TACC -- 2. Tutorial on Symbolic Expressions and 2D Plotting

TACC -- 2. Tutorial on Symbolic Expressions and 2D Plotting

William Stein

University of Washington

This tutorial is mainly about using Sage as a CAS (=Computer Algebra System), i.e., the Mathematica-style aspects of Sage.

```
```

Creating Symbolic Expressions

The "symbolic variable" or indeterminate x is predefined when you startup Sage:

```
x
```

```
x
```

```
(x+2)^3
```

```
(x + 2)^3
```

```
parent(x)
```

```
Symbolic Ring
```

```
type(x)
```

```
<type 'sage.symbolic.expression.Expression'>
```

d to define some symbolic variables. You can separate the variables by commas

or spaces in the var command.

```
reset()
x + y
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'y' is not defined
```

```
var('y theta')
```

```
(y, theta)
```

```
(x+y+theta)^(2+pi)
```

```
(theta + x + y)^(pi + 2)
```

```
var('a,b,c')
expand(a^(b+c))
```

```
a^c*a^b
```

```
show(expand((x+y+theta)^(2+pi)))
```

$$\theta^2(\theta+x+y)^{\pi}+2\,\theta x(\theta+x+y)^{\pi}+2\,\theta y(\theta+x+y)^{\pi}+x^2(\theta+x+y)^{\pi}+2\,xy$$

```
f = (x+y+theta)^(2+pi); show(f)
```

$$(\theta+x+y)^{\pi+2}$$

```
show(f.simplify_full())
```

$$\left(2\,(\theta+x)y+\theta^2+2\,\theta x+x^2+y^2\right)(\theta+x+y)^{\pi}$$

```
print latex(f)
```

```
{\left(\theta + x + y\right)}^{\pi + 2}
```

```
sin(2*y)
```

```
var('x y z epsilon')
```

```
(x, y, z, epsilon)
```

```
cos(x^3) - y^2*z + epsilon
```

```
-y^2*z + epsilon + cos(x^3)
```

```

```

```

```

NOTE: If you want symbolic variables to just "magically" spring into existence when you use them, type **automatic_names(True)**. To turn this off, type **automatic_names(False).**

```
automatic_names(True)
```

```
expand( (fred + sam + verlkja + anything)^3 )
```

```
anything^3 + 3*anything^2*fred + 3*anything^2*sam +
3*anything^2*verlkja + 3*anything*fred^2 + 6*anything*fred*sam
6*anything*fred*verlkja + 3*anything*sam^2 + 6*anything*sam*ver
+ 3*anything*verlkja^2 + fred^3 + 3*fred^2*sam + 3*fred^2*verlk
3*fred*sam^2 + 6*fred*sam*verlkja + 3*fred*verlkja^2 + sam^3 +
3*sam^2*verlkja + 3*sam*verlkja^2 + verlkja^3
```

Using automatic_names is usually a bad idea, since it can easily lead to subtle bugs:

```
theta^3 - x + y + thetta        # "thetta" -- a typo!
```

```
theta^3 + thetta - x + y
```

```
automatic_names(False)
```

```
x + y + askdf
```

```
Traceback (click to the left of this block for traceback)
...
NameError: name 'askdf' is not defined
```

Using automatic_names also makes it so you don't have to use the foo.method(...) notation:

```
automatic_names(True)
```

```
f = 7*12
```

```
f.prime_to_m_part(2)
```

```
21
```

```
prime_to_m_part(f, 2)
```

```
21
```

```
automatic_names(False)
```

```

```

Problem: Create the following expressions:

$$\sin^5(x)\cos^2(x), \qquad \frac{x^3}{x^3+1}, \qquad k \cdot P \cdot \left(1 - \frac{P}{K}\right).$$

Tip: that you *must* put in an asterisk (*) for multiplication.

Tip: You can edit the HTML between cells by double clicking on it. You can create new HTML areas by shift-clicking on the blue bar that appears just above a compute cell.

Most standard functions are defined in Sage. They are named lowercase, e.g.,

`sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech, csch, coth, log, exp, etc.`

```
var('x,y')
sin(x) + cos(y) - tan(x/y) + sec(x*csc(y))^5
```

```
sec(x*csc(y))^5 + sin(x) + cos(y) - tan(x/y)
```

Problem: Construct the symbolic expression

$$\sin(x^{\cos(y)} + \theta) + \coth(2x) + \log(3x) \cdot \exp(y^3).$$

Making substitutions

Use the **subs** method to replace any variables by other variables.

```
var('x,y')
f = sin(x) + cos(y) - x^2 + y
```

```
f.subs(x=5)
```

```
y + sin(5) + cos(y) - 25
```

```
f.subs(x=y, y=x)
```

```
-y^2 + x + sin(y) + cos(x)
```

```
f
```

```
-x^2 + y + sin(x) + cos(y)
```

Problem: Replace x by $\sin(y) - x$ in the expression $x^3 + xy$.

```

```

```

```

Expanding Expressions

To expand a symbolic expression with exponents, use the **expand** method (or function) -- we saw this above:

```
var('x,y')
f = (x+2*y)^3
f
```

```
(x + 2*y)^3
```

```
f.expand().show()                    # tip -- using show makes the
output nicer
```

$$x^3 + 6\,x^2 y + 12\,x y^2 + 8\,y^3$$

```
show(f)
```

$$(x + 2\,y)^3$$

```
latex(f)
```

```
+ 2 \, y\right)}^{3}
```

Problem: Expand the expression $(2\sin(x) - \cos(y))^5$.

Symbolic Unit Conversions

As of Sage-4.4, Sage has a massive symbolic unit conversion package, which was written by David Ackerman (a UW undergrad) recently, as a funded student project after he took Math 480 (the Sage class) last year.

```
a = units.length.rope
```

```
a.convert(units.length.meter)
```

```
762/125*meter
```

```
a = 170*units.mass.pound
```

```
a.convert(units.mass.dalton)
```

```
4.64371586715522e28*dalton
```

```
a = 10 * units.mass.kilogram*units.length.meter /
units.time.minute^2; a
```

```
10*kilogram*meter/minute^2
```

```
a.convert(units.force.newton)
```

```
1/360*newton
```

```
units.force.newton?
```

File: /Users/wstein/sage/build/sage/local/lib/python2.6/site-packages/sage/symbolic/units.py

Type: <class 'sage.symbolic.units.UnitExpression'>

Definition: units.force.newton(*args, **kwds)

Docstring:

```
SI derived unit of force.
Defined to be kilogram*meter/second^2.
```

Problem: Convert **17 meters/second^2** to **miles/minute^2**.

```
#hint
d = 17 * units.length.meter / units.time.second^2; d
```

```
17*meter/second^2
```

Problem: Do a simple conversion that might come up in your own research.

Creating Symbolic Functions

To create a symbolic function, use the notation **f(x,y) = x^3 + y**. A symbolic function is just like a symbolic expression, except you can call it without having to explicitly use subs or name variables and be sure that the order is what you want.

```
var('x,y')
f = y + x^3; f
```

```
    x^3 + y
```

```
f.subs(y=2,x=3)
```

```
    29
```

```
preparse('f(y,x) = x^3 + y')
```

```
    '__tmp__=var("y,x"); f = symbolic_expression(x**Integer(3) +
    y).function(y,x)'
```

```
f(y,x) = x^3 + y
f
```

```
    (y, x) |--> x^3 + y
```

```
f(2,3)
```

```
    29
```

```
f(pi,e)
```

```
    pi + e^3
```

```
```

fast_float: this is *critical* to know about if you're serious about using numerical Python tools that involve repeated function evaluation. (See also **fast_callable** for complex and other inputs.)

```
g = fast_float(f,y,x)
```

```
g(2.3, 1.5)
```

```
    5.6749999999999998
```

```
f(2.3, 1.5)
```

```
    5.6750000000000000
```

```
a = float(2.3); b = float(1.5)
timeit('g(a,b)')
```

```
    625 loops, best of 3: 533 ns per loop
```

```
timeit('f(a,b)')
```

```
    625 loops, best of 3: 507 µs per loop
```

```
```

```
```

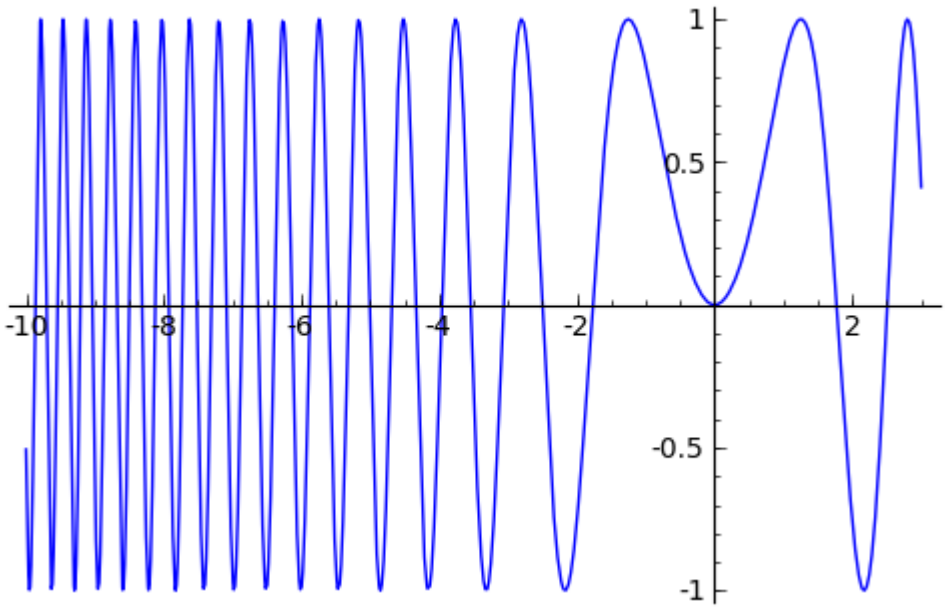
Problem: Create the functions
$x \mapsto x^3 + 1, \qquad (x, y) \mapsto \sin(x) - \cos(y)/y, \qquad (a, x, \theta) \mapsto ax + \theta^2.$

Problem: Evlauate $\sin(x) - \cos(y)/y$ quickly using fast_float.

2D Plotting

Use the **plot** command to plot a function of one variable. *TIP*: Type **plot(<tab key>** to find out much more about the plot command.

```
var('x')
plot(sin(x^2), (x,-10,3))
```




```
P = plot(sin(x^2), (x,-10,3))
P.save('image.pdf')
```

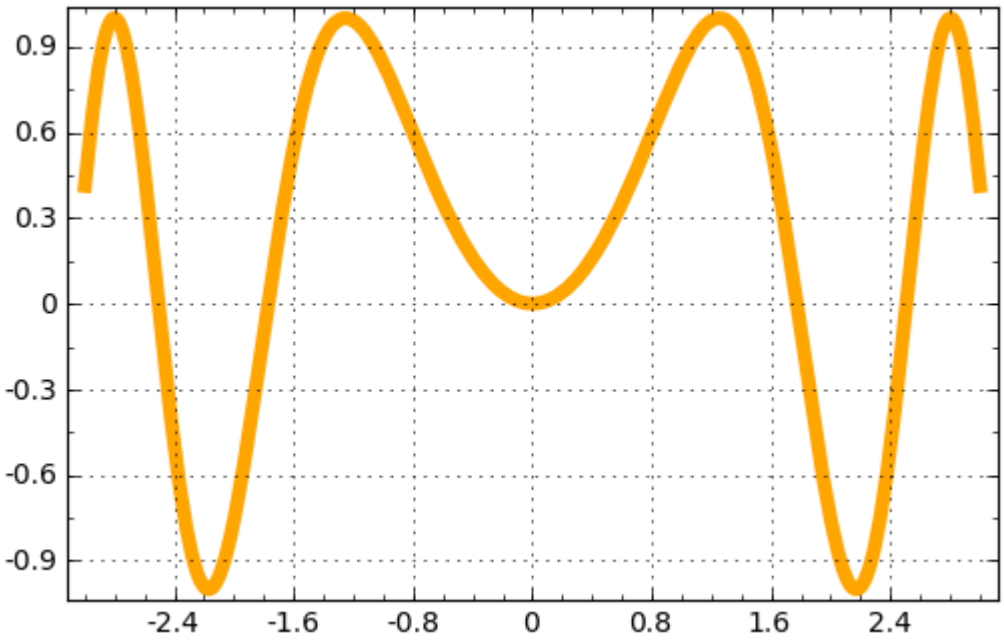
image.pdf

```
P.save('image.eps')
```

image.eps

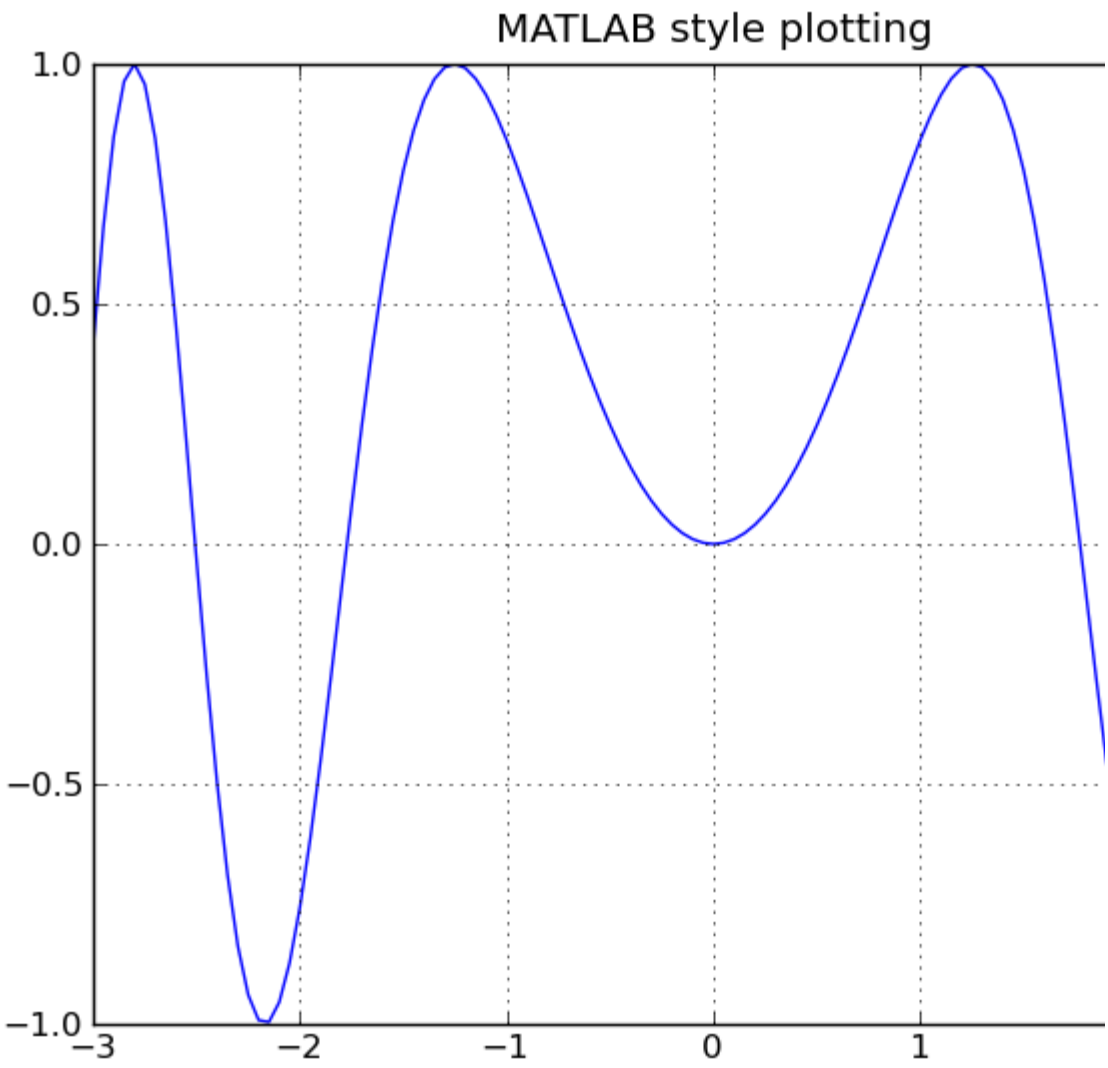
The plot command has many options:

```
plot(sin(x^2), (x,-3,3), thickness=5, color='orange',
gridlines=True, frame=True, axes=False)
```



Note for MATLAB Users: You can also use a full emulation of matlab's plotting system via pylab (which is part of the standard matploblib Python library):

```
from pylab import *
clf()
t = arange(-3.0, 3.0, 0.05)
s = sin(t*t)
P = plot(t, s, linewidth=1.0)
t = title('MATLAB style plotting')
grid(True)
savefig('sage.png')
reset()  # get rid of effect of "from pylab import *"
```



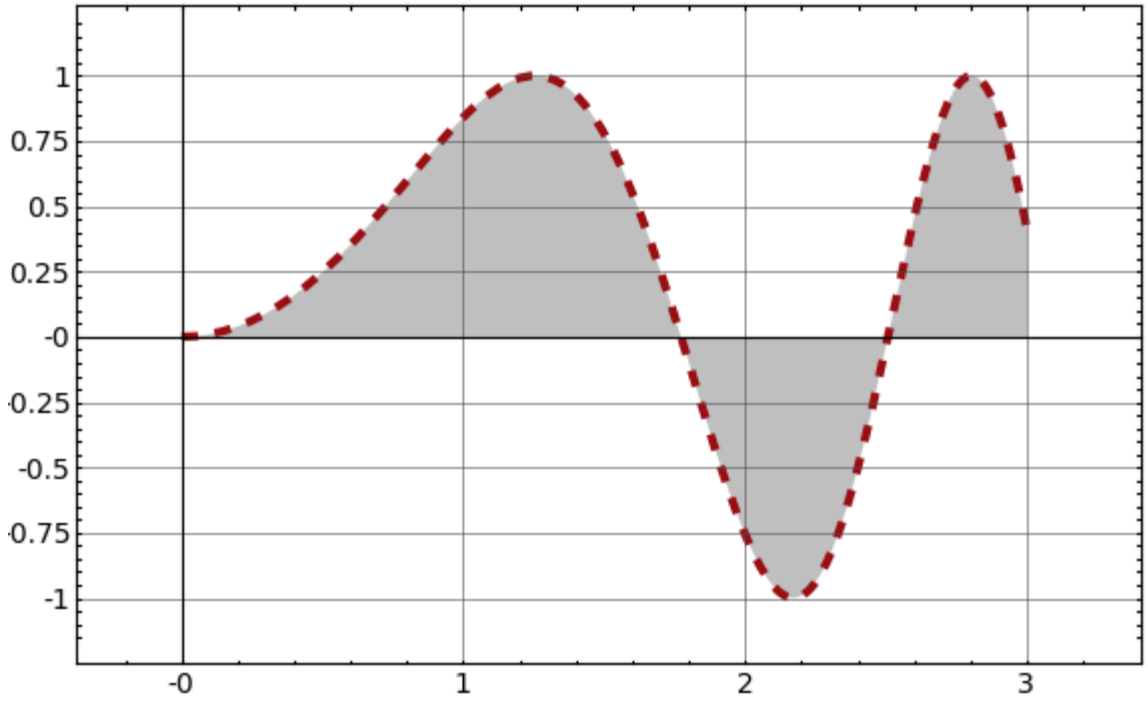
Problem: If you know MATLAB, try testing some simple 2d plotting.

Processing Math: Done

Here's a the same plot, but you can adjust many of the parameters to the plot command interactively.

```
var('x')
@interact
def
plot_example(f=sin(x^2),r=range_slider(-5,5,step_size=1/4,default=
(-3,3)),
                  color=color_selector(widget='colorpicker'),
                  thickness=(3,(1..10)),
                  adaptive_recursion=(5,(0..10)),
adaptive_tolerance=(0.01,(0.001,1)),
                  plot_points=(20,(1..100)),
                  linestyle=['-','--','-.',':'],
                  gridlines=False, fill=False,
                  frame=False, axes=True
                  ):
    show(plot(f, (x,r[0],r[1]), color=color,
thickness=thickness,
                  adaptive_recursion=adaptive_recursion,
                  adaptive_tolerance=adaptive_tolerance,
plot_points=plot_points,
                  linestyle=linestyle, fill=fill if fill else
None),
                  gridlines=gridlines, frame=frame, axes=axes)
```

Problem: Use the above interactive plotter to draw the following plot of $\sin(x^2)$.

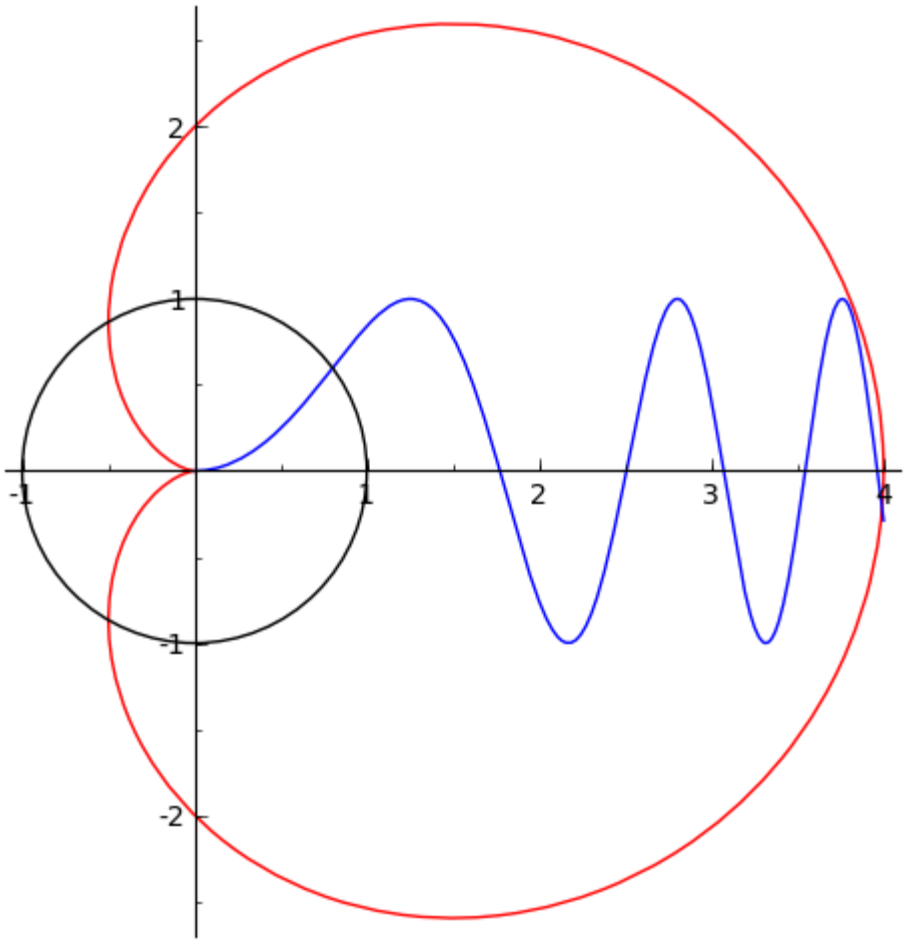


You can plot many other things, including polygons, parametric plots, polar plots, implicit plots, etc.:

line, polygon, circle, text, polar_plot, parametric_plot, circle, implicit_plot

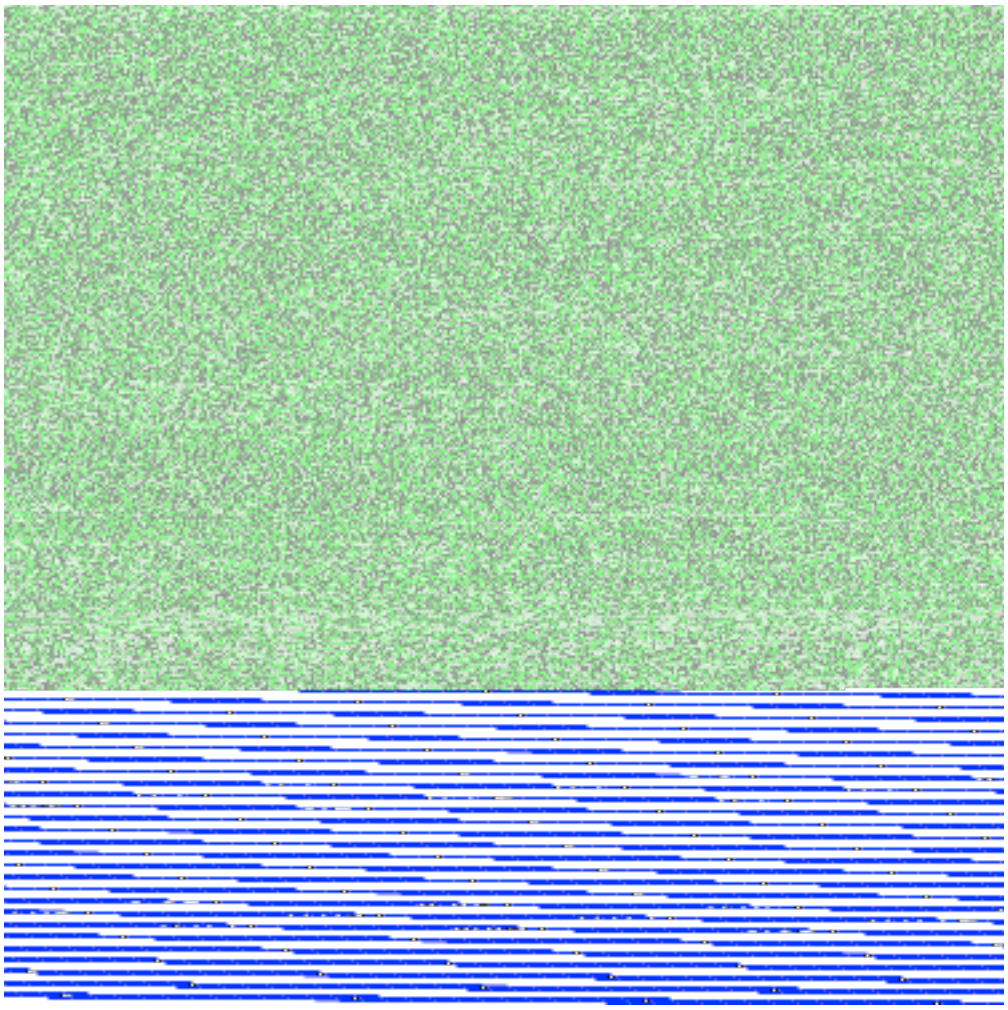
You superimpose plots using **+**.

```
var('x')
P = circle((0,0),1) + polar_plot(2 + 2*cos(x), (x, 0, 2*pi),
rgbcolor='red')+ plot(sin(x^2),(x,0,4))
show(P, aspect_ratio=1)
```



You can also do 3D plots, including parametric lines, parametric surfaces, implicit 3d plots, etc.

```
var('x y')
plot3d(sin(pi*(x^2+y^2))/2,(x,-1,1),(y,-1,1), opacity=.7) +
octahedron(color='red')
```



Get Image

More Symbolic Calculus: Computing Integrals and Derivatives

You can symbolically integrate or differentiate functions, compute limits, Taylor polynomials, etc.

```
var('x')
integrate(x^2, x)
```

```
1/3*x^3
```

```
integrate(sin(x)+tan(2*x),x)
```

```
1/2*log(sec(2*x)) - cos(x)
```

```
integrate(sin(x)+tan(2*x),x, algorithm='sympy')
```

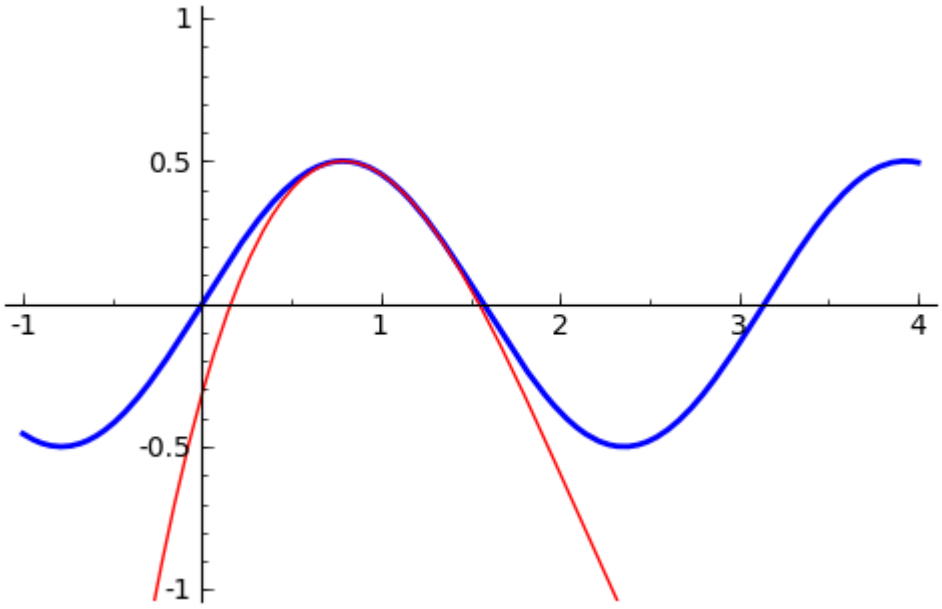
```
1/4*log(tan(2*x)^2 + 1) - cos(x)
```

```
f = sin(x) - cos(y*x) + 1/(x^3+1)
g = f.integrate(x)
show(g)
```

$$\frac{1}{3}\sqrt{3}\arctan\left(\frac{1}{3}(2x-1)\sqrt{3}\right)+\frac{-\sin(xy)}{y}+\frac{1}{3}\log(x+1)-\frac{1}{6}\log\left(x^2-x+\right.$$

```
bool(g.differentiate(x) == f)
```

```
True
```



Symbolic Taylor Series

```
f = sin(x^2)-cos(x)/log(x)
```

```
f.taylor(x, 1, 3)
```

```
    -7/720*(x - 1)^3*(210*sin(1) + 143*cos(1)) - 1/24*(x -
    1)^2*(54*sin(1) - 29*cos(1)) + 1/12*(x - 1)*(6*sin(1) + 31*cos(
    cos(1)/(x - 1) + 2*sin(1) - 1/2*cos(1)
```

```
g = sin(x^2)/x
```

```
g.taylor(x, 0, 20)
```

```
    1/362880*x^17 - 1/5040*x^13 + 1/120*x^9 - 1/6*x^5 + x
```

```
@interact
def ex_taylor(n=(1..10)):
    h(x) = sin(x)*cos(x)
    show(h)
    show(plot(h,-1,4,thickness=2) + plot(h.taylor(x,1,n),-1,4,
color='red'), ymin=-1,ymax=1)
```

```

```

```

```

Other Options for Symbolic Calculus

1. ***Maxima*** -- a powerful symbolic computer algebra system, which is fully usable from Sage, and included in Sage. Sometimes you want to do something, e.g., solve some weird differential equation or evaluate a special functions, and you search the web, find how to do it in Maxima, and can then... paste and do the same in Sage via the Maxima interface.

2. ***SymPy*** -- a powerful pure Python library for symbolic Calculus, written almost entirely by Physics graduate students. This can be installed into anything that you can run Python on, and provides a nice range of capabilities. Sage uses it some, and it is included in Sage.

Maxima

```
maxima('sin(%i)')
```

```
    %i*sinh(1)
```

```
sin(I)
```

```
    sin(I)
```

```
f = log(sin(x)/x)
```

```
f.taylor(x, 0, 10)
```

```
    -1/467775*x^10 - 1/37800*x^8 - 1/2835*x^6 - 1/180*x^4 - 1/6*x^2
```

```
[bernoulli(2*i) for i in range(1,7)]
[1/6, -1/30, 1/42, -1/30, 5/66, -691/2730]
```

```
f_maxima = maxima(f)
f_maxima
```

```
    log(sin(x)/x)
```

```
type(f_maxima)
```

```
    <class 'sage.interfaces.maxima.MaximaElement'>
```

```
parent(f_maxima)
```

```
    Maxima
```

```
maxima(f).powerseries(x,0)
```

```
    'sum((-1)^i3*2^(2*i3-1)*bern(2*i3)*x^(2*i3)/(i3*(2*i3)!),i3,1,i
```

```
```

Sympy

```
reset()
```

```
from sympy import var as svar, sin, integrate, pi, S
```

```
x = svar('x')
integrate(sin(x)*cos(x+1), (x, 0, pi))
```

```
    -pi*sin(1)/2
```

```
```

Mathematica and Maple

The following won't work for you during the tutorial. It illustrates Sage's powerful ***interfaces***.

```
mathematica('Integrate[Sin[x]*Cos[x+1],x]')
```

```
    -Cos[1 + 2*x]/4 - (x*Sin[1])/2
```

```
f = mathematica(sin(x)*cos(x+1)); f
```

```
    Cos[1 + x]*Sin[x]
```

```
f.Integrate(x)
```

```
    -Cos[1 + 2*x]/4 - (x*Sin[1])/2
```

```
maple('integrate(sin(x)*cos(x+1),x)')
```

```
    -1/4*cos(2*x+1)-1/2*sin(1)*x
```