Unevaluated Integrals

In [1]:
```
i = Integral(log((sin(x)**2+1)*sqrt(x**2+1)),(x,0,y))
```

In [2]:
```
i
```

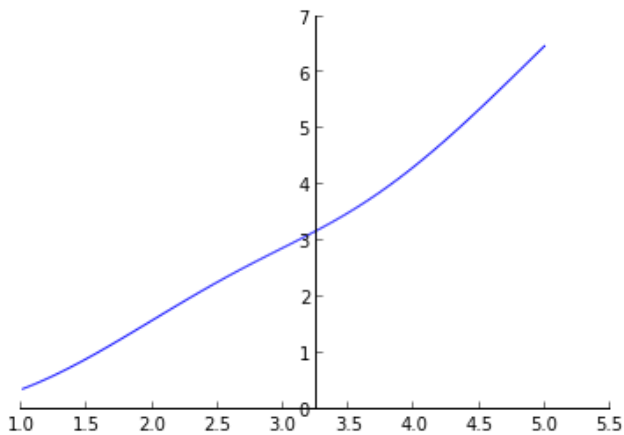
Out[2]:

$$\int_0^y \log\left(\sqrt{x^2+1}\left(\sin^2(x)+1\right)\right) dx$$

In [3]:
```
i.evalf(subs={y:1})
```

Out[3]: 0.358090090085057

In [4]:
```
plot(i,(1,5))
```



Out[4]:

```
                                        Plot object containing:
[0]: cartesian line: Integral(log(sqrt(x**2 + 1)*(sin(x)**2 + 1)), (x, 0, y))


for y over (1.0, 5.0)
```

Infinite Sums

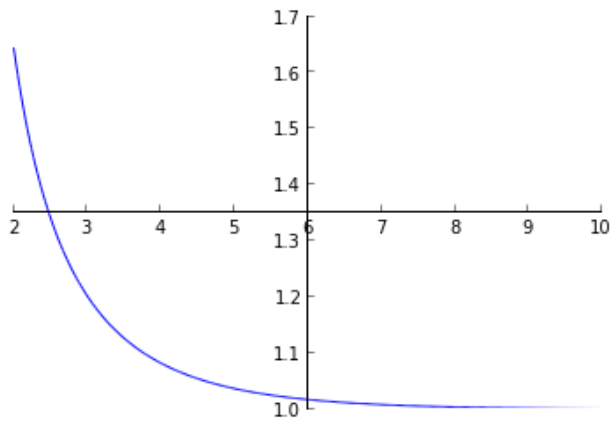
In [5]:
```
s = summation(1/x**y,(x,1,oo))
```

In [6]:
```
s
```

Out[6]:

$$\sum_{x=1}^{\infty} x^{-y}$$

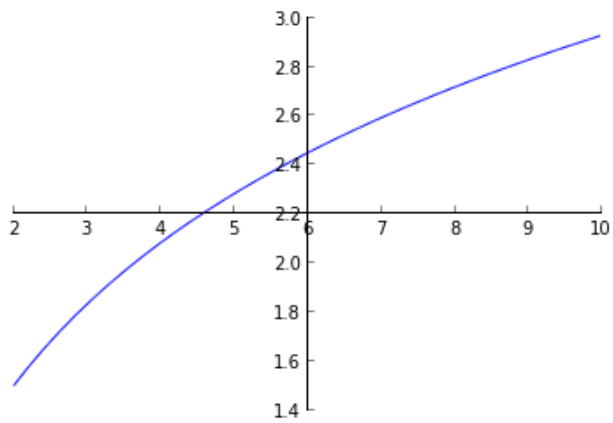
In [7]:
```
plot(s, (2,10))
```



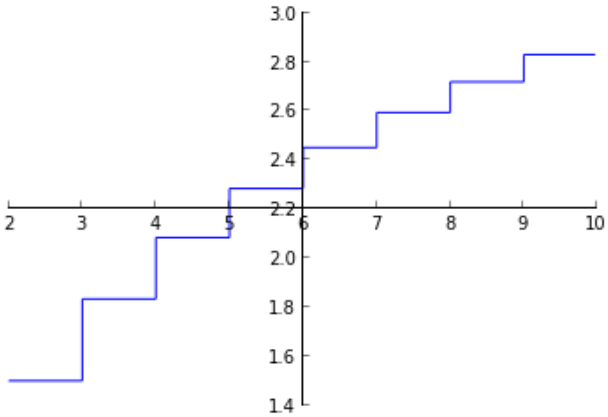
Out[7]:
```
Plot object containing:
[0]: cartesian line: Sum(x**(-y), (x, 1, oo)) for y over (2.0, 10.0)
```

Finite sums

In [8]:
```
p = plot(summation(1/x,(x,1,y)), (2,10))
```



In [9]:
```
p[0].only_integers = True
p[0].steps = True
p.show()
```



Numerical Solutions of Equations

This one is trickier because `nsolve()` is not a sympy expression. So we will have to wrap it in one. It's not yet clear (at least to the author of the plotting module) what is the Right Way TM to do it in sympy (or if there is a right way). For the moment we will use `implemented_function()`.

In [10]:
```
from sympy.utilities.lambdify import implemented_function
```

In [11]:
```
help(implemented_function)
```

```
Help on function implemented_function in module sympy.utilities.lambdify:

implemented_function(symfunc, implementation)
    Add numerical ``implementation`` to function ``symfunc``.

    ``symfunc`` can be an ``UndefinedFunction`` instance, or a name sting.
    In the latter case we create an ``UndefinedFunction`` instance with that
    name.

    Be aware that this is a quick workaround, not a general method to create
    special symbolic functions. If you want to create a symbolic function to be
    used by all the machinery of sympy you should subclass the ``Function``
    class.

    Parameters
    ----------
    symfunc : ``str`` or ``UndefinedFunction`` instance
       If ``str``, then create new ``UndefinedFunction`` with this as
       name.  If `symfunc` is a sympy function, attach implementation to it.
    implementation : callable
       numerical implementation to be called by ``evalf()`` or ``lambdify``

    Returns
    -------
    afunc : sympy.FunctionClass instance
       function with attached implementation

    Examples
    --------
    >>> from sympy.abc import x, y, z
    >>> from sympy.utilities.lambdify import lambdify, implemented_function
    >>> from sympy import Function
    >>> f = implemented_function(Function('f'), lambda x: x+1)
    >>> lam_f = lambdify(x, f(x))
    >>> lam_f(4)
    5
```

The equation we want to solve will be the one giving the magnetisation in the Mean Field Ising model.

In [12]:
```
symbols('t m')
equ = tanh(m/t) - m
```

In [13]:
```
nsolve(equ.subs({t:10}),10) # T higher than the critical temperature
```

Out[13]: 0.0

In [14]:
```
nsolve(equ.subs({t:0.2}),10) # T lower than the critical temperature
```

Out[14]: 0.999909121715233

In [18]:
```
f = implemented_function('f', lambda T : nsolve(equ.subs({t:T}),10))
```

In [27]:
```
f(10).evalf()
```

Out[27]: 0

In [26]:
```
f(0.2).evalf()  # This has stopped working at some point in master.
#I do not know what happened. It was a cool example. The problem is
#with implemented_function and not with the plotting module.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-26-14d33654d943> in <module>()
----> 1 f(0.2).evalf()

/home/stefan/scientific_python_stack/sympy/sympy/core/function.pyc in
__new__(cls, *args, **options)
    705         args = map(sympify, args)
    706         result = super(AppliedUndef, cls).__new__(cls, *args, **options)
--> 707         result.nargs = len(args)
    708         return result
    709

AttributeError: 'Float' object has no attribute 'nargs'
```

In [21]:
```
plot(f(t),(0.2,2))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-ca50bada3dff> in <module>()
----> 1 plot(f(t),(0.2,2))

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in plot(*args,
**kwargs)
    340         p.extend(plot_argument)
    341     if show:
--> 342         p.show()
    343     return p
    344

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in show(self)
    175             self._backend.close()
    176         self._backend = self.backend(self)
--> 177         self._backend.show()
    178
    179     def save(self, path):

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in show(self)
    920
    921     def show(self):
--> 922         self.process_series()
    923         #TODO after fixing https://github.com/ipython/ipython/issues/1255
    924         # you can uncomment the next line and remove the pyplot.show()
call

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in
process_series(self)
    833             # Create the collections
    834             if s.is_2Dline:
--> 835                 collection = LineCollection(s.get_segments())
    836                 self.ax.add_collection(collection)
    837             elif s.is_contour:

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in
get_segments(self)
    439
    440     def get_segments(self):
--> 441         points = self.get_points()
    442         if self.steps == True:
    443             x = np.array((points[0],points[0])).T.flatten()[1:]

/home/stefan/scientific_python_stack/sympy/sympy/plotting/plot.pyc in
get_points(self)
    508             list_x = np.linspace(self.start, self.end,
num=self.nb_of_points)
    509         f = vectorized_lambdify([self.var], self.expr)
--> 510         list_y = f(list_x)
    511         return (list_x, list_y)
    512

/home/stefan/scientific_python_stack/sympy/sympy/plotting
/experimental_lambdify.pyc in __call__(self, *args)
    145                 self.lambda_func = experimental_lambdify(self.args,
self.expr, use_python_math=True)
    146                 self.vector_func = np.vectorize(self.lambda_func, otypes=
[np.float])
--> 147                 results = self.vector_func(*args)
```

In [ ]: