.symbolic
0
>>> diff(airy_bi(x), x, Integer(2)) - x * airy_bi(x)  # needs sage.symbolic
0
```

class `sage.functions.airy.FunctionAiryAiGeneral`

Bases: `BuiltinFunction`

The generalized derivative of the Airy Ai function

INPUT:

- `alpha` – Return the $\alpha$-th order fractional derivative with respect to $z$. For $\alpha = n = 1, 2, 3, \ldots$ this gives the derivative $\text{Ai}^{(n)}(z)$, and for $\alpha = -n = -1, -2, -3, \ldots$ this gives the $n$-fold iterated integral.

\[
f_0(z) = \text{Ai}(z) \\
f_n(z) = \int_0^z f_{n-1}(t)dt
\]

- `x` – The argument of the function

EXAMPLES:
```python
sage: # needs sage.symbolic
sage: from sage.functions.airy import airy_ai_general
sage: x, n = var('x n')
sage: airy_ai_general(-2, x)
airy_ai(-2, x)
sage: derivative(airy_ai_general(-2, x), x)
airy_ai(-1, x)
sage: airy_ai_general(n, x)
airy_ai(n, x)
sage: derivative(airy_ai_general(n, x), x)
airy_ai(n + 1, x)
```

```python
globals()['*']
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.functions.airy import airy_ai_general
>>> x, n = var('x n')
>>> airy_ai_general(-Integer(2), x)
airy_ai(-2, x)
>>> derivative(airy_ai_general(-Integer(2), x), x)
airy_ai(-1, x)
>>> airy_ai_general(n, x)
airy_ai(n, x)
>>> derivative(airy_ai_general(n, x), x)
airy_ai(n + 1, x)
```

class sage.functions.airy.FunctionAiryAiPrime

Bases: BuiltinFunction

The derivative of the Airy Ai function; see `airy_ai()` for the full documentation.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: x, n = var('x n')
sage: airy_ai_prime(x)
airy_ai_prime(x)
>>> airy_ai_prime(Integer(0))
-1/3*3^(2/3)/gamma(1/3)
>>> airy_ai_prime(x)._sympy_()  # needs sympy
airyai_prime(x)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.functions.airy import airy_ai_general
>>> x, n = var('x n')
>>> airy_ai_prime(x)
airy_ai_prime(x)
>>> airy_ai_prime(Integer(0))
-1/3*3^(2/3)/gamma(1/3)
>>> airy_ai_prime(x)._sympy_()  # needs sympy
airyai_prime(x)
```

class sage.functions.airy.FunctionAiryAiSimple

Bases: BuiltinFunction

The class for the Airy Ai function.

1.13. Airy functions
EXAMPLES:

```python
sage: from sage.functions.airy import airy_ai_simple
sage: f = airy_ai_simple(x); f
# needs sage.symbolic
airy_ai(x)
sage: airy_ai_simple(x)._sympy_()
# needs sage.symbolic
airyai(x)
```

```python
>>> from sage.all import *
>>> from sage.functions.airy import airy_ai_simple
>>> f = airy_ai_simple(x); f
# needs sage.symbolic
airy_ai(x)
>>> airy_ai_simple(x)._sympy_()
# needs sage.symbolic
airyai(x)
```

```python
class sage.functions.airy.FunctionAiryBiGeneral
Bases: BuiltinFunction

The generalized derivative of the Airy Bi function.

INPUT:

- alpha – Return the α-th order fractional derivative with respect to z. For α = n = 1, 2, 3,... this gives the derivative Bi^(n)(z), and for α = −n = −1, −2, −3,... this gives the n-fold iterated integral.

\[ f_0(z) = Bi(z) \]
\[ f_n(z) = \int_0^z f_{n-1}(t)dt \]

- x – The argument of the function

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: from sage.functions.airy import airy_bi_general
sage: x, n = var('x n')
sage: airy_bi_general(-2, x)
airy_bi(-2, x)
sage: derivative(airy_bi_general(-2, x), x)
airy_bi(-1, x)
sage: airy_bi_general(n, x)
airy_bi(n, x)
sage: derivative(airy_bi_general(n, x), x)
airy_bi(n + 1, x)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.functions.airy import airy_bi_general
>>> x, n = var('x n')
>>> airy_bi_general(-Integer(2), x)
airy_bi(-2, x)
>>> derivative(airy_bi_general(-Integer(2), x), x)
airy_bi(-1, x)
```

(continues on next page)
class sage.functions.airy.FunctionAiryBiPrime

Bases: BuiltinFunction

The derivative of the Airy Bi function; see airy_bi() for the full documentation.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: x, n = var('x n')
sage: airy_bi_prime(x)
airy_bi_prime(x)
sage: airy_bi_prime(0)
3^(1/6)/gamma(1/3)
sage: airy_bi_prime(x)._sympy_()
# needs sympy
airybiprime(x)
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> x, n = var('x n')
>>> airy_bi_prime(x)
airy_bi_prime(x)
```

```python
>>> from sage.functions.airy import airy_bi_simple
sage: f = airy_bi_simple(x); f
# needs sage.symbolic
airy_bi(x)
sage: f._sympy_()
# needs sympy sage.symbolic
airybi(x)
```

```python
>>> from sage.all import *
>>> from sage.functions.airy import airy_bi_simple
>>> f = airy_bi_simple(x); f
# needs...
airy_bi(x)
>>> f._sympy_()
# needs...
airybi(x)
```
functions airy airy_ai (alpha, x=None, hold_derivative=True, **kwds)

The Airy Ai function

The Airy Ai function Ai(x) is (along with Bi(x)) one of the two linearly independent standard solutions to the Airy differential equation f''(x) − xf(x) = 0. It is defined by the initial conditions:

\[ Ai(0) = \frac{1}{2^{2/3} \Gamma \left(\frac{2}{3}\right)}, \]
\[ Ai'(0) = -\frac{1}{2^{1/3} \Gamma \left(\frac{1}{3}\right)}. \]

Another way to define the Airy Ai function is:

\[ Ai(x) = \frac{1}{\pi} \int_0^\infty \cos \left(\frac{1}{3} t^3 + xt \right) dt. \]

INPUT:

- alpha – Return the \( \alpha \)-th order fractional derivative with respect to \( z \). For \( \alpha = n = 1, 2, 3, \ldots \) this gives the derivative \( Ai^{(n)}(z) \), and for \( \alpha = -n = -1, -2, -3, \ldots \) this gives the \( n \)-fold iterated integral.

\[ f_0(z) = Ai(z) \]
\[ f_n(z) = \int_0^z f_{n-1}(t) dt \]

- x – The argument of the function

- hold_derivative – Whether or not to stop from returning higher derivatives in terms of \( Ai(x) \) and \( Ai'(x) \)

See also:

airy_bi()

EXAMPLES:

sage: n, x = var('n x')

\[ \texttt{airy_ai(x)} \]

>>> from sage.all import *

\[ \texttt{airy_ai(x)} \]

It can return derivatives or integrals:
It can be evaluated symbolically or numerically for real or complex values:

```
sage: airy_ai(0) # needs sage.symbolic
1/3*3^(1/3)/gamma(2/3)
sage: airy_ai(0.0) # needs mpmath
0.355028053887817
sage: airy_ai(I) # needs sage.symbolic
airy_ai(I)
sage: airy_ai(1.0*I) # needs sage.symbolic
0.331493305432141 - 0.317449858968444*I
```

The functions can be evaluated numerically either using mpmath, which can compute the values to arbitrary precision, and scipy:

```
sage: airy_ai(2).n(prec=100) # needs sage.symbolic
0.034924130423274379135322080792
sage: airy_ai(2).n(algorithm='mpmath', prec=100) # needs sage.symbolic
0.034924130423274379135322080792
```

1.13. Airy functions
And the derivatives can be evaluated:

```
sage: airy_ai(1, 0)  # needs sage.symbolic
-1/3*3^(2/3)/gamma(1/3)
sage: airy_ai(1, 0.0) # needs mpmath
-0.258819403792807
```

Plots:

```
sage: plot(airy_ai(x), (x, -10, 5)) + plot(airy_ai_prime(x), # needs sage.plot sage.symbolic
....: (x, -10, 5), color='red')
Graphics object consisting of 2 graphics primitives
```

REFERENCES:

- Abramowitz, Milton; Stegun, Irene A., eds. (1965), “Chapter 10”
- Wikipedia article Airy_function

The Airy Bi function

The Airy Bi function Bi(x) is (along with Ai(x)) one of the two linearly independent standard solutions to the Airy
differential equation $f''(x) - xf(x) = 0$. It is defined by the initial conditions:

$$Bi(0) = \frac{1}{3^{1/6}\Gamma(\frac{2}{3})},$$
$$Bi'(0) = \frac{3^{1/6}}{\Gamma(\frac{2}{3})}.$$

Another way to define the Airy Bi function is:

$$Bi(x) = \frac{1}{\pi} \int_0^\infty \left[ \exp\left(x t - \frac{t^3}{3}\right) + \sin\left(x t + \frac{1}{3} t^3\right) \right] dt.$$

**INPUT:**

- **alpha** – Return the $\alpha$-th order fractional derivative with respect to $z$. For $\alpha = n = 1, 2, 3, \ldots$ this gives the derivative $Bi^{(n)}(z)$, and for $\alpha = -n = -1, -2, -3, \ldots$ this gives the $n$-fold iterated integral.

$$f_0(z) = Bi(z)$$
$$f_n(z) = \int_0^z f_{n-1}(t) dt$$

- **x** – The argument of the function

- **hold_derivative** – Whether or not to stop from returning higher derivatives in terms of $Bi(x)$ and $Bi'(x)$

**See also:**

`airy_ai()`

**EXAMPLES:**

```sage
n, x = var('n x')
airy_bi(x)
```

It can return derivatives or integrals:

```sage
# needs sage.symbolic
airy_bi(2, x)
airy_bi(2, x)
airy_bi_prime(x)
airy_bi(2, x, hold_derivative=False)
x*airy_bi(x)
airy_bi(-2, x, hold_derivative=False)
airy_bi(-2, x)
airy_bi(n, x)
airy_bi(n, x)
```

1.13. Airy functions
It can be evaluated symbolically or numerically for real or complex values:

```
sage: airy_bi(0)
    # needs sage.symbolic
    1/3*3^(5/6)/gamma(2/3)

sage: airy_bi(0.0)
    # needs mpmath
    0.614926627446001

sage: airy_bi(I)
    # needs sage.symbolic
    airy_bi(I)

sage: airy_bi(1.0*I)
    # needs sage.symbolic
    0.648858208330395 + 0.344958634768048*I
```

The functions can be evaluated numerically using mpmath, which can compute the values to arbitrary precision, and scipy:

```
sage: airy_bi(2).n(prec=100)
    # needs sage.symbolic
    3.2980949999782147102806044252

sage: airy_bi(2).n(algorithm='mpmath', prec=100)
    # needs sage.symbolic
    3.2980949999782147102806044252

sage: airy_bi(2).n(algorithm='scipy')
    # rel tol 1e-10
    # needs scipy sage.symbolic
    3.2980949999782134
```

(continues on next page)
And the derivatives can be evaluated:

```python
sage: airy_bi(1, 0)  # needs sage.symbolic
3^(1/6)/gamma(1/3)
sage: airy_bi(1, 0.0)  # needs mpmath
0.448288357353826
```

Plots:

```python
sage: plot(airy_bi(x), (x, -10, 5)) + plot(airy_bi_prime(x),  # needs sage.plot sage.symbolic
       (x, -10, 5), color='red')
Graphics object consisting of 2 graphics primitives
```

REFERENCES:

- Abramowitz, Milton; Stegun, Irene A., eds. (1965), “Chapter 10”
- Wikipedia article Airy function

### 1.14 Bessel functions

This module provides symbolic Bessel and Hankel functions, and their spherical versions. These functions use the mpmath library for numerical evaluation and Maxima, GiNaC, Pynac for symbolics.

The main objects which are exported from this module are:

- `bessel_J(n, x)` – The Bessel J function
- `bessel_Y(n, x)` – The Bessel Y function
• `bessel_I(n, x)` – The Bessel I function
• `bessel_K(n, x)` – The Bessel K function
• `Bessel(...)` – A factory function for producing Bessel functions of various kinds and orders
• `hankel1(nu, z)` – The Hankel function of the first kind
• `hankel2(nu, z)` – The Hankel function of the second kind
• `struve_H(nu, z)` – The Struve function
• `struve_L(nu, z)` – The modified Struve function
• `spherical_bessel_J(n, z)` – The Spherical Bessel J function
• `spherical_bessel_Y(n, z)` – The Spherical Bessel Y function
• `spherical_hankel1(n, z)` – The Spherical Hankel function of the first kind
• `spherical_hankel2(n, z)` – The Spherical Hankel function of the second kind

Bessel functions, first defined by the Swiss mathematician Daniel Bernoulli and named after Friedrich Bessel, are canonical solutions $y(x)$ of Bessel's differential equation:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - \nu^2) y = 0,$$

for an arbitrary complex number $\nu$ (the order).

• In this module, $J_\nu$ denotes the unique solution of Bessel’s equation which is non-singular at $x = 0$. This function is known as the Bessel Function of the First Kind. This function also arises as a special case of the hypergeometric function $\,_{0}F_{1}$:

$$J_\nu(x) = \frac{x^n}{2^n \Gamma(\nu + 1)} \,_{0}F_{1}(\nu + 1, -\frac{x^2}{4}).$$

• The second linearly independent solution to Bessel’s equation (which is singular at $x = 0$) is denoted by $Y_\nu$ and is called the Bessel Function of the Second Kind:

$$Y_\nu(x) = \frac{J_\nu(x) \cos(\pi \nu) - J_{-\nu}(x)}{\sin(\pi \nu)}.$$

• There are also two commonly used combinations of the Bessel J and Y Functions. The Bessel I Function, or the Modified Bessel Function of the First Kind, is defined by:

$$I_\nu(x) = i^{-\nu} J_\nu(ix).$$

The Bessel K Function, or the Modified Bessel Function of the Second Kind, is defined by:

$$K_\nu(x) = \frac{\pi}{2} \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\pi \nu)}.$$  

We should note here that the above formulas for Bessel Y and K functions should be understood as limits when $\nu$ is an integer.

• It follows from Bessel’s differential equation that the derivative of $J_\nu(x)$ with respect to $x$ is:

$$\frac{d}{dx} J_\nu(x) = \frac{1}{x^n} \left( x^n J_{n-1}(x) - n x^{n-1} J_n(x) \right)$$
• Another important formulation of the two linearly independent solutions to Bessel’s equation are the Hankel functions $H^{(1)}_{\nu}(x)$ and $H^{(2)}_{\nu}(x)$, defined by:

$$H^{(1)}_{\nu}(x) = J_{\nu}(x) + iY_{\nu}(x)$$
$$H^{(2)}_{\nu}(x) = J_{\nu}(x) - iY_{\nu}(x)$$

where $i$ is the imaginary unit (and $J_{\nu}$ and $Y_{\nu}$ are the usual $J$- and $Y$-Bessel functions). These linear combinations are also known as Bessel functions of the third kind; they are also two linearly independent solutions of Bessel’s differential equation. They are named for Hermann Hankel.

• When solving for separable solutions of Laplace’s equation in spherical coordinates, the radial equation has the form:

$$x^2 \frac{d^2 y}{dx^2} + 2x \frac{dy}{dx} + [x^2 - n(n + 1)]y = 0.$$  

The spherical Bessel functions $j_n$ and $y_n$, are two linearly independent solutions to this equation. They are related to the ordinary Bessel functions $J_n$ and $Y_n$ by:

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x),$$
$$y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+1/2}(x) = (-1)^n \sqrt{\frac{\pi}{2x}} J_{-n-1/2}(x).$$

**EXAMPLES:**

Evaluate the Bessel J function symbolically and numerically:

```sage
# needs sage.symbolic
sage: bessel_J(0, x)
bessel_J(0, x)
sage: bessel_J(0, 0)
1
sage: bessel_J(0, x).diff(x)
-1/2*bessel_J(1, x) + 1/2*bessel_J(-1, x)
sage: N(bessel_J(0, 0), digits=20)
1.0000000000000000000
sage: find_root(bessel_J(0,x), 0, 5)  # needs scipy
2.404825557695773
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> bessel_J(Integer(0), x)
bessel_J(0, x)
>>> bessel_J(Integer(0), Integer(0))
1
>>> bessel_J(Integer(0), x).diff(x)
-1/2*bessel_J(1, x) + 1/2*bessel_J(-1, x)
>>> N(bessel_J(Integer(0), Integer(0)), digits=Integer(20))
1.0000000000000000000
>>> find_root(bessel_J(Integer(0),x), Integer(0), Integer(5))  # needs scipy
2.404825557695773
```

Plot the Bessel J function:
Functions, Release 10.4

```
sage: f(x) = Bessel(0)(x); f
           # needs sage.symbolic
x |--> bessel_J(0, x)
sage: plot(f, (x, 1, 10))
           # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

```
>>> from sage.all import *
>>>
__tmp__=var("x"); f = symbolic_expression(Bessel(Integer(0))(x)).function(x);
# needs sage.symbolic
x |--> bessel_J(0, x)
>>> plot(f, (x, Integer(1), Integer(10)))
# needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

Visualize the Bessel Y function on the complex plane (set plot_points to a higher value to get more detail):

```
sage: complex_plot(bessel_Y(0, x), (-5, 5), (-5, 5), plot_points=20)
# needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

```
>>> from sage.all import *
>>>
complex_plot(bessel_Y(Integer(0), x), (-Integer(5), Integer(5)), (-Integer(5), Integer(5)), plot_points=Integer(20))
# needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

Evaluate a combination of Bessel functions:

```
sage: # needs sage.symbolic
sage: f(x) = bessel_J(1, x) - bessel_Y(0, x)
sage: f(pi)
bessel_J(1, pi) - bessel_Y(0, pi)
sage: f(pi).n()
-0.0437509653365599
sage: f(pi).n(digits=Integer(50))
-0.043750965336559909054985168023342675387737118378169
```

```
>>> from sage.all import *
>>>
# needs sage.symbolic
>>>
__tmp__=var("x"); f = symbolic_expression(bessel_J(Integer(1), x) - bessel_Y(Integer(0), x)).function(x)
>>>
__tmp__=var("x"); f = symbolic_expression(bessel_J(Integer(1), x) - bessel_Y(Integer(0), x)).function(x)
```

Symbolically solve a second order differential equation with initial conditions \(y(1) = a\) and \(y'(1) = b\) in terms of Bessel functions:

```
sage: # needs sage.symbolic
sage: y = function('y')(x)
sage: a, b = var('a, b')
```
Functions, Release 10.4

(continued from previous page)

```
sage: diffeq = x^2*diff(y,x,x) + x*diff(y,x) + x^2*y == 0
g sage: f = desolve(diffeq, y, [1, a, b]); f
(a*bessel_Y(1, 1) + b*bessel_Y(0, 1))*bessel_J(0, x)/(bessel_J(0, 1)*bessel_Y(1, 1) - bessel_J(1, 1)*bessel_Y(0, 1)) -
(a*bessel_J(1, 1) + b*bessel_J(0, 1))*bessel_Y(0, x)/(bessel_J(0, 1)*bessel_Y(1, 1) - bessel_J(1, 1)*bessel_Y(0, 1))
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> y = function('y')(x)
>>> a, b = var('a, b')
>>> diffeq = x**Integer(2)*diff(y,x,x) + x*diff(y,x) + x**Integer(2)*y ==
˓→Integer(0)
>>> f = desolve(diffeq, y, [Integer(1), a, b]); f
(a*bessel_Y(1, 1) + b*bessel_Y(0, 1))*bessel_J(0, x)/(bessel_J(0, 1)*bessel_Y(1, 1) - bessel_J(1, 1)*bessel_Y(0, 1)) -
(a*bessel_J(1, 1) + b*bessel_J(0, 1))*bessel_Y(0, x)/(bessel_J(0, 1)*bessel_Y(1, 1) - bessel_J(1, 1)*bessel_Y(0, 1))
```

For more examples, see the docstring for `Bessel()`.

AUTHORS:
- Some of the documentation here has been adapted from David Joyner's original documentation of Sage's special functions module (2006).

REFERENCES:
- [AS-Bessel]
- [AS-Spherical]
- [AS-Struve]
- [DLMF-Bessel]
- [DLMF-Struve]
- [WP-Bessel]
- [WP-Struve]

```
sage.functions.bessel.Bessel(*args, **kwds)
```

A function factory that produces symbolic I, J, K, and Y Bessel functions. There are several ways to call this function:

- `Bessel(order, type)`
- `Bessel(order) – type defaults to 'J'`
- `Bessel(order, typ=T)`
- `Bessel(typ=T) – order is unspecified, this is a 2-parameter function`
- `Bessel() – order is unspecified, type is 'J'`

where `order` can be any integer and `T` must be one of the strings 'I', 'J', 'K', or 'Y'.

See the EXAMPLES below.

EXAMPLES:

Construction of Bessel functions with various orders and types:
sage: Bessel()
bessel_J
sage: Bessel(typ='K')
bessel_K

sage: # needs sage.symbolic
sage: Bessel(1)(x)
bessel_J(1, x)
sage: Bessel(1, 'Y')(x)
bessel_Y(1, x)
sage: Bessel(-2, 'Y')(x)
bessel_Y(-2, x)
sage: Bessel(0, typ='I')(x)
bessel_I(0, x)

>>> from sage.all import *
... Bessel()
bessel_J
... Bessel(typ='K')
bessel_K

... # needs sage.symbolic
... Bessel(Integer(1))(x)
bessel_J(1, x)
... Bessel(Integer(1), 'Y')(x)
bessel_Y(1, x)
... Bessel(- Integer(2), 'Y')(x)
bessel_Y(-2, x)
... Bessel(Integer(0), typ='I')(x)
bessel_I(0, x)

Evaluation:

sage: f = Bessel(1)
sage: f(3.0)  # needs mpmath
0.339058958525936

sage: # needs sage.symbolic
sage: f(3)
bessel_J(1, 3)
sage: f(3).n(digits=50)
0.33905895852593645892551459720647889697308041819801

sage: g = Bessel(typ='J')
sage: g(1,3)
bessel_J(1, 3)
sage: g(2, 3+I).n()
0.634160370148554 + 0.0253384000032695*I

sage: numerical_integral(1/pi*cos(3*sin(x)), 0.0, pi)[0]  # needs mpmath
0.339058958525936

...:  - Bessel(0, 'J')(3.0) < 1e-15
True

>>> from sage.all import *
... f = Bessel(Integer(1))
... f(RealNumber('3.0'))  # needs mpmath
0.339058958525936

(continues on next page)
Symbolic calculus:

\[
\text{sage}: f(x) = Bessel(0, 'J')(x) \quad \# \text{needs sage.symbolic} \\
\text{sage}: \text{derivative}(f, x) \quad \# \text{needs sage.symbolic} \\
\text{x |--> -1/2*bessel_J(1, x) + 1/2*bessel_J(-1, x)} \\
\text{sage}: \text{derivative}(f, x, x) \quad \# \text{needs sage.symbolic} \\
\text{x |--> 1/4*bessel_J(2, x) - 1/2*bessel_J(0, x) + 1/4*bessel_J(-2, x)}
\]

Verify that \( J_0 \) satisfies Bessel's differential equation numerically using the \text{test_relation()} method:

\[
\text{sage}: y = bessel_J(0, x) \quad \# \text{needs sage.symbolic} \\
\text{sage}: \text{diffeq} = x^2*\text{derivative}(y, x, x) + x*\text{derivative}(y, x) + x^2*y == 0 \quad \# \text{needs sage.symbolic} \\
\text{sage}: \text{diffeq.test_relation}(\text{proof}=\text{False}) \quad \# \text{needs sage.symbolic} \\
\text{True}
\]

Conversion to other systems:

\[
\text{from sage.all import *} \quad \# \text{needs sage.symbolic} \\
\text{y = bessel_J(Integer(0), x)} \quad \# \text{needs sage.symbolic} \\
\text{diffeq = x**Integer(2)*\text{derivative}(y, x, x) + x*\text{derivative}(y, x) + x**Integer(2)*y == Integer(0)} \quad \# \text{needs sage.symbolic} \\
\text{diffeq.test_relation(\text{proof}=\text{False})} \quad \# \text{needs sage.symbolic} \\
\text{True}
\]
Compute the particular solution to Bessel’s Differential Equation that satisfies $y(1) = 1$ and $y'(1) = 1$, then verify the initial conditions and plot it:

$$\text{sage: } \# \text{ needs sage.symbolic}$$
$$\text{sage: } y = \text{function('y')(x)}$$
$$\text{sage: } \text{diff} = x^2\text{diff}(y,x,x) + x\text{diff}(y,x) + x^2y = 0$$
$$\text{sage: } f = \text{desolve}\left(\text{diff}, \text{y}, [1, 1, 1]\right)$$
$$\text{sage: } f = (\text{bessel}_Y(1, 1) + \text{bessel}_Y(0, 1))\text{bessel}_J(0, x)/(\text{bessel}_J(0, 1)\text{bessel}_Y(1, 1) - \text{bessel}_J(1, 1)\text{bessel}_Y(0, 1)) - (\text{bessel}_J(1, 1) + \text{bessel}_J(0, 1))\text{bessel}_Y(0, x)/(\text{bessel}_J(0, 1)\text{bessel}_Y(1, 1) - \text{bessel}_J(1, 1)\text{bessel}_Y(0, 1))$$

$$\text{sage: } f.subs(x=1).n() \quad \# \text{ numerical verification}$$
$$1.00000000000000$$

$$\text{sage: } f.p = f.diff(x)$$
$$\text{sage: } f.p.subs(x=1).n()$$
$$1.00000000000000$$

$$\text{sage: } f.p.subs(x=1).simplify_full() \quad \# \text{ symbolic verification}$$
$$1$$

$$\text{sage: } \text{plot}(f, (x, 0, 5))$$
$$\text{Graphics object consisting of 1 graphics primitive}$$

(continues on next page)
>> f.subs(x=Integer(1)).n()  # numerical verification
1.00000000000000
>>> fp = f.diff(x)
>>> fp.subs(x=Integer(1)).n()
1.00000000000000
>>> f.subs(x=Integer(1)).simplify_full()  # symbolic verification
1
>>> fp = f.diff(x)
# needs sage.symbolic
>>> fp.subs(x=Integer(1)).simplify_full()
1

>>> plot(f, (x,Integer(0),Integer(5)))  # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive

Plotting:

```python
sage: f(x) = Bessel(0)(x); f  # needs sage.symbolic
x |--> bessel_J(0, x)
sage: plot(f, (x, 1, 10))  # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```
... for i in range(Integer(5)))
>>> show(G)  # needs sage.plot

A recreation of Abramowitz and Stegun Figure 9.1:

```
sage: # needs sage.plot sage.symbolic
sage: G = plot(Bessel(0, 'J'), 0, 15, color='black')
sage: G += plot(Bessel(0, 'Y'), 0, 15, color='black')
sage: G += plot(Bessel(1, 'J'), 0, 15, color='black', linestyle='dotted')
sage: G += plot(Bessel(1, 'Y'), 0, 15, color='black', linestyle='dotted')
sage: show(G, ymin=-1, ymax=1)
```

```python
>>> from sage.all import *
>>> # needs sage.plot sage.symbolic
>>> G = plot(Bessel(Integer(0), 'J'), Integer(0), Integer(15), color='black')
>>> G += plot(Bessel(Integer(0), 'Y'), Integer(0), Integer(15), color='black')
>>> G += plot(Bessel(Integer(1), 'J'), Integer(0), Integer(15), color='black',
            linestyle='dotted')
>>> G += plot(Bessel(Integer(1), 'Y'), Integer(0), Integer(15), color='black',
            linestyle='dotted')
>>> show(G, ymin=-Integer(1), ymax=Integer(1))
```

class sage.functions.bessel.Function_Bessel_I

Bases: BuiltinFunction

The Bessel I function, or the Modified Bessel Function of the First Kind.

**DEFINITION:**

\[ I_\nu(x) = i^{-\nu} J_\nu(i x) \]

**EXAMPLES:**

```
sage: bessel_I(1.0, 1.0)  # needs mpmath
0.565159103992485

sage: # needs sage.symbolic
sage: bessel_I(1, x)
bessel_I(1, x)
sage: n = var('n')
sage: bessel_I(n, x)
bessel_I(n, x)
sage: bessel_I(2, I).n()
-0.114903484931900
```

```python
>>> from sage.all import *
```
Examples of symbolic manipulation:

```
sage: # needs sage.symbolic
sage: a = bessel_I(pi, bessel_I(1, I))
sage: N(a, digits=20)
0.00026073272117205890525 - 0.001152895488908057268*I
sage: f = bessel_I(2, x)
sage: f.diff(x)
1/2*bessel_I(3, x) + 1/2*bessel_I(1, x)
```

Special identities that `bessel_I` satisfies:

```
sage: # needs sage.symbolic
sage: bessel_I(1/2, x)
sqrt(2)*sqrt(1/(pi*x))*sinh(x)
sage: eq = bessel_I(1/2, x) == bessel_I(0.5, x)
sage: eq.test_relation()
True
sage: bessel_I(-1/2, x)
sqrt(2)*sqrt(1/(pi*x))*cosh(x)
sage: eq = bessel_I(-1/2, x) == bessel_I(-0.5, x)
sage: eq.test_relation()
True
```

Examples of asymptotic behavior:

```
sage: limit(bessel_I(0, x), x=oo)  # needs sage.symbolic
+Infinity
```

1.14. Bessel functions
Functions, Release 10.4

(continued from previous page)

```python
sage: limit(bessel_I(0, x), x=0) # needs sage.symbolic
1
```

```python
>>> from sage.all import *

>>> limit(bessel_I(Integer(0), x), x=oo) # needs sage.symbolic
+Infinity

>>> limit(bessel_I(Integer(0), x), x=Integer(0)) # needs sage.symbolic
1
```

High precision and complex valued inputs:

```python
sage: bessel_I(0, 1).n(128) # needs sage.symbolic
1.266065877752008335982446252147175376
sage: bessel_I(0, RealField(200)(1)) # needs sage.rings.real_mpfr
1.266065877752008335982446252147175376076703113549622068081
sage: bessel_I(0, ComplexField(200)(0.5+I)) # needs sage.symbolic
0.8064435758349361947242851841501947244285184150194724428518415019
+ 0.22686958987911161413974301487525043310874687430711021434*I
```

```python
>>> from sage.all import *

>>> plot(bessel_I(Integer(1), x), (x, Integer(0), Integer(5)), color='blue') # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive

>>> complex_plot(bessel_I(Integer(1), x), (-Integer(5), Integer(5)), plot_points=Integer(20)) # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

Visualization (set plot_points to a higher value to get more detail):

ALGORITHM:
Numerical evaluation is handled by the mpmath library. Symbolics are handled by a combination of Maxima and Sage (Ginac/Pynac).

REFERENCES:
- [AS-Bessel]
- [DLMF-Bessel]
- [WP-Bessel]

class sage.functions.bessel.Function_Bessel_J

Bases: BuiltinFunction

The Bessel J Function, denoted by \( \text{bessel}_J(\nu, x) \) or \( J_\nu(x) \). As a Taylor series about \( x = 0 \) it is equal to:

\[
J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k! (k + \nu + 1)} \left( \frac{x}{2} \right)^{2k+\nu}
\]

The parameter \( \nu \) is called the order and may be any real or complex number; however, integer and half-integer values are most common. It is defined for all complex numbers \( x \) when \( \nu \) is an integer or greater than zero and it diverges as \( x \to 0 \) for negative non-integer values of \( \nu \).

For integer orders \( \nu = n \) there is an integral representation:

\[
J_n(x) = \frac{1}{\pi} \int_0^\pi \cos(nt - x \sin(t)) \, dt
\]

This function also arises as a special case of the hypergeometric function \( {}_0F_1 \):

\[
J_\nu(x) = \frac{x^n}{2^n \Gamma(n + 1)} {}_0F_1 \left( \nu + 1, -\frac{x^2}{4} \right).
\]

EXAMPLES:

```python
sage: bessel_J(1.0, 1.0)  # needs mpmath
0.440050585744933

sage: # needs sage.symbolic
sage: bessel_J(2, I).n(digits=30)  # needs sage.symbolic
-0.135747669767038281182852569995

sage: bessel_J(1, x)

sage: n = var('n')

sage: bessel_J(n, x)

.. note::

   sage: from sage.all import *
   sage: bessel_J(RealNumber('1.0'), RealNumber('1.0'))  # needs mpmath
   0.440050585744933

.. note::

   sage: bessel_J(Integer(2), I).n(digits=30)  # needs sage.symbolic
   -0.135747669767038281182852569995

   sage: bessel_J(Integer(1), x)
   sage: n = var('n')
   sage: bessel_J(n, x)
```

1.14. Bessel functions 197
Examples of symbolic manipulation:

```
sage: # needs sage.symbolic
sage: a = bessel_J(pi, bessel_J(1, I)); a
bessel_J(pi, bessel_J(1, I))
sage: N(a, digits=20)
0.00059023706363796717363 - 0.0026098820470081958110*I
sage: f = bessel_J(2, x)
sage: f.diff(x)
-1/2*bessel_J(3, x) + 1/2*bessel_J(1, x)
```

Comparison to a well-known integral representation of $J_1(1)$:

```
sage: A = numerical_integral(1/pi*cos(x - sin(x)), 0, pi)
# abs tol 1e-14
sage: A[0] # needs sage.symbolic
0.44005058574493355
sage: bessel_J(1.0, 1.0) - A[0] < 1e-15 # needs sage.symbolic
True
```

Integration is supported directly and through Maxima:

```
sage: f = bessel_J(2, x)
# needs sage.symbolic
sage: f.integrate(x)
1/24*x^3*hypergeometric((3/2,), (5/2, 3), -1/4*x^2)
```

Visualization (set plot_points to a higher value to get more detail):
### sage: plot(bessel_J(1,x), (x,0,5), color='blue')

```
<Graphics object consisting of 1 graphics primitive>
```

`# needs sage.plot sage.symbolic`

### sage: complex_plot(bessel_J(1, x), (-5, 5), (-5, 5), plot_points=20)

```
<Graphics object consisting of 1 graphics primitive>
```

`# needs sage.plot sage.symbolic`

```
>>> from sage.all import *

>>> plot(bessel_J(Integer(1),x), (x,Integer(0),Integer(5)), color='blue')  # needs sage.plot sage.symbolic

<Graphics object consisting of 1 graphics primitive>
```

``` complex_plot(bessel_J(Integer(1), x), (Integer(-5), Integer(5)), (Integer(-5),
Integer(5)), plot_points=Integer(20))  # needs sage.plot sage.symbolic

<Graphics object consisting of 1 graphics primitive>
```

ALGORITHM:

Numerical evaluation is handled by the mpmath library. Symbolics are handled by a combination of Maxima and Sage (Ginac/Pynac).

Check whether the return value is real whenever the argument is real (Issue #10251):

```
sage: bessel_J(5, 1.5) in RR  # needs mpmath
True
```

``` from sage.all import *

>>> bessel_J(Integer(5), RealNumber('1.5')) in RR  # needs mpmath
True
```

REFERENCES:

- [AS-Bessel]
- [DLMF-Bessel]
- [AS-Bessel]

**class** `sage.functions.bessel.Function_Bessel_K`

Bases: `BuiltinFunction`

The Bessel K function, or the modified Bessel function of the second kind.

**DEFINITION:**

\[
K_\nu(x) = \frac{\pi}{2} \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\nu \pi)}
\]

**EXAMPLES:**

```
sage: bessel_K(1.0, 1.0)  # needs mpmath

0.601907230197235
```

```
sage: # needs sage.symbolic

sage: bessel_K(1, x)

bessel_K(1, x)
```

(continues on next page)
sage: n = var(’n’)  
sage: bessel_K(n, x)  
bessel_K(n, x)  
sage: bessel_K(2, I).n()  
-2.59288617549120 + 0.180489972066962*I

Examples of symbolic manipulation:

sage: # needs sage.symbolic  
sage: a = bessel_K(pi, bessel_K(1, I)); a  
bessel_K(pi, bessel_K(1, I))  
sage: N(a, digits=20)  
3.8507583115005220156 + 0.068528298579883425456*I  
sage: f = bessel_K(2, x)  
sage: f.diff(x)  
-1/2*bessel_K(3, x) - 1/2*bessel_K(1, x)  
sage: bessel_K(1/2, x)  
sqrt(1/2)*sqrt(pi)*e^(-x)/sqrt(x)  
sage: bessel_K(1/2, -1)  
-I*sqrt(1/2)*sqrt(pi)*e  
sage: bessel_K(1/2, 1)  
sqrt(1/2)*sqrt(pi)*e^(-1)

Examples of asymptotic behavior:

>>> from sage.all import *  
>>> bessel_K(RealNumber(’1.0’), RealNumber(’1.0’))  
˓→  # needs mpmath  
0.601907230197235

>>> # needs sage.symbolic  
>>> bessel_K(Integer(1), x)  
bessel_K(1, x)  
>>> n = var(’n’)  
>>> bessel_K(n, x)  
bessel_K(n, x)  
>>> bessel_K(Integer(2), I).n()  
-2.59288617549120 + 0.180489972066962*I

>>> from sage.all import *  
>>> # needs sage.symbolic  
>>> a = bessel_K(pi, bessel_K(Integer(1), I)); a  
bessel_K(pi, bessel_K(Integer(1), I))  
>>> N(a, digits=Integer(20))  
3.8507583115005220156 + 0.068528298579883425456*I  
>>> f = bessel_K(2, x)  
>>> f.diff(x)  
-1/2*bessel_K(3, x) - 1/2*bessel_K(1, x)  
>>> bessel_K(Integer(1)/Integer(2), Integer(1))  
sqrt(1/2)*sqrt(pi)*e^(-1)

Examples of asymmetric behavior:
```
sage: bessel_K(0, 0.0)  # needs mpmath
+infinity
sage: limit(bessel_K(0, x), x=0)  # needs sage.symbolic
+Infinity
sage: limit(bessel_K(0, x), x=oo)  # needs sage.symbolic
0

>>> from sage.all import *

>>> bessel_K(Integer(0), RealNumber('0.0'))  # needs mpmath
+infinity
>>> limit(bessel_K(Integer(0), x), x=Integer(0))  # needs sage.symbolic
+Infinity
>>> limit(bessel_K(Integer(0), x), x=oo)  # needs sage.symbolic
0

High precision and complex valued inputs:

```
sage: bessel_K(0, 1).n(128)  # needs sage.symbolic
0.42102443824070833333562737921260903614
sage: bessel_K(0, RealField(200)(1))  # needs sage.rings.real_mpfr
0.42102443824070833333562737921260903613621974822666047229897
sage: bessel_K(0, ComplexField(200)(0.5+I))  # needs sage.rings.real_mpfr, sage.symbolic
0.0583659790931038640837531164336004814715516692187818271179
- 0.67645499731334483535184142196073004335768129348518210260256*I

>>> from sage.all import *

>>> bessel_K(Integer(0), Integer(1)).n(Integer(128))  # needs sage.symbolic
0.42102443824070833333562737921260903614
>>> bessel_K(Integer(0), RealField(Integer(200))(Integer(1)))  # needs sage.rings.real_mpfr
0.42102443824070833333562737921260903613621974822666047229897
>>> bessel_K(Integer(0), ComplexField(Integer(200))(RealNumber('0.5')+I))  # needs sage.rings.real_mpfr, sage.symbolic
0.0583659790931038640837531164336004814715516692187818271179
- 0.67645499731334483535184142196073004335768129348518210260256*I

Visualization (set plot_points to a higher value to get more detail):
```
sage: plot(bessel_K(1,x), (x, 0,5), color='blue')  # needs sage.plot, sage.symbolic
Graphics object consisting of 1 graphics primitive
sage: complex_plot(bessel_K(1, x), (-5, 5), (-5, 5), plot_points=20)  # needs sage.plot, sage.symbolic
Graphics object consisting of 1 graphics primitive

>>> from sage.all import *
```
>>> plot(bessel_K(Integer(1), x), (x, Integer(0), Integer(5)), color='blue')
   # needs sage.plot sage.symbolic
   Graphics object consisting of 1 graphics primitive

>>> complex_plot(bessel_K(Integer(1), x), (-Integer(5), Integer(5)), (-Integer(5),
   Integer(5)), plot_points=Integer(20))
   # needs sage.plot sage.
   symbolic
   Graphics object consisting of 1 graphics primitive

ALGORITHM:

Numerical evaluation is handled by the mpmath library. Symbolics are handled by a combination of
Maxima and Sage (Ginac/Pynac).

REFERENCES:

• [AS-Bessel]
• [DLMF-Bessel]
• [WP-Bessel]

class sage.functions.bessel.Function_Bessel_Y

Bases: BuiltinFunction

The Bessel Y functions, also known as the Bessel functions of the second kind, Weber functions, or Neumann
functions.

\( Y_\nu(z) \) is a holomorphic function of \( z \) on the complex plane, cut along the negative real axis. It is singular at \( z = 0 \).
When \( z \) is fixed, \( Y_\nu(z) \) is an entire function of the order \( \nu \).

DEFINITION:

\[
Y_\nu(z) = \frac{J_\nu(z) \cos(\nu z) - J_{-\nu}(z)}{\sin(\nu z)}
\]

Its derivative with respect to \( z \) is:

\[
\frac{d}{dz} Y_\nu(z) = \frac{1}{z^n} \left( z^n Y_{\nu-1}(z) - n z^{n-1} Y_\nu(z) \right)
\]

EXAMPLES:

sage: bessel_Y(1, x)    #...
   needs sage.symbolic

bessel_Y(1, x)

sage: bessel_Y(1.0, 1.0)  #...
   needs mpmath

-0.781212821300289

sage: # needs sage.symbolic

sage: n = var('n')
sage: bessel_Y(n, x)
bessel_Y(n, x)

sage: bessel_Y(2, I).n()
1.03440456978312 - 0.13574766976767038*I

sage: bessel_Y(0, 0).n()  # infinity
-infinity

sage: bessel_Y(0, 1).n(128)
0.088256964215676957982926766023515162828
>>> from sage.all import *
>>> bessel_Y(Integer(1), x)  # needs sage.symbolic
bessel_Y(1, x)
>>> bessel_Y(RealNumber('1.0'), RealNumber('1.0'))  # needs mpmath
-0.781212821300289

>>> # needs sage.symbolic
>>> n = var('n')
>>> bessel_Y(n, x)
bessel_Y(n, x)

Examples of symbolic manipulation:

sage: a = bessel_Y(pi, bessel_Y(1, I)); a
bessel_Y(pi, bessel_Y(1, I))
sage: N(a, digits=20)
4.2059146571791095708 + 21.307914215321993526*I

sage: f = bessel_Y(2, x)
sage: f.diff(x)
-1/2*bessel_Y(3, x) + 1/2*bessel_Y(1, x)

High precision and complex valued inputs (see Issue #4230):

sage: bessel_Y(0, 1).n(128)  # needs sage.symbolic
0.088256964215676957982926766023515162828

sage: bessel_Y(0, RealField(200)(1))  # needs sage.rings.real_mpfr
0.08825696421567695798292676602351516282781752309067554671104

sage: bessel_Y(0, ComplexField(200)(0.5+I))  # needs sage.symbolic
0.07763160184438051408593468823822434235010300228009867784073 + 1.014233604991606915264467768282326441579314239591288411739*I

>>> from sage.all import *
>>> bessel_Y(Integer(0), Integer(1)).n(Integer(128))  # needs sage.symbolic
0.088256964215676957982926766023515162828

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```
>>> bessel_Y(Integer(0), RealField(Integer(200))(Integer(1)))
˓→ # needs sage.rings.real_mpfr
0.08825696421567695798292676602351516287817523090675546711044
>>> bessel_Y(Integer(0), ComplexField(Integer(200))(RealNumber('0.5')+I))
˓→ # needs sage.symbolic
0.07776160184438051408593468823822433235010300228009867784073
+ 1.014233604991606915264467768282326441579314239591288411793*I
```

Visualization (set plot_points to a higher value to get more detail):

```
sage: plot(bessel_Y(1, x), (x, 0, 5), color =blue)
˓→ # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
sage: complex_plot(bessel_Y(1, x), ( -5, 5), ( -5, 5), plot_points=20)
˓→ needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
```

```
>>> from sage.all import *
>>> plot(bessel_Y(Integer(1), x), (x, Integer(0), Integer(5)), color =blue)    # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive
>>> complex_plot(bessel_Y(Integer(1), x), ( -Integer(5), Integer(5)), ( -Integer(5),
˓→ Integer(5)), plot_points=Integer(20)) # needs sage.plot sage.symbolic
˓→ symbolic
Graphics object consisting of 1 graphics primitive
```

ALGORITHM:

Numerical evaluation is handled by the mpmath library. Symbolics are handled by a combination of Maxima and Sage (Ginac/Pynac).

REFERENCES:

• [AS-Bessel]
• [DLMF-Bessel]
• [WP-Bessel]

class sage.functions.bessel.Function_Hankel1
Bases: BuiltinFunction

The Hankel function of the first kind

DEFINITION:

\[ H^{(1)}_\nu(z) = J_\nu(z) + iY_\nu(z) \]

EXAMPLES:

```
sage: hankel1(3, x)  # needs sage.symbolic
hankel1(3, x)
sage: hankel1(3, 4.)  # needs mpmath
0.430171473875622 - 0.182022115953485*I
sage: latex(hankel1(3, x))  # needs sage.symbolic
H_3^{(1)}(x)
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```python
sage: hankel1(3., x).series(x == 2, 10).subs(x=3).n() # abs tol 1e-12
\rightarrow needs sage.symbolic
0.309062682819597 - 0.512591541605233*I
```

```python
sage: hankel1(3, x) #...
\rightarrow needs mpmath
0.309062722255252 - 0.538541616105032*I
```

```python
>>> from sage.all import *
>>> hankel1(Integer(3), x) # needs sage.symbolic
hankel1(3, x)
```

```python
>>> hankel1(Integer(3), RealNumber('4.')) # needs mpmath
0.430171473875622 - 0.182022115953485*I
```

```python
>>> latex(hankel1(Integer(3), x)) # needs sage.symbolic
H_{3}^{(1)}(x)
```

```python
>>> hankel1(RealNumber('3.'), x) .series(x == Integer(2), Integer(10)).
\rightarrow needs sage.symbolic
subs(x=Integer(3)).n() # abs tol 1e-12
0.309062682819597 - 0.512591541605234*I
```

```python
>>> hankel1(Integer(3), RealNumber('3.')) # needs mpmath
0.309062722255252 + 0.538541616105032*I
```

REFERENCES:

• [AS-Bessel] see 9.1.6

class sage.functions.bessel.Function_Hankel2

Bases: BuiltinFunction

The Hankel function of the second kind

DEFINITION:

\[ H_{\nu}^{(2)}(z) = J_{\nu}(z) - iY_{\nu}(z) \]

EXAMPLES:

```python
sage: hankel2(3, x) #...
\rightarrow needs sage.symbolic
hankel2(3, x)
```

```python
sage: hankel2(3, 4.) #...
\rightarrow needs mpmath
0.430171473875622 + 0.182022115953485*I
```

```python
sage: latex(hankel2(3, x)) # needs sage.symbolic
H_{3}^{(2)}(x)
```

```python
>>> hankel2(RealNumber('3.'), x) .series(x == Integer(2), Integer(10)).
\rightarrow needs sage.symbolic
subs(x=Integer(3)).n() # abs tol 1e-12
0.309062682819597 + 0.512591541605234*I
```

```python
>>> hankel2(Integer(3), RealNumber('3.')) # needs mpmath
0.309062722255252 + 0.538541616105032*I
```

>>> from sage.all import *
>>> hankel2(Integer(3), x) #...
```

(continues on next page)

1.14. Bessel functions 205
Functions, Release 10.4

(continued from previous page)

\[ \texttt{hankel2}(3, x) \]
\[
\begin{array}{l}
\texttt{hankel2}(\texttt{Integer}(3), \texttt{RealNumber}(4.)) \rightarrow \texttt{0.430171473875622 + 0.182022115953485*I} \\
\texttt{latex(hankel2}(\texttt{Integer}(3), x)) \rightarrow \texttt{# needs sage.symbolic} \\
H_{3}^{(2)}(x) \rightarrow \texttt{# needs mpmath} \\
\texttt{hankel2}(\texttt{RealNumber(3.), x}) .\texttt{series(x == Integer(2), Integer(10))}.subs(x=Integer(3)).n() \rightarrow \texttt{0.309062722255252 + 0.538541616105032*I} \\
\end{array}
\]

REFERENCES:

• [AS-Bessel] see 9.1.6

**class** sage.functions.bessel.Function_Struve_H

Bases: BuiltinFunction

The Struve functions, solutions to the non-homogeneous Bessel differential equation:

\[
x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - \alpha^2)y = \frac{4\left(\frac{x}{\pi}\right)^{\alpha+1}}{\sqrt{\pi} \Gamma(\alpha + \frac{1}{2})},
\]

\[H_{\alpha}(x) = y(x)\]

EXAMPLES:

\[
\begin{array}{l}
\texttt{sage: struve_H(-1/2, x)} \rightarrow \sqrt{2} \sqrt{1/(\pi x)} \sin(x) \\
\texttt{sage: struve_H(2, x)} \rightarrow \sqrt{2} \sqrt{1/(\pi x)} \sin(x) \\
\texttt{sage: struve_H(1/2, pi).n()} \rightarrow 0.900316316157106 \\
\end{array}
\]

\[
\begin{array}{l}
\texttt{from sage.all import *} \\
\texttt{struve_H(-Integer(1)/Integer(2), x)} \rightarrow \sqrt{2} \sqrt{1/(\pi x)} \sin(x) \\
\texttt{struve_H(2, x)} \rightarrow \sqrt{2} \sqrt{1/(\pi x)} \sin(x) \\
\texttt{struve_H(1/2, pi).n()} \rightarrow 0.900316316157106 \\
\end{array}
\]

REFERENCES:

• [AS-Struve]
• [DLMF-Struve]
• [WP-Struve]
class sage.functions.bessel.Function_Struve_L

Bases: BuiltinFunction

The modified Struve functions.

\[ L_\alpha(x) = -i \cdot e^{-\frac{\alpha \pi}{2}} \cdot H_\alpha(ix) \]

EXAMPLES:

```python
sage: struve_L(2, x)  # needs sage.symbolic
struve_L(2, x)
sage: struve_L(1/2, pi).n()  # needs sage.symbolic
4.76805417696286
sage: diff(struve_L(1, x), x)  # needs sage.symbolic
1/3*x/pi - 1/2*struve_L(2, x) + 1/2*struve_L(0, x)
```

REFERENCES:
- [AS-Struve]
- [DLMF-Struve]
- [WP-Struve]

class sage.functions.bessel.SphericalBesselJ

Bases: BuiltinFunction

The spherical Bessel function of the first kind

**DEFINITION:**

\[ j_n(z) = \sqrt{\frac{\pi}{2z}} J_{n+\frac{1}{2}}(z) \]

**EXAMPLES:**

```python
sage: spherical_bessel_J(3, 3.)  # needs mpmath
0.152051662030533
sage: spherical_bessel_J(2., 3.)  # rel tol 1e-10  # needs mpmath
0.2986374970757335
```

(continues on next page)
sage: spherical_bessel_J(3 + 0.2 * I, 3)
0.150770999183897 - 0.0260662466510632*I
sage: spherical_bessel_J(3, x).series(x == 2, 10).subs(x=3).n()
0.152051648665037
sage: spherical_bessel_J(4, x).simplify()
-((45/x^2 - 105/x^4 - 1)*sin(x) + 5*(21/x^2 - 2)*cos(x)/x)/x
sage: integrate(spherical_bessel_J(1,x)^2,(x,0,oo))
1/6*pi
sage: latex(spherical_bessel_J(4, x))
\(j_{4}(x)\)

>>> from sage.all import *
>>> spherical_bessel_J(Integer(3), RealNumber('3.'))  # needs mpmath
0.152051662030533
>>> spherical_bessel_J(RealNumber('2.'),RealNumber('3.'))  # rel tol 1e-10  # needs mpmath
0.2986374970757335

>>> # needs sage.symbolic
>>> spherical_bessel_J(Integer(3), x)
spherical_bessel_J(3, x)
>>> spherical_bessel_J(Integer(3) + RealNumber('0.2') * I, Integer(3))
0.150770999183897 - 0.0260662466510632*I
>>> spherical_bessel_J(Integer(3), x).series(x == Integer(2), Integer(10)).
...subs(x=Integer(3)).n()
0.152051648665037
>>> spherical_bessel_J(Integer(4), x).simplify()
-((45/x^2 - 105/x^4 - 1)*sin(x) + 5*(21/x^2 - 2)*cos(x)/x)/x
>>> integrate(spherical_bessel_J(Integer(1),x)**Integer(2),(x,Integer(0),oo))
1/6*pi
>>> latex(spherical_bessel_J(Integer(4), x))
\(j_{4}(x)\)

REFERENCES:
• [AS-Spherical]
• [DLMF-Bessel]
• [WP-Bessel]

class sage.functions.bessel.SphericalBesselY

Bases: BuiltinFunction

The spherical Bessel function of the second kind

\[ y_n(z) = \sqrt{\frac{\pi}{2z}} Y_{n+\frac{1}{2}}(z) \]

EXAMPLES:

sage: # needs sage.symbolic
sage: spherical_bessel_Y(3, x)
spherical_bessel_Y(3, x)

sage: spherical_bessel_Y(3 + 0.2 * I, 3)
-0.505215297588210 - 0.0508835883281404*I
\[\text{sage: } \text{spherical}_\text{bessel}_\text{Y}(-3, x).\text{simplify()} \]
\[((3/x^2 - 1)*\sin(x) - 3*\cos(x)/x)/x\]
\[\text{sage: } \text{spherical}_\text{bessel}_\text{Y}(3 + 2*I, 5 - 0.2*I)\]
\[-0.270205813266440 - 0.615994702714957*I\]
\[\text{sage: } \text{integrate(} \text{spherical}_\text{bessel}_\text{Y}(0, x), x)\]
\[-1/2^*\text{Ei}(I*x) - 1/2^*\text{Ei}(-I*x)\]
\[\text{sage: } \text{integrate(} \text{spherical}_\text{bessel}_\text{Y}(1,x)\text{^2},(x,0,oo))\]
\[-1/6^*\pi\]
\[\text{sage: } \text{latex(} \text{spherical}_\text{bessel}_\text{Y}(0, x)\text{)}\]
\[y_{0}\left(x\right)\]

>>> from sage.all import *
>>> # needs sage.symbolic
>>> \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(3), x)
\text{spherical}_\text{bessel}_\text{Y}(3, x)
>>> \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(3) + \text{RealNumber}(0.2)\times I, \text{Integer}(3))
-0.505215297588210 - 0.0508835883281404*I
>>> \text{spherical}_\text{bessel}_\text{Y}(-\text{Integer}(3), x).\text{simplify()}\]
\[((3/x^2 - 1)*\sin(x) - 3*\cos(x)/x)/x\]
>>> \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(3) + \text{Integer}(2)\times I, \text{Integer}(5) - \text{RealNumber}(0.2)\times I)\]
-0.270205813266440 - 0.615994702714957*I
>>> \text{integrate(} \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(0), x), x)\]
\[-1/2^*\text{Ei}(I*x) - 1/2^*\text{Ei}(-I*x)\]
>>> \text{integrate(} \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(1),x)^\text{**}\text{Integer}(2),(x,\text{Integer}(0),\text{oo})\]
\[-1/6^*\pi\]
>>> \text{latex(} \text{spherical}_\text{bessel}_\text{Y}(\text{Integer}(0), x)\text{)}
y_{0}\left(x\right)\left\{\text{left}(x\text{right})\right\}

REFERENCES:
- [AS-Spherical]
- [DLMF-Bessel]
- [WP-Bessel]

\text{class sage.functions.bessel.SphericalHankel1}
\text{Bases: BuiltinFunction}

The spherical Hankel function of the first kind

\text{DEFINITION:}
\[h_n^{(1)}(z) = \sqrt{\frac{\pi}{2z}} H_{n+\frac{1}{2}}(z)\]

\text{EXAMPLES:}
\[\text{sage: } \# \text{ needs sage.symbolic}\]
\[\text{sage: } \text{spherical}_\text{hankel1}(3, x)\]
\[\text{spherical}_\text{hankel1}(3, x)\]
\[\text{sage: } \text{spherical}_\text{hankel1}(3 + 0.2\times I, 3)\]
0.201654587512037 - 0.53128154239273*I
\[\text{sage: } \text{spherical}_\text{hankel1}(1, x).\text{simplify()}\]
\[-(x + I)\times e^{(I*x)/(x^2}\]
\[\text{sage: } \text{spherical}_\text{hankel1}(3 + 2\times I, 5 - 0.2\times I)\]
1.25375216869913 - 0.518011435921789*I
\[ \text{sage: integrate(spherical_hankel1(3, x), x)} \]
\[ Ei(I*x) - 6*gamma(-1, -I*x) - 15*gamma(-2, -I*x) - 15*gamma(-3, -I*x) \]
\[ \text{sage: latex(spherical_hankel1(3, x))} \]
\[ h_{3}^{(1)}(x) \]

>>> from sage.all import *
>>> # needs sage.symbolic
>>> spherical_hankel1(Integer(3), x)
spherical_hankel1(3, x)
>>> spherical_hankel1(Integer(3) + RealNumber('0.2') * I, Integer(3))
0.201654587512037 - 0.531281544239273*I
>>> spherical_hankel1(Integer(1), x).simplify()
-(x + I)*e^(I*x)/x^2
>>> spherical_hankel1(Integer(3) + Integer(2) * I, Integer(5) - RealNumber('0.2')*I)
1.25375216869913 - 0.518011435921789*I
>>> integrate(spherical_hankel1(Integer(3), x), x)
Ei(-I*x) - 6*gamma(-1, I*x) - 15*gamma(-2, I*x) - 15*gamma(-3, I*x)
>>> latex(spherical_hankel1(Integer(3), x))
\[ h_{3}^{(1)}(x) \]

REFERENCES:
- [AS-Spherical]
- [DLMF-Bessel]
- [WP-Bessel]

\textbf{class sage.functions.bessel.SphericalHankel2}

Bases: BuiltinFunction

The spherical Hankel function of the second kind

DEFINITION:

\[ h_n^{(2)}(z) = \sqrt{\frac{\pi}{2z}} H_n^{(2)}(z) \]

EXAMPLES:

\[ \text{sage: # needs sage.symbolic} \]
\[ \text{sage: spherical_hankel2(3, x)} \]
\[ \text{spherical_hankel2(3, x)} \]
\[ \text{sage: spherical_hankel2(3 + 0.2 * I, 3)} \]
\[ 0.0998874108557565 + 0.479149050937147*I \]
\[ \text{sage: spherical_hankel2(1, x).simplify()} \]
\[ -(x - I)*e^{(-I*x)}/x^2 \]
\[ \text{sage: spherical_hankel2(2,i).simplify()} \]
\[ -e \]
\[ \text{sage: spherical_hankel2(2,x).simplify()} \]
\[ (-I*x^2 - 3*x + 3*I)*e^{(-I*x)}/x^3 \]
\[ \text{sage: spherical_hankel2(3 + 2*I, 5 - 0.2*I)} \]
\[ 0.0217627632692163 + 0.0224001906110906*I \]
\[ \text{sage: integrate(spherical_hankel2(3, x), x)} \]
\[ Ei(-I*x) - 6*gamma(-1, I*x) - 15*gamma(-2, I*x) - 15*gamma(-3, I*x) \]
\[ \text{sage: latex(spherical_hankel2(3, x))} \]
\[ h_{3}^{(2)}(x) \]
REFERENCES:

• [AS-Spherical]
• [DLMF-Bessel]
• [WP-Bessel]

sage.functions.bessel.spherical_bessel_f(F, n, z)

Numerically evaluate the spherical version, \( f \), of the Bessel function \( F \) by computing

\[
 f_n(z) = \sqrt{\frac{1}{2} \pi} \frac{F_{n+\frac{1}{2}}(z)}{z^{n+\frac{1}{2}}}.
\]

According to Abramowitz & Stegun, this identity holds for the Bessel functions \( J, Y, K, I, H^{(1)}, \) and \( H^{(2)} \).

EXAMPLES:

```python
sage: from sage.functions.bessel import spherical_bessel_f
sage: spherical_bessel_f('besselj', 3, 4) # needs mpmath
mpf('0.22924385795503024')
```

```python
sage: spherical_bessel_f('hankel1', 3, 4) # needs mpmath
mpc(real='0.22924385795503024', imag='-0.21864196590306359')
```

```python
from sage.all import *
from sage.functions.bessel import spherical_bessel_f
spherical_bessel_f('besselj', Integer(3), Integer(4)) # needs mpmath
mpf('0.22924385795503024')
```

```python
spherical_bessel_f('hankel1', Integer(3), Integer(4)) # needs mpmath
mpc(real='0.22924385795503024', imag='-0.21864196590306359')
```
1.15 Exponential integrals

AUTHORS:

• Benjamin Jones (2011-06-12)

This module provides easy access to many exponential integral special functions. It utilizes Maxima’s special functions package and the mpmath library.

REFERENCES:

• [AS1964] Abramowitz and Stegun: Handbook of Mathematical Functions
• Wikipedia article Exponential_integral
• Online Encyclopedia of Special Function: http://algo.inria.fr/esf/index.html
• NIST Digital Library of Mathematical Functions: https://dlmf.nist.gov/
• Maxima special functions package
• mpmath library

AUTHORS:

• Benjamin Jones

Implementation of the classes Function_exp_integral_*. 

• David Joyner and William Stein

Authors of the code which was moved from special.py and trans.py. Implementation of exp_int() (from sage/functions/special.py). Implementation of exponential_integral_1() (from sage/functions/transcendental.py).

class sage.functions.exp_integral.Function_cos_integral

Bases: BuiltinFunction

The trigonometric integral Ci(z) defined by

\[ \text{Ci}(z) = \gamma + \log(z) + \int_0^z \frac{\cos(t) - 1}{t} \, dt, \]

where \( \gamma \) is the Euler gamma constant (euler_gamma in Sage), see [AS1964] 5.2.1.

EXAMPLES:

```
sage: z = var('z')  #...

sage: cos_integral(z)  #...

sage: cos_integral(3.0)  #...

0.119629786008000

sage: N(cos_integral(0))  #...

-infinity
```
Functions, Release 10.4

```python
>>> from sage.all import *

>>> z = var('z')
˓→ needs sage.symbolic

>>> cos_integral(z)
˓→ needs sage.symbolic

cos_integral(z)

>>> cos_integral(RealNumber('3.0'))
˓→ # needs mpmath
0.119629786008000

>>> cos_integral(Integer(0))
˓→ # needs sage.symbolic

cos_integral(0)

>>> N(cos_integral(Integer(0)))
˓→ # needs mpmath
-\infty

Numericalevaluationforrealandcomplexargumentsishandledusingmpmath:

```
sage: cos_integral(3.0)
˓→ needs mpmath
0.119629786008000
```

Thealias \texttt{Ci}canbeusedinsteadof \texttt{cos_integral}:

```
sage: Ci(3.0)
˓→ needs mpmath
0.119629786008000
```

Compare \texttt{cos_integral(3.0)} to the definition of the value using numerical integration:

```
sage: a = numerical_integral((cos(x)-1)/x, 0, 3)[0]
˓→ needs sage.symbolic

sage: abs(N(euler_gamma + log(3)) + a - N(cos_integral(3.0))) < 1e-14
˓→ needs sage.symbolic
True
```

Arbitrary precision and complex arguments are handled:

```
sage: a = numerical_integral((cos(x)-Integer(1))/x, Integer(0),
˓→ Integer(0))[Integer(0)] # needs sage.symbolic

sage: abs(N(euler_gamma + log(Integer(3))) + a - N(cos_integral(RealNumber('3.0')))) < RealNumber('1e-14') # needs sage.symbolic
True
```

1.15. Exponential integrals
The limit \( \text{Ci}(z) \) as \( z \to \infty \) is zero:

```
sage: N(cos_integral(1e23))  # needs mpmath
-3.24053937643003e-24
```
ALGORITHM:

Numerical evaluation is handled using mpmath, but symbolic are handled by Sage and Maxima.

REFERENCES:
- Wikipedia article Trigonometric_integral
- mpmath documentation: ci

class sage.functions.exp_integral.Function_cosh_integral
Bases: BuiltinFunction

The trigonometric integral $\text{Chi}(z)$ defined by

$$\text{Chi}(z) = \gamma + \log(z) + \int_{0}^{z} \frac{\cosh(t) - 1}{t} \, dt,$$

see [AS1964] 5.2.4.

EXAMPLES:

```python
sage: z = var('z')  # needs sage.symbolic
sage: cosh_integral(z)  # needs sage.symbolic
cosh_integral(z)
sage: cosh_integral(3.0)  # needs mpmath
4.96039209476561
```

Numerical evaluation for real and complex arguments is handled using mpmath:

```python
sage: cosh_integral(1.0)  # needs mpmath
0.837866940980208
```
The alias \( \text{Chi} \) can be used instead of \( \text{cosh\_integral} \):

\[
\text{sage: Chi}(1.0) \quad \# \text{ needs mpmath}
\]
\[
0.837866940980208
\]

Here is an example from the mpmath documentation:

\[
\text{sage: } f(x) = \text{cosh\_integral}(x) \quad \# \ldots
\]
\[
\text{sage: find\_root}(f, 0.1, 1.0) \quad \# \ldots
\]
\[
0.523822571389...
\]

Compare \( \text{cosh\_integral}(3.0) \) to the definition of the value using numerical integration:

\[
\text{sage: } a = \text{numerical\_integral}((\text{cosh}(x)-1)/x, 0, 3)[0] \quad \# \ldots
\]
\[
\text{sage: } \text{abs}(\text{N}(\text{euler\_gamma} + \log(3)) + a - \text{N}(\text{cosh\_integral}(3.0))) < 1e-14 \quad \# \ldots
\]
\[
\text{True}
\]

Arbitrary precision and complex arguments are handled:

\[
\text{sage: } \text{N}(\text{cosh\_integral}(3), \text{digits}=30) \quad \# \ldots
\]
\[
4.96039209476560976029791763669
\]
\[
\text{sage: } \text{cosh\_integral}(\text{ComplexField}(100)(3+I)) \quad \# \ldots
\]
\[
3.909672309968641712843516794 + 3.0547519627014217273323873274*I
\]
The limit of $\text{Chi}(z)$ as $z \to \infty$ is $\infty$:

```python
sage: N(cosh_integral(Infinity))
# needs mpmath
+infinity
```

Symbolic derivatives and integrals are handled by Sage and Maxima:

```python
sage: # needs sage.symbolic
sage: x = var('x')
sage: f = cosh_integral(x)
sage: f.diff(x)
cosh(x)/x
sage: f.integrate(x)
x*cosh_integral(x) - sinh(x)
```

ALGORITHM:

Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

REFERENCES:

- Wikipedia article Trigonometric_integral
- mpmath documentation: chi

```python
class sage.functions.exp_integral.Function_exp_integral
    Bases: BuiltinFunction

The generalized complex exponential integral $\text{Ei}(z)$ defined by

$$\text{Ei}(x) = \int_{-\infty}^{x} \frac{e^t}{t} dt$$

for $x > 0$ and for complex arguments by analytic continuation, see [AS1964] 5.1.2.

EXAMPLES:
The precision for the result is deduced from the precision of the input. Convert the input to a higher precision explicitly if a result with higher precision is desired:

```python
sage: Ei(RealField(300)(1.1))  # needs sage.rings.real_mpfr
2.7213988802320235
```
2. \[ \text{from sage.all import } * \]
>>> E1(RealField(Integer(300))(RealNumber('1.1')))  
\[ \rightarrow \text{# needs sage.rings.real_mpfr} \]

ALGORITHM: Uses mpmath.

class sage.functions.exp_integral.Function_exp_integral_e

Bases: BuiltinFunction

The generalized complex exponential integral \( E_n(z) \) defined by
\[
E_n(z) = \int_1^\infty \frac{e^{-zt}}{t^n} \, dt
\]
for complex numbers \( n \) and \( z \), see [AS1964] 5.1.4.

The special case where \( n = 1 \) is denoted in Sage by \( \text{exp_integral_e1} \).

EXAMPLES:

Numerical evaluation is handled using mpmath:

\[
\begin{align*}
\text{sage: } & \text{N(exp_integral_e(1, 1))} & \# & \text{# needs sage.symbolic} \\
& 0.219383934395520 \\
\text{sage: } & \text{exp_integral_e(1, RealField(100)(1))} & \# & \text{# needs sage.symbolic} \\
& 0.21938393439552027367716377546
\end{align*}
\]

We can compare this to PARI's evaluation of \( \text{exponential_integral_1())} \):

\[
\begin{align*}
\text{sage: } & \text{N(exp_integral_1(1))} & \# & \text{# needs sage.symbolic} \\
& 0.219383934395520
\end{align*}
\]

We can verify one case of [AS1964] 5.1.45, i.e. \( E_n(z) = z^{n-1} \Gamma(1 - n, z) \):

\[
\begin{align*}
\text{sage: } & \text{N(exp_integral_e(2, 3+I))} & \# & \text{# needs sage.symbolic} \\
& 0.219383934395520
\end{align*}
\]
Functions, Release 10.4

0.00354575823814662 - 0.00973200528288687*I

```python
sage: N((3+I)*gamma(-1, 3+I))  # needs sage.symbolic
0.00354575823814662 - 0.00973200528288687*I
```

```python
>>> from sage.all import *
```

```python
>>> N(exp_integral_e(Integer(2), Integer(3)+I))  # needs sage.symbolic
0.00354575823814662 - 0.00973200528288687*I
```

```python
>>> N((Integer(3)+I)*gamma(-Integer(1), Integer(3)+I))  # needs sage.symbolic
0.00354575823814662 - 0.00973200528288687*I
```

Maxima returns the following improper integral as a multiple of \texttt{exp_integral_e}(1,1):

```python
sage: uu = integral(e^(-x)*log(x+1), x, 0, oo); uu  # needs sage.symbolic
\frac{e \cdot \text{exp}\_\text{integral}\_\text{e}(1, 1)}{}
```

```python
sage: uu.n(digits=30)  # needs sage.symbolic
0.596347362323194074341078499369
```

```python
>>> from sage.all import *
```

```python
>>> uu = integral(e**(-x)*log(x+Integer(1)), x, Integer(0), oo); uu  # needs sage.symbolic
\frac{e \cdot \text{exp}\_\text{integral}\_\text{e}(1, 1)}{}
```

```python
>>> uu.n(digits=Integer(30))  # needs sage.symbolic
0.596347362323194074341078499369
```

Symbolic derivatives and integrals are handled by Sage and Maxima:

```python
sage: # needs sage.symbolic
dx = var('x')
sage: f = exp_integral_e(Integer(2), x)
sage: f.diff(x)
-\text{exp}\_\text{integral}\_\text{e}(1, x)
```

```python
sage: f.integrate(x)
-\text{exp}\_\text{integral}\_\text{e}(3, x)
```

```python
sage: f = exp_integral_e(-Integer(1), x)
sage: f.integrate(x)
\text{Ei}(-x) - \gamma(-1, x)
```

Some special values of \texttt{exp\_integral\_e} can be simplified. [AS1964] 5.1.23:
Functions, Release 10.4

```python
sage: exp_integral_e(0, x)  # needs sage.symbolic
e^(-x)/x

>>> from sage.all import *
>>> exp_integral_e(Integer(0), x)  # needs sage.symbolic
e^(-x)/x
```

[AS1964] 5.1.24:

```python
sage: # needs sage.symbolic
sage: exp_integral_e(6, 0)
1/5

sage: nn = var('nn')
sage: assume(nn > 1)
sage: f = exp_integral_e(nn, 0)
sage: f.simplify()
1/(nn - 1)
```

ALGORITHM:
Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

class sage.functions.exp_integral.Function_exp_integral_e1

Bases: BuiltinFunction

The generalized complex exponential integral $E_1(z)$ defined by

$$E_1(z) = \int_z^\infty \frac{e^{-t}}{t} \, dt$$

see [AS1964] 5.1.4.

EXAMPLES:

```python
sage: exp_integral_e1(x)  # needs sage.symbolic
exp_integral_e1(x)
sage: exp_integral_e1(1.0)  # needs mpmath
0.219383934395520
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> exp_integral_e1(x)  # needs sage.symbolic
>>> exp_integral_e1(x)
```

(continues on next page)

1.15. Exponential integrals 221
>> exp_integral_e1(RealNumber('1.0'))
˓→ # needs mpmath
0.219383934395520

**Numerical evaluation is handled using mpmath:**

```python
sage: N(exp_integral_e1(1))  # needs sage.symbolic
0.219383934395520
sage: exp_integral_e1(RealField(100)(1))  # needs sage.rings.real_mpfr
0.21938393439552027367716377546
```

We can compare this to PARI's evaluation of `exponential_integral_1()`:

```python
sage: N(exp_integral_e1(2.0))  # needs mpmath
0.0489005107080611
sage: N(exponential_integral_1(2.0))  # needs sage.rings.real_mpfr
0.0489005107080611
```

**Symbolic derivatives and integrals are handled by Sage and Maxima:**

```python
sage: # needs sage.symbolic
type(x)
'sage.symbolic.expression.Expression'
sage: f = exp_integral_e1(x)
sage: f.diff(x)
-e^(-x)/x
sage: f.integrate(x)
-exp_integral_e(2, x)
```

```python
from sage.all import *

sage: # needs sage.symbolic
type(x)
'sage.symbolic.expression.Expression'
sage: f = exp_integral_e1(x)
sage: f.diff(x)
-e^(-x)/x
sage: f.integrate(x)
-exp_integral_e(2, x)
```
ALGORITHM:

Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

```
class sage.functions.exp_integral.Function_log_integral
Bases: BuiltinFunction

The logarithmic integral \( \text{li}(z) \) defined by

\[
\text{li}(x) = \int_0^x \frac{dt}{\ln(t)} = \text{Ei}(\ln(x))
\]

for \( x > 1 \) and by analytic continuation for complex arguments \( z \) (see [AS1964] 5.1.3).

EXAMPLES:

Numerical evaluation for real and complex arguments is handled using mpmath:

```
sage: N(log_integral(3))
# needs sage.symbolic
2.16358859466719
sage: N(log_integral(3), digits=30)
# needs sage.symbolic
2.16358859466719197287692236735
sage: log_integral(ComplexField(100)(3+I))
# needs sage.symbolic
2.2879892769816826157078450911 + 0.87232935488528370139883806779*I
sage: log_integral(0)
# needs mpmath
0
```

Symbolic derivatives and integrals are handled by Sage and Maxima:

```
sage: x = var('x')
sage: f = log_integral(x)
sage: f.diff(x)
x/log(x)
sage: f.integrate(x)
x*log_integral(x) - Ei(2*log(x))
```

(continues on next page)
Here is a test from the mpmath documentation. There are 1,925,320,391,606,803,968,923 many prime numbers less than $1e23$. The value of $\log_integral(1e23)$ is very close to this:

```
sage: log_integral(1e23)  # needs mpmath
1.92532039161405e21
```

```
>>> from sage.all import *
>>>
```

```
log_integral(RealNumber('1e23')) # needs mpmath
1.92532039161405e21
```

**ALGORITHM:**

Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

**REFERENCES:**
- Wikipedia article Logarithmic_integral_function
- mpmath documentation: logarithmic-integral

```
class sage.functions.exp_integral.Function_log_integral_offset
```

**ALGORITHM:**

The offset logarithmic integral, or Eulerian logarithmic integral, $\text{Li}(x)$ is defined by

$$
\text{Li}(x) = \int_2^x \frac{dt}{\ln(t)} = \text{li}(x) - \text{li}(2)
$$

for $x \geq 2$.

The offset logarithmic integral should also not be confused with the polylogarithm (also denoted by $\text{Li}(x)$ ), which is implemented as `sage.functions.log.Function_polylog`.

$\text{Li}(x)$ is identical to $\text{li}(x)$ except that the lower limit of integration is 2 rather than 0 to avoid the singularity at $x = 1$ of

$$
\frac{1}{\ln(t)}
$$

See `Function_log_integral` for details of $\text{li}(x)$. Thus $\text{Li}(x)$ can also be represented by

$$
\text{Li}(x) = \text{li}(x) - \text{li}(2)
$$

So we have:

```
sage: li(4.5) - li(2.0) - Li(4.5)  # needs mpmath
0.000000000000000
```

```
>>> from sage.all import *
>>>
```

```
li(RealNumber('4.5')) - li(RealNumber('2.0')) - Li(RealNumber('4.5')) # needs mpmath
0.000000000000000
```
Li(x) is extended to complex arguments z by analytic continuation (see [AS1964] 5.1.3):

```sage
sage: Li(6.6 + 5.4*I) # needs sage.symbolic
3.97032201503632 + 2.62311237593572*I
```

The function Li is an approximation for the number of primes up to x. In fact, the famous Riemann Hypothesis is

\[ |\pi(x) - Li(x)| \leq \sqrt{x} \log(x). \]

For “small” x, Li(x) is always slightly bigger than \( \pi(x) \). However it is a theorem that there are very large values of x (e.g., around \( 10^{316} \)), such that \( \exists x : \pi(x) > Li(x) \). See “A new bound for the smallest x with \( \pi(x) > li(x) \)”, Bays and Hudson, Mathematics of Computation, 69 (2000) 1285-1296.

**Note:** Definite integration returns a part symbolic and part numerical result. This is because when Li(x) is evaluated it is passed as li(x)-li(2).

**EXAMPLES:**

Numerical evaluation for real and complex arguments is handled using mpmath:

```sage
sage: # needs sage.symbolic
sage: N(log_integral_offset(3))
1.11842481454970
sage: N(log_integral_offset(3), digits=30)
1.11842481454969918803233347815
sage: log_integral_offset(ComplexField(100)(3+I))
1.2428254968641898308632562019 + 0.87232935488528370139883806779*I
sage: log_integral_offset(2)
0
```

```sage
sage: for n in range(1,7):
    print('n=10^{n}, pi(pi(10^n)), N(LogIntegral(10^n)))')
10 4 5.12043572466980
100 25 29.0809778039621
1000 168 176.564494210035
10000 1229 1245.09205211927
100000 9592 9628.76383727068
1000000 78498 78626.5039956821
```

(continues on next page)
Here is a test from the mpmath documentation. There are 1,925,320,391,606,803,968,923 prime numbers less than 1e23. The value of log_integral_offset(1e23) is very close to this:

```
sage: log_integral_offset(1e23) # needs mpmath
1.92532039161405e21
```

Symbolic derivatives are handled by Sage and integration by Maxima:

```
sage: # needs sage.symbolic
sage: x = var('x')
sage: f = log_integral_offset(x)
sage: f.diff(x)
1/log(x)
sage: f.integrate(x)
-x*log_integral(2) + x*log_integral(x) - Ei(2*log(x))
sage: N(f.integrate(x, 2.0, 4.5).n(digits=10)) # abs tol 1e-15
0.601621785860587
```

ALGORITHM:
Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

REFERENCES:
- Wikipedia article Logarithmic_integral_function
- mpmath documentation: logarithmic-integral
class sage.functions.exp_integral.Function_sin_integral

Bases: BuiltinFunction

The trigonometric integral \( \text{Si}(z) \) defined by

\[
\text{Si}(z) = \int_0^z \frac{\sin(t)}{t} \, dt,
\]

see [AS1964] 5.2.1.

EXAMPLES:

Numerical evaluation for real and complex arguments is handled using mpmath:

```python
sage: sin_integral(0)  # needs mpmath
0
sage: sin_integral(0.0)  # needs mpmath
0.000000000000000
sage: sin_integral(3.0)  # needs mpmath
1.84865252799947
sage: N(sin_integral(3), digits=30)  # needs sage.symbolic
1.84865252799946825639773025111
sage: sin_integral(ComplexField(100)(3+I))  # needs sage.symbolic
2.0277151656451253616038525998 + 0.015210926166954211913653130271*I
```

The alias \( \text{Si} \) can be used instead of \( \text{sin_integral} \):

```python
>>> from sage.all import *

>>> sin_integral(Integer(0))  # needs mpmath
0
>>> sin_integral(RealNumber('0.0'))  # needs mpmath
0.000000000000000
>>> sin_integral(RealNumber('3.0'))  # needs mpmath
1.84865252799947
>>> N(sin_integral(Integer(3)), digits=Integer(30))  # needs sage.symbolic
1.84865252799946825639773025111
>>> sin_integral(ComplexField(Integer(100))(Integer(3)+I))  # needs sage.symbolic
2.0277151656451253616038525998 + 0.015210926166954211913653130271*I
```

The limit of \( \text{Si}(z) \) as \( z \to \infty \) is \( \pi/2 \):
The exponential sine integral is analytic everywhere:

```
sage: sin_integral(-1.0)  # needs mpmath
-0.946083070367183
sage: sin_integral(-2.0)  # needs mpmath
-1.60541297680269
sage: sin_integral(-1e23)  # needs mpmath
-1.57079632679490

>>> from sage.all import *

>>> sin_integral(-RealNumber('1.0'))  # needs mpmath
-0.946083070367183
>>> sin_integral(-RealNumber('2.0'))  # needs mpmath
-1.60541297680269
>>> sin_integral(-RealNumber('-1e23'))  # needs mpmath
-1.57079632679490
```

Symbolic derivatives and integrals are handled by Sage and Maxima:
Compare values of the functions $\text{Si}(x)$ and $f(x) = (1/2)i \cdot \text{Ei}(-ix) - (1/2)i \cdot \text{Ei}(ix) - \pi/2$, which are both anti-derivatives of $\sin(x)/x$, at some random positive real numbers:

```
sage: f(x) = 1/2*I*Ei(-I*x) - 1/2*I*Ei(I*x) - pi/2  # needs sage.symbolic
sage: g(x) = sin_integral(x)  # needs sage.symbolic
sage: R = [abs(RDF.random_element()) for i in range(100)]
sage: all(abs(f(x) - g(x)) < 1e-10 for x in R)  # needs sage.symbolic
True
```

The Nielsen spiral is the parametric plot of $(\text{Si}(t), \text{Ci}(t))$:

```
sage: x = var('x')
sage: f(x) = sin_integral(x)
sage: g(x) = cos_integral(x)
sage: P = parametric_plot([f, g], (x, 0.5 ,20))  # needs sage.plot
sage: show(P, frame=True, axes=False)  # needs sage.plot
```

1.15. Exponential integrals 229
>>> from sage.all import *
>>> # needs sage.symbolic
>>> x = var('x')
>>> _tmp__=var("x"); f = symbolic_expression(sin_integral(x)).function(x)
>>> _tmp__=var("x"); g = symbolic_expression(cos_integral(x)).function(x)
>>> P = parametric_plot([f, g], (x, RealNumber('0.5'), Integer(20))) # needs sage.plot
>>> show(P, frame=True, axes=False) # needs sage.plot

ALGORITHM:
Numerical evaluation is handled using mpmath, but symbolic handling is performed by Sage and Maxima.

REFERENCES:
- Wikipedia article Trigonometric_integral
- mpmath documentation: si

class sage.functions.exp_integral.Function_sinh_integral

Bases: BuiltinFunction

The trigonometric integral Shi(z) defined by

\[ \text{Shi}(z) = \int_0^z \frac{\sinh(t)}{t} \, dt, \]

see [AS1964] 5.2.3.

EXAMPLES:

Numerical evaluation for real and complex arguments is handled using mpmath:

```sage
c: sinh_integral(3.0)  # needs mpmath
4.97344047585981
c: sinh_integral(1.0)  # needs mpmath
1.05725087537573
c: sinh_integral(-1.0) # needs mpmath
-1.05725087537573
```

```sage
>>> from sage.all import *
>>> # needs mpmath
>>> sinh_integral(RealNumber('3.0'))  # needs mpmath
4.97344047585981
>>> sinh_integral(RealNumber('1.0'))  # needs mpmath
1.05725087537573
>>> sinh_integral(-RealNumber('1.0')) # needs mpmath
-1.05725087537573
```

The alias Shi can be used instead of sinh_integral:

```sage
Shi(3.0)  # needs mpmath
4.97344047585981
```
>>> from sage.all import *
>>> Shi(RealNumber('3.0'))  # needs mpmath
4.97344047585981

Compare \( \sinh \text{Integral}(3.0) \) to the definition of the value using numerical integration:

```python
sage: a = numerical_integral(sinh(x)/x, 0, 3)[0]  # needs sage.symbolic
sage: abs(a - N(sinh_integral(3))) < 1e-14  # needs sage.symbolic
True
```

Arbitrary precision and complex arguments are handled:

```python
sage: N(sinh_integral(3), digits=30)  # needs sage.symbolic
4.97344047585980679771041838252
sage: sinh_integral(ComplexField(100)(3+I))  # needs sage.symbolic
3.9134623660329374406788354078 + 3.0427678212908839256360163759*I
```

The limit \( \text{Shi}(z) \) as \( z \to \infty \) is \( \infty \):

```python
sage: N(sinh_integral(Infinity))  # needs mpmath
+infinity
```

Symbolic derivatives and integrals are handled by Sage and Maxima:

```python
sage: x = var('x')  # needs sage.symbolic
sage: f = sinh_integral(x)  # needs sage.symbolic
sage: f.diff(x)  # needs sage.symbolic
```

(continues on next page)
\[
\frac{\sinh(x)}{x}
\]

```
sage: f.integrate(x)  # needs sage.symbolic
x*sinh_integral(x) - \cosh(x)
```

```
>>> from sage.all import *

>>> x = var('x')  # needs sage.symbolic

>>> f = sinh_integral(x)  # needs sage.symbolic

>>> f.diff(x)  # needs sage.symbolic
\frac{\sinh(x)}{x}

>>> f.integrate(x)  # needs sage.symbolic
x*sinh_integral(x) - \cosh(x)
```

Note that due to some problems with the way Maxima handles these expressions, definite integrals can sometimes give unexpected results (typically when using inexact endpoints) due to inconsistent branching:

```
sage: integrate(sinh_integral(x), x, 0, 1/2)  # needs sage.symbolic
-cosh(1/2) + \frac{1}{2}\sinh_integral(1/2) + 1

sage: integrate(sinh_integral(x), x, 0, 1/2).n()  # correct  # needs sage.symbolic
0.125872409703453

sage: integrate(sinh_integral(x), x, 0, 0.5).n()  # fixed in maxima 5.29.1  # needs sage.symbolic
0.125872409703453

>>> from sage.all import *

>>> integrate(sinh_integral(x), x, Integer(0), Integer(1)/Integer(2))  # needs sage.symbolic
-cosh(1/2) + \frac{1}{2}\sinh_integral(1/2) + 1

>>> integrate(sinh_integral(x), x, Integer(0), Integer(1)/Integer(2)).n()  # needs sage.symbolic
0.125872409703453

>>> integrate(sinh_integral(x), x, Integer(0), RealNumber('0.5')).n()  # fixed in maxima 5.29.1  # needs sage.symbolic
0.125872409703453
```

**ALGORITHM:**

Numerical evaluation is handled using mpmath, but symbolics are handled by Sage and Maxima.

**REFERENCES:**

- Wikipedia article Trigonometric_integral
- mpmath documentation: shi
- sage.functions.exp_integral.exponential_integral_1(x, n=0)

Returns the exponential integral \( E_1(x) \). If the optional argument \( n \) is given, computes list of the first \( n \) values of the exponential integral \( E_1(xm) \).
The exponential integral $E_1(x)$ is

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t} \, dt$$

**INPUT:**

- $x$ – a positive real number
- $n$ – (default: 0) a nonnegative integer; if nonzero, then return a list of values $E_1(x^m)$ for $m = 1, 2, 3, \ldots, n$.
  This is useful, e.g., when computing derivatives of L-functions.

**OUTPUT:**

A real number if $n$ is 0 (the default) or a list of reals if $n > 0$. The precision is the same as the input, with a default of 53 bits in case the input is exact.

**EXAMPLES:**

```sage
sage: # needs sage.libs.pari sage.rings.real_mpfr
sage: exponential_integral_1(2)
0.0489005107080611
sage: exponential_integral_1(2, 4)  # abs tol 1e-18
[0.0489005107080611, 0.00377935240984891, 0.00036008245216259, 0.
˓→0000376656228439245]

sage: exponential_integral_1(40, 5)
[0.00000000000000, 2.2285432586847e-37, 6.33732515501151e-55,
  2.02336191509997e-72, 6.88522610630764e-90]

sage: r = exponential_integral_1(RealField(150)(1)); r
0.21938393439552027367716377546012164903104729

sage: parent(r)
Real Field with 150 bits of precision

sage: exponential_integral_1(RealField(150)(100))
3.6835977616820321802351926205081189876552201e-46

sage: exponential_integral_1(0)
+Infinity

>>> from sage.all import *

>>> # needs sage.libs.pari sage.rings.real_mpfr

>>> exponential_integral_1(Integer(2))
0.0489005107080611

>>> exponential_integral_1(Integer(2), Integer(4))  # abs tol 1e-18
[0.0489005107080611, 0.00377935240984891, 0.00036008245216259, 0.
˓→0000376656228439245]

>>> exponential_integral_1(Integer(40), Integer(5))
[0.00000000000000, 2.2285432586847e-37, 6.33732515501151e-55,
  2.02336191509997e-72, 6.88522610630764e-90]

>>> r = exponential_integral_1(RealField(Integer(150))(1)); r
0.21938393439552027367716377546012164903104729

>>> parent(r)
Real Field with 150 bits of precision

>>> exponential_integral_1(RealField(Integer(150))(100))
3.6835977616820321802351926205081189876552201e-46

>>> exponential_integral_1(Integer(0))
+Infinity

>>> from sage.all import *

>>> # needs sage.libs.pari sage.rings.real_mpfr

>>> exponential_integral_1(Integer(2))
0.0489005107080611

>>> exponential_integral_1(Integer(2), Integer(4))  # abs tol 1e-18
[0.0489005107080611, 0.00377935240984891, 0.00036008245216259, 0.
˓→0000376656228439245]

>>> exponential_integral_1(Integer(40), Integer(5))
[0.00000000000000, 2.2285432586847e-37, 6.33732515501151e-55,
  2.02336191509997e-72, 6.88522610630764e-90]

>>> r = exponential_integral_1(RealField(Integer(150))(1)); r
0.21938393439552027367716377546012164903104729

>>> parent(r)
Real Field with 150 bits of precision

>>> exponential_integral_1(RealField(Integer(150))(100))
3.6835977616820321802351926205081189876552201e-46

>>> exponential_integral_1(Integer(0))
+Infinity

ALGORITHM: use the PARI C-library function pari:eint1.

REFERENCE:
• See Proposition 5.6.12 of Cohen’s book “A Course in Computational Algebraic Number Theory”.

1.16 Wigner, Clebsch-Gordan, Racah, and Gaunt coefficients

Collection of functions for calculating Wigner 3-j, 6-j, 9-j, Clebsch-Gordan, Racah as well as Gaunt coefficients exactly, all evaluating to a rational number times the square root of a rational number [RH2003].

Please see the description of the individual functions for further details and examples.

AUTHORS:
• Jens Rasch (2009-03-24): initial version for Sage
• Jens Rasch (2009-05-31): updated to sage-4.0

sage.functions.wigner.clebsch_gordan(j_1, j_2, j_3, m_1, m_2, m_3, prec=None)

Return the Clebsch-Gordan coefficient \( \langle j_1 m_1 | j_2 m_2 | j_3 m_3 \rangle \).

The reference for this function is [Ed1974].

INPUT:
• \( j_1, j_2, j_3, m_1, m_2, m_3 \) – integer or half integer
• \( \text{prec} \) – precision, default: None. Providing a precision can drastically speed up the calculation.

OUTPUT:
Rational number times the square root of a rational number (if \( \text{prec}=\text{None} \)), or real number if a precision is given.

EXAMPLES:

```python
sage: simplify(clebsch_gordan(3/2,1/2,2, 3/2,1/2,2))  # needs sage.symbolic
1
sage: clebsch_gordan(1.5,0.5,1, 1.5,-0.5,1)  # needs sage.symbolic
1/2*sqrt(3)
sage: clebsch_gordan(3/2,1/2,1, -1/2,1/2,0)  # needs sage.symbolic
-sqrt(3)*sqrt(1/6)
```

```python
>>> from sage.all import *
>>> simplify(clebsch_gordan(Integer(3)/Integer(2),Integer(1)/Integer(2),Integer(1)/Integer(2),Integer(1)/Integer(2),Integer(1)/Integer(2),Integer(1)/Integer(2)))  # needs sage.symbolic
1
>>> clebsch_gordan(RealNumber('1.5'),RealNumber('0.5'),Integer(1), RealNumber('1.5 ->'),-RealNumber('0.5'),Integer(1))  # needs sage.symbolic
1/2*sqrt(3)
>>> clebsch_gordan(Integer(3)/Integer(2),Integer(1)/Integer(2),Integer(1), -Integer(1)/Integer(2),Integer(1)/Integer(2),Integer(0))  # needs sage.symbolic
-sqrt(3)*sqrt(1/6)
```
Note: The Clebsch-Gordan coefficient will be evaluated via its relation to Wigner 3-\(j\) symbols:

\[
\langle j_1 m_1 \, j_2 m_2 | j_3 m_3 \rangle = (-1)^{j_1 - j_2 + m_3} \sqrt{2j_3 + 1} \begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & -m_3 \end{pmatrix}
\]

See also the documentation on Wigner 3-\(j\) symbols which exhibit much higher symmetry relations than the Clebsch-Gordan coefficient.

AUTHORS:

• Jens Rasch (2009-03-24): initial version

\texttt{sage.functions.wigner.gaunt(l_1,l_2,l_3,m_1,m_2,m_3,prec=None)}

Return the Gaunt coefficient.

The Gaunt coefficient is defined as the integral over three spherical harmonics:

\[
Y(l_1,l_2,l_3,m_1,m_2,m_3) = \int Y_{l_1,m_1}(\Omega) \, Y_{l_2,m_2}(\Omega) \, Y_{l_3,m_3}(\Omega) \, d\Omega
\]

\[
= \sqrt{\frac{(2l_1+1)(2l_2+1)(2l_3+1)}{4\pi}} \times \begin{pmatrix} l_1 & l_2 & l_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} m_1 & m_2 & m_3 \end{pmatrix}
\]

INPUT:

• \(l_1, l_2, l_3, m_1, m_2, m_3\) – integer

• \(\text{prec}\) – precision, default: None. Providing a precision can drastically speed up the calculation.

OUTPUT:

Rational number times the square root of a rational number (if \(\text{prec}=\text{None}\)), or real number if a precision is given.

EXAMPLES:

\[
\begin{align*}
\text{sage: } & \# \text{ needs sage.symbolic} \\
\text{sage: } & \text{gaunt}(1,0,1,0,-1) \\
& -1/2/sqrt(pi) \\
\text{sage: } & \text{gaunt}(1,0,1,0,0) \\
& 0 \\
\text{sage: } & \text{gaunt}(29,29,34,10,-5,-5) \\
& 1821867940156/215552371055153321*sqrt(22134)/sqrt(pi) \\
\text{sage: } & \text{gaunt}(20,20,40,1,-1,0) \\
& 28384503878959800/74029560764440771/sqrt(pi) \\
\text{sage: } & \text{gaunt}(12,15,5,2,3,-5) \\
& 91/124062*sqrt(36890)/sqrt(pi) \\
\text{sage: } & \text{gaunt}(10,10,12,9,3,-12) \\
& -98/62031*sqrt(6279)/sqrt(pi) \\
\text{sage: } & \text{gaunt}(1000,1000,1200,9,3,-12).n(64) \\
& 0.0068950042192211348 \\
\end{align*}
\]

>>> from sage.all import *

>>> \# needs sage.symbolic

>>> gaunt(Integer(1),Integer(0),Integer(1),Integer(1),Integer(1),Integer(0),-Integer(1))

(continues on next page)
Functions, Release 10.4

-\frac{1}{2}/\sqrt{\pi}

>>> \text{gaunt}(\text{Integer}(1), \text{Integer}(0), \text{Integer}(1), \text{Integer}(1), \text{Integer}(0), \text{Integer}(0))
0

>>> \text{gaunt}(\text{Integer}(29), \text{Integer}(29), \text{Integer}(34), \text{Integer}(10), -\text{Integer}(5), -\text{Integer}(5))
1821867940156/215552371055153321*\sqrt{22134}/\sqrt{\pi}

>>> \text{gaunt}(\text{Integer}(20), \text{Integer}(20), \text{Integer}(40), \text{Integer}(1), -\text{Integer}(1), \text{Integer}(0))
28384503878959800/74029560764440771/\sqrt{\pi}

>>> \text{gaunt}(\text{Integer}(12), \text{Integer}(15), \text{Integer}(5), \text{Integer}(2), \text{Integer}(3), -\text{Integer}(5))
91/124062*sqrt(36890)/sqrt(\pi)

>>> \text{gaunt}(\text{Integer}(20), \text{Integer}(20), \text{Integer}(40), \text{Integer}(1), -\text{Integer}(1), \text{Integer}(0))
28384503878959800/74029560764440771/\sqrt{\pi}

If the sum of the \(l_i\) is odd, the answer is zero, even for Python ints (see Issue #14766):

```python
sage: \text{gaunt}(1, 2, 2, 1, 0, -1)
0
sage: \text{gaunt}(\text{int}(1), \text{int}(2), \text{int}(2), 1, 0, -1)
0
```

```python
>>> \text{from sage.all import }*

sage: \text{gaunt}(\text{Integer}(1), \text{Integer}(2), \text{Integer}(2), \text{Integer}(1), \text{Integer}(0), -\text{Integer}(1))
0
```

It is an error to use non-integer values for \(l\) or \(m\):

```python
sage: \text{gaunt}(1.2, 0, 1.2, 0, 0, 0)
# needs sage.rings.real_mpfr
Traceback (most recent call last):
... TypeError: Attempt to coerce non-integral RealNumber to Integer
```

```python
sage: \text{gaunt}(1, 0, 1.1, 0, -1.1)
# needs sage.rings.real_mpfr
Traceback (most recent call last):
... TypeError: Attempt to coerce non-integral RealNumber to Integer
```

```python
>>> \text{from sage.all import }*

sage: \text{gaunt}(\text{RealNumber}'1.2'), \text{Integer}(0), \text{RealNumber}'1.2', \text{Integer}(0), \text{Integer}(0), \text{RealNumber}'1.1'
# needs sage.rings.real_mpfr
Traceback (most recent call last):
... TypeError: Attempt to coerce non-integral RealNumber to Integer
```

```python
>>> \text{gaunt}(\text{Integer}(1), \text{Integer}(0), \text{RealNumber}'1.1')
# needs sage.rings.real_mpfr
Traceback (most recent call last):
... TypeError: Attempt to coerce non-integral RealNumber to Integer
```

ALGORITHM:
This function uses the algorithm of [LdB1982] to calculate the value of the Gaunt coefficient exactly. Note that the formula contains alternating sums over large factorials and is therefore unsuitable for finite precision arithmetic and only useful for a computer algebra system [RH2003].

AUTHORS:


sage.functions.wigner.racah (aa, bb, cc, dd, ee, ff, prec=None)

Return the Racah symbol $W(aa, bb, cc, dd; ee, ff)$.

INPUT:

- $aa, ..., ff$ – integer or half integer
- $prec$ – precision, default: None. Providing a precision can drastically speed up the calculation.

OUTPUT:

Rational number times the square root of a rational number (if $prec=None$), or real number if a precision is given.

EXAMPLES:

```python
sage: racah(3,3,3,3,3,3)  # needs sage.symbolic
-1/14
```

```python
>>> from sage.all import *

Note: The Racah symbol is related to the Wigner 6-$j$ symbol:

$$\begin{pmatrix} j_1 & j_2 & j_3 \\ j_4 & j_5 & j_6 \end{pmatrix} = (-1)^{j_1+j_2+j_4+j_6} W(j_1, j_2, j_4; j_3, j_6)$$

Please see the 6-$j$ symbol for its much richer symmetries and for additional properties.

ALGORITHM:

This function uses the algorithm of [Ed1974] to calculate the value of the 6-$j$ symbol exactly. Note that the formula contains alternating sums over large factorials and is therefore unsuitable for finite precision arithmetic and only useful for a computer algebra system [RH2003].

AUTHORS:

- Jens Rasch (2009-03-24): initial version

sage.functions.wigner.wigner_3j (j_1, j_2, j_3, m_1, m_2, m_3, prec=None)

Return the Wigner 3-$j$ symbol $\begin{pmatrix} j_1 & j_2 & j_3 \\ m_1 & m_2 & m_3 \end{pmatrix}$.

INPUT:

- $j_1, j_2, j_3, m_1, m_2, m_3$ – integer or half integer
- $prec$ – precision, default: None. Providing a precision can drastically speed up the calculation.
OUTPUT:

Rational number times the square root of a rational number (if prec=None), or real number if a precision is given.

EXAMPLES:

```
sage: wigner_3j(2, 6, 4, 0, 0, 0)  # needs sage.symbolic
    sqrt(5/143)
sage: wigner_3j(2, 6, 4, 0, 0, 1)
    0
sage: wigner_3j(0.5, 0.5, 1, 0.5, -0.5, 0)  # needs sage.symbolic
    sqrt(1/6)
sage: wigner_3j(40, 100, 60, -10, 60, -50)  # needs sage.symbolic
    95608/18702538494885*sqrt(210827358367353143364163310/220491455010479533763)
sage: wigner_3j(2500, 2500, 5000, 2488, 2400, -4888, prec=64)  # needs sage.rings.real_mpfr
    7.60424456883448589e-12
```

It is an error to have arguments that are not integer or half integer values:

```
sage: wigner_3j(2.1, 6, 4, 0, 0, 0)
ValueError: j values must be integer or half integer
sage: wigner_3j(2, 6, 4, 1, 0, -1.1)
ValueError: m values must be integer or half integer
```

(continues on next page)
The Wigner 3-\(j\) symbol obeys the following symmetry rules:

- invariant under any permutation of the columns (with the exception of a sign change where \(J = j_1 + j_2 + j_3\)):

\[
\begin{pmatrix}
  j_1 & j_2 & j_3 \\
  m_1 & m_2 & m_3
\end{pmatrix}
= \begin{pmatrix}
  j_3 & j_1 & j_2 \\
  m_3 & m_1 & m_2
\end{pmatrix} = \begin{pmatrix}
  j_2 & j_3 & j_1 \\
  m_2 & m_3 & m_1
\end{pmatrix}
= (-1)^J \begin{pmatrix}
  j_3 & j_2 & j_1 \\
  m_3 & m_2 & m_1
\end{pmatrix} = (-1)^J \begin{pmatrix}
  j_1 & j_3 & j_2 \\
  m_1 & m_3 & m_2
\end{pmatrix} = (-1)^J \begin{pmatrix}
  j_2 & j_1 & j_3 \\
  m_2 & m_1 & m_3
\end{pmatrix}
\]

- invariant under space inflection, i.e.

\[
\begin{pmatrix}
  j_1 & j_2 & j_3 \\
  m_1 & m_2 & m_3
\end{pmatrix}
= (-1)^J \begin{pmatrix}
  j_1 & j_2 & j_3 \\
  -m_1 & -m_2 & -m_3
\end{pmatrix}
\]

- symmetric with respect to the 72 additional symmetries based on the work by [Reg1958]
- zero for \(j_1, j_2, j_3\) not fulfilling triangle relation
- zero for \(m_1 + m_2 + m_3 \neq 0\)
- zero for violating any one of the conditions \(j_1 \geq |m_1|, j_2 \geq |m_2|, j_3 \geq |m_3|\)

**ALGORITHM:**

This function uses the algorithm of [Ed1974] to calculate the value of the 3-\(j\) symbol exactly. Note that the formula contains alternating sums over large factorials and is therefore unsuitable for finite precision arithmetic and only useful for a computer algebra system [RH2003].

**AUTHORS:**

- Jens Rasch (2009-03-24): initial version

```python
sage.functions.wigner.wigner_6j(j_1, j_2, j_3, j_4, j_5, j_6, prec=None)
```

Return the Wigner 6-\(j\) symbol \(\{j_1, j_2, j_3, j_4, j_5, j_6\}\).

**INPUT:**

- \(j_1, \ldots, j_6\) – integer or half integer
- \(\text{prec}\) – precision, default: None. Providing a precision can drastically speed up the calculation.

**OUTPUT:**

Rational number times the square root of a rational number (if \(\text{prec}=\text{None}\)), or real number if a precision is given.

**EXAMPLES:**

```python
sage: # needs sage.symbolic
sage: wigner_6j(3,3,3,3,3,3)
-1/14
sage: wigner_6j(5,5,5,5,5,5)
1/52
```

(continues on next page)
Functions, Release 10.4

(continued from previous page)

```
sage: wigner_6j(6,6,6,6,6,6)
309/10868
sage: wigner_6j(8,8,8,8,8,8)
-12219/965770
sage: wigner_6j(30,30,30,30,30,30)
36082186869033479581/87954851694828981714124
sage: wigner_6j(0.5,0.5,0.5,0.5,0.5,0.5)
1/6
sage: wigner_6j(200,200,200,200,200,200, prec=1000)*1.0
0.000155903212413242

>>> from sage.all import *
>>> # needs sage.symbolic
>>> wigner_6j(Integer(3),Integer(3),Integer(3),Integer(3),Integer(3),Integer(3))
-1/14
>>> wigner_6j(Integer(5),Integer(5),Integer(5),Integer(5),Integer(5),Integer(5))
1/52
>>> wigner_6j(Integer(6),Integer(6),Integer(6),Integer(6),Integer(6),Integer(6))
309/10868
>>> wigner_6j(Integer(8),Integer(8),Integer(8),Integer(8),Integer(8),Integer(8))
-12219/965770
>>> wigner_6j(Integer(30),Integer(30),Integer(30),Integer(30),Integer(30),Integer(30),
Integer(30))
36082186869033479581/87954851694828981714124
>>> wigner_6j(RealNumber('0.5'),RealNumber('0.5'),Integer(1),RealNumber('0.5'),
RealNumber('0.5'),Integer(1))
1/6
>>> wigner_6j(Integer(200),Integer(200),Integer(200),Integer(200),Integer(200),
Integer(200), prec=Integer(1000))*RealNumber('1.0')
0.000155903212413242

It is an error to have arguments that are not integer or half integer values or do not fulfill the triangle relation:

```
sage: wigner_6j(2.5,2.5,2.5,2.5,2.5,2.5)
Traceback (most recent call last):
... ValueError: j values must be integer or half integer and fulfill the triangle relation
sage: wigner_6j(0.5,0.5,1.1,0.5,0.5,1.1)
Traceback (most recent call last):
... ValueError: j values must be integer or half integer and fulfill the triangle relation

>>> from sage.all import *
>>> wigner_6j(RealNumber('2.5'),RealNumber('2.5'),RealNumber('2.5'),RealNumber('2.5'),
RealNumber('2.5'),RealNumber('2.5'))
Traceback (most recent call last):
... ValueError: j values must be integer or half integer and fulfill the triangle relation
>>> wigner_6j(RealNumber('0.5'),RealNumber('0.5'),RealNumber('1.1'),RealNumber('0.5'),
RealNumber('0.5'),RealNumber('1.1'))
Traceback (most recent call last):
... ValueError: j values must be integer or half integer and fulfill the triangle relation
```
The Wigner 6-$j$ symbol is related to the Racah symbol but exhibits more symmetries as detailed below.

\[
\begin{pmatrix}
  j_1 & j_2 & j_3 \\
  j_4 & j_5 & j_6 \\
\end{pmatrix} = (-1)^{j_1+j_2+j_4+j_5} W(j_1, j_2, j_4; j_3, j_5, j_6)
\]

The Wigner 6-$j$ symbol obeys the following symmetry rules:

1. Wigner 6-$j$ symbols are left invariant under any permutation of the columns:

\[
\begin{pmatrix}
  j_1 & j_2 & j_3 \\
  j_4 & j_5 & j_6 \\
\end{pmatrix} = \begin{pmatrix}
  j_3 & j_1 & j_2 \\
  j_6 & j_4 & j_5 \\
\end{pmatrix} = \begin{pmatrix}
  j_1 & j_3 & j_2 \\
  j_4 & j_6 & j_5 \\
\end{pmatrix} = \begin{pmatrix}
  j_2 & j_1 & j_3 \\
  j_5 & j_4 & j_6 \\
\end{pmatrix}
\]

2. They are invariant under the exchange of the upper and lower arguments in each of any two columns, i.e.

\[
\begin{pmatrix}
  j_1 & j_2 & j_3 \\
  j_4 & j_5 & j_6 \\
\end{pmatrix} = \begin{pmatrix}
  j_1 & j_5 & j_6 \\
  j_4 & j_3 & j_2 \\
\end{pmatrix} = \begin{pmatrix}
  j_4 & j_2 & j_3 \\
  j_1 & j_5 & j_6 \\
\end{pmatrix} = \begin{pmatrix}
  j_4 & j_5 & j_3 \\
  j_1 & j_2 & j_6 \\
\end{pmatrix}
\]

3. additional 6 symmetries [Reg1959] giving rise to 144 symmetries in total

4. only non-zero if any triple of $j$'s fulfill a triangle relation

**ALGORITHM:**

This function uses the algorithm of [Ed1974] to calculate the value of the 6-$j$ symbol exactly. Note that the formula contains alternating sums over large factorials and is therefore unsuitable for finite precision arithmetic and only useful for a computer algebra system [RH2003].

```python
sage.functions.wigner.wigner_9j(j_1, j_2, j_3, j_4, j_5, j_6, prec=None)
```

Return the Wigner 9-$j$ symbol \( \begin{pmatrix}
  j_1 & j_2 & j_3 \\
  j_4 & j_5 & j_6 \\
\end{pmatrix} \).

**INPUT:**

- \( j_1, \ldots, j_9 \) – integer or half integer
- \( \text{prec} \) – precision, default: None. Providing a precision can drastically speed up the calculation.

**OUTPUT:**

Rational number times the square root of a rational number (if \( \text{prec} = \text{None} \)), or real number if a precision is given.

**EXAMPLES:**

A couple of examples and test cases, note that for speed reasons a precision is given:

```python
sage: # needs sage.symbolic
sage: wigner_9j(1, 1, 1, 1, 1, 1, 1, 1, 0, prec=64) # ==1/18
0.05555555555555555
sage: wigner_9j(1, 1, 1, 1, 1, 1, 1, 1, 1)
0
sage: wigner_9j(1, 1, 1, 1, 1, 1, 1, 1, 2, prec=64) # ==1/18
0.05555555555555555
sage: wigner_9j(1, 1, 1, 1, 1, 1, 1, 1, 2, prec=64) # ==-1/150
```

(continues on next page)
>>> from sage.all import *
>>> # needs sage.symbolic
>>> wigner_9j(Integer(1),Integer(1),Integer(1), Integer(1),Integer(1),Integer(1), Integer(1),Integer(1),Integer(1))
# ==1/18
0.0555555555555555556

>>> wigner_9j(Integer(1),Integer(1),Integer(1), Integer(1),Integer(1),Integer(1), Integer(1),Integer(2), Integer(2))
# ==1/18
0.0555555555555555556

>>> wigner_9j(Integer(3),Integer(3),Integer(2), Integer(2),Integer(2),Integer(2), Integer(2),Integer(2),Integer(2))
# ==157/14700
0.0106802721088435374

>>> wigner_9j(Integer(3),Integer(3),Integer(2), Integer(3),Integer(3),Integer(2), Integer(3),Integer(3),Integer(3))
# ==3221*sqrt(70)/(246960*sqrt(105)) - 365/(3528*sqrt(70)*sqrt(105))
0.0094424774665111739

>>> wigner_9j(Integer(3),Integer(3),Integer(1), RealNumber(3.5),RealNumber(3.5),Integer(2), RealNumber(3.5),RealNumber(3.5),Integer(1))
# ==3221*sqrt(70)/(246960*sqrt(105)) - 365/(3528*sqrt(70)*sqrt(105))
0.011021667854351364

>>> wigner_9j(Integer(100),Integer(80),Integer(50), Integer(50),Integer(100), Integer(70), Integer(60),Integer(50),Integer(100))
# ==80944680186359968990/95103769817469)*sqrt(1/682288158959699477295)
0.0000325841699408828
It is an error to have arguments that are not integer or half integer values or do not fulfill the triangle relation:

```python
sage: wigner_9j(0.5,0.5,0.5, 0.5,0.5,0.5, 0.5,0.5,0.5,prec=64)
Traceback (most recent call last):
  ... ValueError: j values must be integer or half integer and fulfill the triangle relation
```

```python
sage: wigner_9j(1,1,1, 0.5,1,1.5, 0.5,1,2.5,prec=64)
# needs sage.rings.real_mpfr
Traceback (most recent call last):
  ... ValueError: j values must be integer or half integer and fulfill the triangle relation
```

```python
>>> from sage.all import *
```

```python
>>> wigner_9j(RealNumber('0.5'),RealNumber('0.5'),RealNumber('0.5'), RealNumber('0.5'),RealNumber('0.5'),RealNumber('0.5'),RealNumber('0.5'),RealNumber('0.5'),prec=Integer(64))
Traceback (most recent call last):
  ... ValueError: j values must be integer or half integer and fulfill the triangle relation
```

```python
>>> wigner_9j(Integer(1),Integer(1),Integer(1), RealNumber('0.5'),Integer(1), RealNumber('1.5'),RealNumber('0.5'),Integer(1),RealNumber('2.5'), prec=Integer(64))  # needs sage.rings.real_mpfr
Traceback (most recent call last):
  ... ValueError: j values must be integer or half integer and fulfill the triangle relation
```

**ALGORITHM:**

This function uses the algorithm of [Ed1974] to calculate the value of the 3-\(j\) symbol exactly. Note that the formula contains alternating sums over large factorials and is therefore unsuitable for finite precision arithmetic and only useful for a computer algebra system [RH2003].
1.17 Generalized functions

Sage implements several generalized functions (also known as distributions) such as Dirac delta, Heaviside step functions. These generalized functions can be manipulated within Sage like any other symbolic functions.

AUTHORS:

• Golam Mortuza Hossain (2009-06-26): initial version

EXAMPLES:

Dirac delta function:

```python
sage: dirac_delta(x)
needs sage.symbolic
dirac_delta(x)
```

```python
>>> from sage.all import *

heaviside(x)
```

Heaviside step function:

```python
sage: heaviside(x)
needs sage.symbolic
heaviside(x)
```

```python
>>> from sage.all import *

heaviside(x)
```

Unit step function:

```python
sage: unit_step(x)
needs sage.symbolic
unit_step(x)
```

```python
>>> from sage.all import *

unit_step(x)
```

Signum (sgn) function:

```python
sage: sgn(x)
needs sage.symbolic
sgn(x)
```

```python
>>> from sage.all import *

sgn(x)
```

Kronecker delta function:
class sage.functions.generalized.FunctionDiracDelta

Bases: BuiltinFunction

The Dirac delta (generalized) function, \( \delta(x) \) (dirac_delta(x)).

INPUT:
- \( x \) – a real number or a symbolic expression

DEFINITION:
Dirac delta function \( \delta(x) \), is defined in Sage as:

\[
\delta(x) = 0 \quad \text{for real} \quad x \neq 0 \quad \text{and} \quad \int_{-\infty}^{\infty} \delta(x) dx = 1
\]

Its alternate definition with respect to an arbitrary test function \( f(x) \) is

\[
\int_{-\infty}^{\infty} f(x) \delta(x - a) dx = f(a)
\]

EXAMPLES:

sage: dirac_delta(1) # needs sage.symbolic
0
sage: dirac_delta(0)
dirac_delta(0)
sage: dirac_delta(x)
dirac_delta(x)
sage: integrate(dirac_delta(x), x, -1, 1, algorithm='sympy') # needs sympy
1

REFERENCES:
- Wikipedia article Dirac_delta_function
class sage.functions.generalized.FunctionHeaviside

Bases: GinacFunction

The Heaviside step function, \( H(x)(\text{heaviside}(x)) \).

INPUT:

- \( x \) – a real number or a symbolic expression

DEFINITION:

The Heaviside step function, \( H(x) \) is defined in Sage as:

\[
H(x) = 0 \text{ for } x < 0 \text{ and } H(x) = 1 \text{ for } x > 0
\]

See also:

unit_step()

EXAMPLES:

```
sage: # needs sage.symbolic
sage: heaviside(-1)
0
sage: heaviside(1)
1
sage: heaviside(0)
heaviside(0)
sage: heaviside(x)
heaviside(x)
sage: heaviside(-1/2)               #...
→ needs sage.symbolic
0
sage: heaviside(exp(-1000000000000000000000))  #...
→ needs sage.symbolic
1
```

```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> heaviside(-Integer(1))
0
>>> heaviside(Integer(1))
1
>>> heaviside(Integer(0))
heaviside(0)
>>> heaviside(x)
heaviside(x)

>>> heaviside(-Integer(1)/Integer(2))   #...
→ # needs sage.symbolic
0
>>> heaviside(exp(-Integer(1000000000000000000000))) #...
→ # needs sage.symbolic
1
```

REFERENCES:

- Wikipedia article Heaviside_function
class sage.functions.generalized.FunctionKroneckerDelta

Bases: BuiltinFunction

The Kronecker delta function \( \delta_{m,n} \) (kronecker_delta(m, n)).

INPUT:

- \( m \) – a number or a symbolic expression
- \( n \) – a number or a symbolic expression

DEFINITION:

Kronecker delta function \( \delta_{m,n} \) is defined as:

\[
\delta_{m,n} = \begin{cases} 
0 & \text{for } m \neq n \\
1 & \text{for } m = n
\end{cases}
\]

EXAMPLES:

```sage
sage: kronecker_delta(1, 2)  # ... needs sage.rings.complex_interval_field
0
sage: kronecker_delta(1, 1)  # ... needs sage.rings.complex_interval_field
1
sage: m, n = var('m,n')     # ... needs sage.symbolic
sage: kronecker_delta(m, n) # ... needs sage.symbolic
kronecker_delta(m, n)
```

```python
>>> from sage.all import *
>>> kronecker_delta(Integer(1), Integer(2))  # needs sage.rings.complex_interval_field
0
>>> kronecker_delta(Integer(1), Integer(1))  # needs sage.rings.complex_interval_field
1
>>> m, n = var('m,n') # ... needs sage.symbolic
>>> kronecker_delta(m, n) # ... needs sage.symbolic
kronecker_delta(m, n)
```

REFERENCES:

- Wikipedia article Kronecker_delta

class sage.functions.generalized.FunctionSignum

Bases: BuiltinFunction

The signum or sgn function \( \text{sgn}(x) \) (\text{sgn}(x)).

INPUT:

- \( x \) – a real number or a symbolic expression

DEFINITION:

The sgn function, \( \text{sgn}(x) \) is defined as:

\[
\text{sgn}(x) = \begin{cases} 
1 & \text{for } x > 0 \\
0 & \text{for } x = 0 \\
-1 & \text{for } x < 0
\end{cases}
\]

EXAMPLES:
Functions, Release 10.4

\begin{verbatim}
sage: sgn(-1)
sage: sgn(1)
sage: sgn(0)
sage: sgn(x)   #needs sage.symbolic
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> sgn(-Integer(1))
-1
>>> sgn(Integer(1))
1
>>> sgn(Integer(0))
0
>>> sgn(x)   #needs sage.symbolic
\end{verbatim}

We can also use \texttt{sign}:

\begin{verbatim}
sage: sign(1)
sage: sign(0)
sage: a = AA(-5).nth_root(7)   #needs sage.rings.number_field
sage: sign(a)   #needs sage.rings.number_field
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> sign(Integer(1))
1
>>> sign(Integer(0))
0
>>> a = AA(-Integer(5)).nth_root(Integer(7))   # needs sage.rings.number_field
>>> sign(a)   #needs sage.rings.number_field
\end{verbatim}

REFERENCES:

- Wikipedia article \textit{Sign function}

\texttt{class} \texttt{sage.functions.generalized.FunctionUnitStep}

Bases: \texttt{GinacFunction}

The unit step function, \( u(x) \) (unit\_step(\( x \))).

INPUT:

- \( x \) – a real number or a symbolic expression

DEFINITION:

The unit step function, \( u(x) \) is defined in Sage as:
\[ u(x) = 0 \text{ for } x < 0 \text{ and } u(x) = 1 \text{ for } x \geq 0 \]

See also:

\texttt{heaviside()}

EXAMPLES:

```
sage: # needs sage.symbolic
sage: unit_step(-1)
0
sage: unit_step(1)
1
sage: unit_step(0)
1
sage: unit_step(x)
unit_step(x)
sage: unit_step(-\exp(-10000000000000000000))
0

>>> from sage.all import *
>>> # needs sage.symbolic
>>> unit_step(-Integer(1))
0
>>> unit_step(Integer(1))
1
>>> unit_step(Integer(0))
1
>>> unit_step(x)
unit_step(x)
>>> unit_step(-exp(-Integer(10000000000000000000)))
0
```

1.18 Counting primes

EXAMPLES:

```
sage: z = sage.functions.prime_pi.PrimePi()
sage: loads(dumps(z))
prime_pi
sage: loads(dumps(z)) == z
True

>>> from sage.all import *
>>> z = sage.functions.prime_pi.PrimePi()
>>> loads(dumps(z))
prime_pi
>>> loads(dumps(z)) == z
True
```

AUTHORS:

- R. Andrew Ohana (2009): initial version of efficient prime_pi
- R. Andrew Ohana (2011): complete rewrite, ~5x speedup
• Dima Pasechnik (2021): removed buggy cython code, replaced it with calls to primecount/primecountpy spkg

class sage.functions.prime_pi.PrimePi

    Bases: BuiltinFunction

    The prime counting function, which counts the number of primes less than or equal to a given value.
    INPUT:
    • \( x \) – a real number
    • prime_bound – (default 0) a real number < \( 2^{32} \); \texttt{prime\_pi()} \ will make sure to use all the primes up to prime_bound (although, possibly more) in computing \texttt{prime\_pi}, this can potentially speedup the time of computation, at a cost to memory usage.

    OUTPUT:
    integer – the number of primes \( \leq x \)

    EXAMPLES:

    These examples test common inputs:

    sage: # needs sage.symbolic
    sage: prime_pi(7)
    4
    sage: prime_pi(100)
    25
    sage: prime_pi(1000)
    168
    sage: prime_pi(100000)
    9592
    sage: prime_pi(500509)
    41581

    >>> from sage.all import *
    >>> # needs sage.symbolic
    >>> prime_pi(Integer(7))
    4
    >>> prime_pi(Integer(100))
    25
    >>> prime_pi(Integer(1000))
    168
    >>> prime_pi(Integer(100000))
    9592
    >>> prime_pi(Integer(500509))
    41581

    The following test is to verify that \texttt{Issue \#4670} has been essentially resolved:

    sage: prime_pi(10^{10}) #...
    455052511

    >>> from sage.all import *
    >>> # needs sage.symbolic
    >>> prime_pi(I\texttt{nteger}(10) ** Integer(10)) ...
    455052511

    The \texttt{prime\_pi()} function also has a special plotting method, so it plots quickly and perfectly as a step function:
sage: P = plot(prime_pi, 50, 100)  # needs sage.plot sage.symbolic

>>> from sage.all import *
>>> P = plot(prime_pi, Integer(50), Integer(100))  # needs sage.plot sage.symbolic

plot (xmin=0, xmax=100, vertical_lines=True, **kwds)

Draw a plot of the prime counting function from xmin to xmax. All additional arguments are passed on to the line command.

WARNING: we draw the plot of prime_pi as a stairstep function with explicitly drawn vertical lines where the function jumps. Technically there should not be any vertical lines, but they make the graph look much better, so we include them. Use the option vertical_lines=False to turn these off.

EXAMPLES:

sage: plot(prime_pi, 1, 100)  # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive

sage: prime_pi.plot(1, 51, thickness=2, vertical_lines=False)  # needs sage.plot sage.symbolic
Graphics object consisting of 16 graphics primitives

>>> from sage.all import *

sage: plot(prime_pi, Integer(1), Integer(100))  # needs sage.plot sage.symbolic
Graphics object consisting of 1 graphics primitive

sage: prime_pi.plot(Integer(1), Integer(51), thickness=Integer(2), vertical_lines=False)  # needs sage.plot sage.symbolic
Graphics object consisting of 16 graphics primitives

sage.functions.prime_pi.legendre_phi(x, a)

Legendre’s formula, also known as the partial sieve function, is a useful combinatorial function for computing the prime counting function (the prime_pi method in Sage). It counts the number of positive integers \( \leq x \) that are not divisible by the first \( a \) primes.

INPUT:

• \( x \) – a real number
• \( a \) – a non-negative integer

OUTPUT:

integer – the number of positive integers \( \leq x \) that are not divisible by the first \( a \) primes

EXAMPLES:

sage: legendre_phi(100, 0)
100
sage: legendre_phi(29375, 1)
14688
sage: legendre_phi(91753, 5973)
2893
sage: legendre_phi(4215701455, 6450023226)
1
Legendre’s formula, also known as the partial sieve function, is a useful combinatorial function for computing the prime counting function (the `prime_pi` method in Sage). It counts the number of positive integers \( \leq x \) that are not divisible by the first \( a \) primes.

**INPUT:**
- \( x \) – a real number
- \( a \) – a non-negative integer

**OUTPUT:**
integer – the number of positive integers \( \leq x \) that are not divisible by the first \( a \) primes

**EXAMPLES:**

```python
>>> from sage.all import *
>>> legendre_phi(Integer(100), Integer(0))
100
>>> legendre_phi(Integer(29375), Integer(1))
14688
>>> legendre_phi(Integer(91753), Integer(5973))
2893
>>> legendre_phi(Integer(4215701455), Integer(6450023226))
1
```

1.19 Symbolic minimum and maximum

Sage provides a symbolic maximum and minimum due to the fact that the Python builtins `max()` and `min()` are not able to deal with variables as users might expect. These functions wait to evaluate if there are variables.

Here you can see some differences:

```python
sage: max(x, x^2)
# needs sage.symbolic
x
```

(continues on next page)
This works as expected for more than two entries:

```python
sage: # needs sage.symbolic
sage: max(3, 5, x)
5
sage: min(3, 5, x)
3
sage: max_symbolic(3, 5, x)
max(x, 5)
sage: min_symbolic(3, 5, x)
min(x, 3)
```

```python
from sage.all import *

>>> from sage.all import *

>>> max(Integer(3), Integer(5), x)
5
>>> min(Integer(3), Integer(5), x)
3
>>> max_symbolic(Integer(3), Integer(5), x)
max(x, 5)
>>> min_symbolic(Integer(3), Integer(5), x)
min(x, 3)
```

```python
class sage.functions.min_max.MaxSymbolic
    Bases: MinMax_base

Symbolic max function.

The Python built-in `max()` function does not work as expected when symbolic expressions are given as arguments. This function delays evaluation until all symbolic arguments are substituted with values.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: max_symbolic(3, x)
max(3, x)
sage: max_symbolic(3, x).subs(x=5)
```

(continues on next page)
Function documentation continues from the previous page:

```python
sage: max_symbolic(3, 5, x)
max(x, 5)
sage: max_symbolic([3, 5, x])
max(x, 5)

```  

```python
>>> from sage.all import *

>>> # needs sage.symbolic

>>> max_symbolic(Integer(3), x)
max(3, x)
>>> max_symbolic(Integer(3), x).subs(x=Integer(5))
5
>>> max_symbolic(Integer(3), Integer(5), x)
max(x, 5)
>>> max_symbolic([Integer(3), Integer(5), x])
max(x, 5)
```

```python
class sage.functions.min_max.MinMax_base

Bases: BuiltinFunction

eval_helper(this_f, builtin_f, initial_val, args)

EXAMPLES:

```python
sage: # needs sage.symbolic

sage: max_symbolic(3, x) # indirect doctest
max(x, 3)
sage: max_symbolic([5.0]) # indirect doctest
5.0

```  

```python
>>> from sage.all import *

>>> # needs sage.symbolic

>>> max_symbolic(Integer(3), Integer(5), x) # indirect doctest
max(x, 5)
>>> max_symbolic([5.0]) # indirect doctest
5.0

```  

```python
class sage.functions.min_max.MinSymbolic

Bases: MinMax_base

Symbolic min function.

The Python builtin `min()` function does not work as expected when symbolic expressions are given as arguments. This function delays evaluation until all symbolic arguments are substituted with values.

EXAMPLES:

```python
sage: # needs sage.symbolic

sage: min_symbolic(3, x)
min(x, 3)
sage: min_symbolic([5.0]) # indirect doctest
5.0

```  

(continues on next page)
functions, release 10.4

(continued from previous page)

```
min(3, x)
sage: min_symbolic(3, x).subs(x=5)
3
sage: min_symbolic(3, 5, x)
min(x, 3)
sage: min_symbolic([3, 5, x])
min(x, 3)
```

```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> min_symbolic(Integer(3), x)
min(3, x)
>>> min_symbolic(Integer(3), x).subs(x=Integer(5))
3
>>> min_symbolic(Integer(3), Integer(5), x)
min(x, 3)
>>> min_symbolic([Integer(3), Integer(5), x])
min(x, 3)
```

Please find extensive developer documentation for creating new functions in Symbolic Calculus, in particular in the section Classes for symbolic functions.
INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.functions.airy, 176
sage.functions.bessel, 185
sage.functions.error, 60
sage.functions.exp_integral, 212
sage.functions.generalized, 244
sage.functions.hyperbolic, 34
sage.functions.hypergeometric, 153
sage.functions.jacobi, 171
sage.functions.log, 1
sage.functions.min_max, 252
sage.functions.orthogonal_polys, 88
sage.functions.other, 120
sage.functions.piecewise, 65
sage.functions.prime_pi, 249
sage.functions.special, 143
sage.functions.spike_function, 85
sage.functions.transcendental, 52
sage.functions.trig, 14
sage.functions.wigner, 234
INDEX

A
airy_ai() (in module sage.functions.airy), 179
airy_bi() (in module sage.functions.airy), 182
approximate() (sage.functions.transcendental.DickmanRho method), 53

B
Bessel() (in module sage.functions.bessel), 189

C
ChebyshevFunction (class in sage.functions.orthogonal_polys), 93
clebsch_gordan() (in module sage.functions.wigner), 234
closed_form() (in module sage.functions.hypergeometric), 169
convolution() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 66
critical_points() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 68

domain() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 69
domains() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 69

deflated() (sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method), 159
deprecated_function_alias() (sage.functions.orthogonal_polys.Func_assoc_legendre_P method), 95
DickmanRho (class in sage.functions.transcendental), 52
domains() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 69

E
eliptic_eu_f() (in module sage.functions.special), 151
eliptic_j() (in module sage.functions.special), 151
EllipticE (class in sage.functions.special), 145
EllipticEC (class in sage.functions.special), 146
EllipticEU (class in sage.functions.special), 146
EllipticF (class in sage.functions.special), 147
EllipticKC (class in sage.functions.special), 147
EllipticPi (class in sage.functions.special), 148
end_points() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 69
eval_algebraic() (sage.functions.orthogonal_polys.Func_chebyshev_T method), 98
eval_algebraic() (sage.functions.orthogonal_polys.Func_chebyshev_U method), 101
eval_formula() (sage.functions.orthogonal_polys.Func_chebyshev_T method), 100
eval_formula() (sage.functions.orthogonal_polys.Func_chebyshev_U method), 102
eval_formula() (sage.functions.orthogonal_polys.Func_hahn method), 104
eval_formula() (sage.functions.orthogonal_polys.Func_krawtchouk method), 109
eval_formula() (sage.functions.orthogonal_polys.Func_legendre_Q method), 113
eval_formula() (sage.functions.orthogonal_polys.Func_meixner method), 114
eval_formula() (sage.functions.orthogonal_polys.OrthogonalFunction method), 119
eval_gen_poly() (sage.functions.orthogonal_polys.Func_assoc_legendre_P method), 96
eval_helper() (sage.functions.min_max.MinMax_base method), 254
eval_poly() (sage.functions.orthogonal_polys.Func_assoc_legendre_P method), 97
eval_recursive() (sage.functions.orthogonal_polys Func_assoc_legendre_Q method), 97
eval_recursive() (sage.functions.orthogonal_polys.Func_hahn method), 105
eval_recursive() (sage.functions.orthogonal_polys.Func_krawtchouk method), 109
eval_recursive() (sage.functions.orthogonal_polys.Func_meixner
method), 115
exponential_integral_1() (in module sage.functions.exp_integral), 232
expression_at() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods
method), 70
expressions() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods
method), 71
extension() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods
method), 71

Function_arccosh (class in sage.functions.hyperbolic), 35
Function_arccoth (class in sage.functions.hyperbolic), 16
Function_arccsc (class in sage.functions.hyperbolic), 17
Function_arccsch (class in sage.functions.hyperbolic), 39
Function_arcsec (class in sage.functions.hyperbolic), 18
Function_arcsech (class in sage.functions.hyperbolic), 40
Function_arcsin (class in sage.functions.hyperbolic), 19
Function_arcsinh (class in sage.functions.hyperbolic), 41
Function_arctan (class in sage.functions.trig), 21
Function_arctan2 (class in sage.functions.trig), 22
Function_arctanh (class in sage.functions.hyperbolic), 42
Function_arg (class in sage.functions.other), 121
Function_Bessel_I (class in sage.functions.bessel), 194
Function_Bessel_J (class in sage.functions.bessel), 197
Function_Bessel_K (class in sage.functions.bessel), 199
Function_Bessel_Y (class in sage.functions.bessel), 202
Function_binomial (class in sage.functions.other), 123
Function_cases (class in sage.functions.other), 125
Function_celld (class in sage.functions.other), 126
Function_conjugate (class in sage.functions.other), 129
Function_cos (class in sage.functions.trig), 25
Function_cos_integral (class in sage.functions.exp_integral), 212
Function_cosh (class in sage.functions.hyperbolic), 44
Function_cosh_integral (class in sage.functions.exp_integral), 215
Function_cot (class in sage.functions.trig), 26
Function_coth (class in sage.functions.hyperbolic), 45
Function_crootof (class in sage.functions.other), 130
Function_csc (class in sage.functions.trig), 28
Function_csch (class in sage.functions.hyperbolic), 46
Function_dilog (class in sage.functions.log), 1
Function_elementof (class in sage.functions.other), 130
Function_erf (class in sage.functions.error), 62
Function_erfc (class in sage.functions.error), 63
Function_erfi (class in sage.functions.error), 64
Function_erfinv (class in sage.functions.error), 64
Function_exp (class in sage.functions.log), 3
Function_exp_integral (class in sage.functions.exp_integral), 217
Function_exp_integral_e (class in sage.functions.exp_integral), 219
Function_exp_integral_e1 (class in sage.functions.exp_integral), 221
Function_exp_polar (class in sage.functions.log), 5
Function_factorial (class in sage.functions.other), 131
Function_floor (class in sage.functions.other), 133
Function_frac (class in sage.functions.other), 136
Function_Fresnel_cos (class in sage.functions.error), 61
Function_Fresnel_sin (class in sage.functions.error), 61
Function_Hankel1 (class in sage.functions.bessel), 204
Function_Hankel2 (class in sage.functions.bessel), 205
Function_harmonic_number (class in sage.functions.log), 6
Function_harmonic_number_generalized (class in sage.functions.log), 7
Function_HurwitzZeta (class in sage.functions.transcendental), 54
Function_imag_part (class in sage.functions.other), 137
Function_lambert_w (class in sage.functions.log), 9
Function_limit (class in sage.functions.other), 138
Function_log1 (class in sage.functions.log), 11
Function_log2 (class in sage.functions.log), 12
Function_log_integral (class in sage.functions.exp_integral), 223
Function_log_integral_offset (class in sage.functions.exp_integral), 224
Function_Order (class in sage.functions.other), 120
Function_polylog (class in sage.functions.log), 12
Function_prod (class in sage.functions.other), 139
Function_real_nth_root (class in sage.functions.other), 139
Function_real_part (class in sage.functions.other), 141
Function_sec (class in sage.functions.trig), 30
Function_sech (class in sage.functions.hyperbolic), 48
Function_sin (class in sage.functions.trig), 31
Function_sin_integral (class in sage.functions.exp_integral), 226
Function_sinh (class in sage.functions.hyperbolic), 49
Function_sinh_integral (class in sage.functions.exp_integral), 230
Function_sqrt (class in sage.functions.other), 142
Function_stieltjes (class in sage.functions.transcendental), 54
Function_Struve_H (class in sage.functions.bessel), 206
Function_Struve_L (class in sage.functions.bessel), 206
Function_zeta (class in sage.functions.transcendental), 55
Function_zetaderiv (class in sage.functions.transcendental), 57
Function_AiryAiGeneral (class in sage.functions.airy), 176
Function_AiryAiPrime (class in sage.functions.airy), 177
Function_AiryAiSimple (class in sage.functions.airy), 177
Function_AiryBiGeneral (class in sage.functions.airy), 178
Function_AiryBiPrime (class in sage.functions.airy), 179
Function_AiryBiSimple (class in sage.functions.airy), 179
Function.DiracDelta (class in sage.functions.generalized), 245
Function.Heaviside (class in sage.functions.generalized), 245
Function.KroneckerDelta (class in sage.functions.generalized), 246
Function.Signum (class in sage.functions.generalized), 247
Function.UnitStep (class in sage.functions.generalized), 248

G

gaunt () (in module sage.functions.wigner), 235
generalized() (sage.functions.hypergeometric.Hypergeometric_M.EvaluationMethods method), 167

H

hurwitz_zeta() (in module sage.functions.transcendental), 58
Hypergeometric (class in sage.functions.hypergeometric), 159
Hypergeometric_M (class in sage.functions.hypergeometric), 166
Hypergeometric_M.EvaluationMethods (class in sage.functions.hypergeometric), 167
Hypergeometric_U (class in sage.functions.hypergeometric), 167
Hypergeometric_U.EvaluationMethods (class in sage.functions.hypergeometric), 168
Hypergeometric.EvaluationMethods (class in sage.functions.hypergeometric), 159
Functions, Release 10.4

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_operands()</td>
<td>sage.functions.piecewise.PiecewiseFunction static method, 84</td>
</tr>
<tr>
<td>inverse_jacobi() (in module sage.functions.jacobi)</td>
<td>173</td>
</tr>
<tr>
<td>inverse_jacobi_f() (in module sage.functions.jacobi)</td>
<td>174</td>
</tr>
<tr>
<td>InverseJacobi (class in sage.functions.jacobi)</td>
<td>173</td>
</tr>
<tr>
<td>is_absolutely_convergent()</td>
<td>sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method, 161</td>
</tr>
<tr>
<td>is_terminating()</td>
<td>sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method, 163</td>
</tr>
<tr>
<td>is_termwise_finite()</td>
<td>sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method, 164</td>
</tr>
<tr>
<td>items()</td>
<td>sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method, 80</td>
</tr>
<tr>
<td>Jacobi (class in sage.functions.jacobi)</td>
<td>173</td>
</tr>
<tr>
<td>JacobiAmplitude (class in sage.functions.jacobi)</td>
<td>173</td>
</tr>
<tr>
<td>laplace()</td>
<td>sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method, 80</td>
</tr>
<tr>
<td>legendre_phi() (in module sage.functions.prime_pi)</td>
<td>251</td>
</tr>
<tr>
<td>MaxSymbolic (class in sage.functions.min_max)</td>
<td>253</td>
</tr>
<tr>
<td>MinMax_base (class in sage.functions.min_max)</td>
<td>254</td>
</tr>
<tr>
<td>MinSymbolic (class in sage.functions.min_max)</td>
<td>254</td>
</tr>
<tr>
<td>module</td>
<td></td>
</tr>
<tr>
<td>sage.functions.airy</td>
<td>176</td>
</tr>
<tr>
<td>sage.functions.bessel</td>
<td>185</td>
</tr>
<tr>
<td>sage.functions.error</td>
<td>60</td>
</tr>
<tr>
<td>sage.functions.exp_integral</td>
<td>212</td>
</tr>
<tr>
<td>sage.functions.generalized</td>
<td>244</td>
</tr>
<tr>
<td>sage.functions.hyperbolic</td>
<td>34</td>
</tr>
<tr>
<td>sage.functions.hypergeometric</td>
<td>153</td>
</tr>
<tr>
<td>sage.functions.jacobi</td>
<td>171</td>
</tr>
<tr>
<td>sage.functions.log</td>
<td>1</td>
</tr>
<tr>
<td>sage.functions.min_max</td>
<td>252</td>
</tr>
<tr>
<td>sage.functions.orthogonal_polys</td>
<td>88</td>
</tr>
<tr>
<td>sage.functions.other</td>
<td>120</td>
</tr>
<tr>
<td>sage.functions.piecewise</td>
<td>65</td>
</tr>
<tr>
<td>sage.functions.prime_pi</td>
<td>249</td>
</tr>
<tr>
<td>sage.functions.special</td>
<td>143</td>
</tr>
<tr>
<td>sage.functions.spike_function</td>
<td>85</td>
</tr>
<tr>
<td>sage.functions.trig</td>
<td>14</td>
</tr>
<tr>
<td>sage.functions.wigner</td>
<td>234</td>
</tr>
<tr>
<td>OrthogonalFunction (class in sage.functions.orthogonal_polys)</td>
<td>119</td>
</tr>
<tr>
<td>partial_sieve_function() (in module sage.functions.prime_pi)</td>
<td>252</td>
</tr>
<tr>
<td>pieces()</td>
<td>sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method, 81</td>
</tr>
<tr>
<td>PiecewiseFunction (class in sage.functions.piecewise)</td>
<td>66</td>
</tr>
<tr>
<td>PiecewiseFunction.EvaluationMethods</td>
<td>(class in sage.functions.piecewise), 66</td>
</tr>
<tr>
<td>plot()</td>
<td>sage.functions.prime_pi.PrimePi method, 251</td>
</tr>
<tr>
<td>plot()</td>
<td>sage.functions.spike_function.SpikeFunction method, 86</td>
</tr>
<tr>
<td>plot_fft_abs()</td>
<td>sage.functions.spike_function.SpikeFunction method, 86</td>
</tr>
<tr>
<td>plot_fft_arg()</td>
<td>sage.functions.spike_function.SpikeFunction method, 87</td>
</tr>
<tr>
<td>power_series()</td>
<td>sage.functions.transcendental.DickmanRho method, 53</td>
</tr>
<tr>
<td>PrimePi (class in sage.functions.prime_pi)</td>
<td>250</td>
</tr>
<tr>
<td>racah()</td>
<td>(in module sage.functions.wigner), 237</td>
</tr>
<tr>
<td>rational_param_as_tuple() (in module sage.functions.hypergeometric)</td>
<td>171</td>
</tr>
<tr>
<td>Sage.functions.airy</td>
<td>module, 176</td>
</tr>
<tr>
<td>Sage.functions.bessel</td>
<td>module, 185</td>
</tr>
<tr>
<td>Sage.functions.exp_integral</td>
<td>module, 212</td>
</tr>
<tr>
<td>Sage.functions.generalized</td>
<td>module, 244</td>
</tr>
<tr>
<td>Sage.functions.hyperbolic</td>
<td>module, 34</td>
</tr>
</tbody>
</table>
sage.functions.hypergeometric module, 153
sage.functions.jacobi module, 171
sage.functions.log module, 1
sage.functions.min_max module, 252
sage.functions.orthogonal_polys module, 88
sage.functions.other module, 120
sage.functions.piecewise module, 65
sage.functions.prime_pi module, 249
sage.functions.special module, 143
sage.functions.spike_function module, 85
sage.functions.transcendental module, 52
sage.functions.trig module, 14
sage.functions.wigner module, 234
simplify() (sage.functions.piecewise.PiecewiseFunction static method), 85
sorted_parameters() (sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method), 165
spherical_bessel_f() (in module sage.functions.bessel), 211
SphericalBesselJ (class in sage.functions.bessel), 207
SphericalBesselY (class in sage.functions.bessel), 208
SphericalHankel1 (class in sage.functions.bessel), 209
SphericalHankel2 (class in sage.functions.bessel), 210
SphericalHarmonic (class in sage.functions.special), 149
spike_function (in module sage.functions.spike_function), 88
SpikeFunction (class in sage.functions.spike_function), 85

terms() (sage.functions.hypergeometric.Hypergeometric.EvaluationMethods method), 165
trapezoid() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 82

U
unextend_zero() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 83

V
vector() (sage.functions.spike_function.SpikeFunction method), 87

W
which_function() (sage.functions.piecewise.PiecewiseFunction.EvaluationMethods method), 83
wigner_3j() (in module sage.functions.wigner), 237
wigner_6j() (in module sage.functions.wigner), 239
wigner_9j() (in module sage.functions.wigner), 241

Z
zeta_symmetric() (in module sage.functions.transcendental), 59