Sage应用在代数领域

山东理工大学数学院 周世祥

置换群：

to create a permutation group, give a list of generators, as in the following example.

In [2]:

```
G = PermutationGroup(['(1,2,3)(4,5)', '(3,4)'])
G
```

Out[2]:

```
Permutation Group with generators [(3,4), (1,2,3)(4,5)]
```

In [3]:

```
G.order()
```

Out[3]:

```
120
```

In [4]:

```
G.is_abelian()
```

Out[4]:

```
False
```

In [5]:

```
G.derived_series()           # random-ish output
```

Out[5]:

```
[Subgroup of (Permutation Group with generators [(3,4), (1,2,3)(4,5)]) generated b
y [(3,4), (1,2,3)(4,5)],
 Subgroup of (Permutation Group with generators [(3,4), (1,2,3)(4,5)]) generated b
y [(1,5,3), (1,5)(3,4), (1,5)(2,4)]]
```

In [6]:

```
G.center()
```

Out[6]:

```
Subgroup of (Permutation Group with generators [(3,4), (1,2,3)(4,5)]) generated by
[()]
```

In [7]:

```
G.random_element()          # random output
```

Out[7]:

```
(1,3,2)
```

In [8]:

```
print(latex(G))
```

```
\langle (3,4), (1,2,3)(4,5) \rangle
```

$$\langle (3,4),(1,2,3)(4,5)\rangle$$

obtain the character table (in LaTeX format) in Sage:

In [9]:

```
G = PermutationGroup([[(1,2),(3,4)], [(1,2,3)]])
latex(G.character_table())
```

Out[9]:

```
\left(\begin{array}{rrrr}
1 & 1 & 1 & 1 \\
1 & -\zeta_{3} - 1 & \zeta_{3} & 1 \\
1 & \zeta_{3} & -\zeta_{3} - 1 & 1 \\
3 & 0 & 0 & -1
\end{array}\right)
```

$$\left(\begin{array}{rrrr}
1 & 1 & 1 & 1 \\
1 & -\zeta_{3} - 1 & \zeta_{3} & 1 \\
1 & \zeta_{3} & -\zeta_{3} - 1 & 1 \\
3 & 0 & 0 & -1
\end{array}\right)$$

includes classical and matrix groups over finite fields:

In [10]:

```
MS = MatrixSpace(GF(7), 2)
gens = [MS([[1,0],[-1,1]]),MS([[1,1],[0,1]])]
G = MatrixGroup(gens)
G.conjugacy_classes_representatives()
```

Out[10]:

```
(
[1 0]  [0 6]  [0 4]  [6 0]  [0 6]  [0 4]  [0 6]  [0 6]  [0 6]  [4 0]
[0 1], [1 5], [5 5], [0 6], [1 2], [5 2], [1 0], [1 4], [1 3], [0 2],

[5 0]
[0 3]
)
```

In [11]:

```
G = Sp(4,GF(7))
G
```

Out[11]:

Symplectic Group of degree 4 over Finite Field of size 7

In [12]:

```
G.random_element()             # random output
G.order()
```

Out[12]:

276595200

In [13]:

```
# You can also compute using abelian groups (infinite and finite):
F = AbelianGroup(5, [5,5,7,8,9], names='abcde')
(a, b, c, d, e) = F.gens()
```

In [14]:

```
d * b**2 * c**3
```

Out[14]:

b^2*c^3*d

In [15]:

```
F = AbelianGroup(3,[2]*3); F
```

Out[15]:

Multiplicative Abelian group isomorphic to C2 x C2 x C2

In [16]:

```
H = AbelianGroup([2,3], names="xy"); H
```

Out[16]:

Multiplicative Abelian group isomorphic to C2 x C3

In [17]:

```
AbelianGroup(5)
```

Out[17]:

Multiplicative Abelian group isomorphic to Z x Z x Z x Z x Z

In [18]:

```
AbelianGroup(5).order()
```

Out[18]:

+Infinity

In [21]:

```
#3n+1问题
n = 2005
for i in range(1000):
    n = 3*odd_part(n) + 1
    if odd_part(n)==1:
        print(i)
        break
```

38

In [87]:

```


x=var("x")
y=var("y")

f(x,y)=(x+y)^3-(x^3+3*x^2*y+3*x*y^2+y^3)
f(10000,0.0001)
```

Out[87]:

-0.000177929687500000

In [88]:

```
f(10000,1/10000)
```

Out[88]:

0

In [31]:

```
x,y=10000,0.0001
z=(x+y)^3-(x^3+3*x^2*y+3*x*y^2+y^3)
z
```

Out[31]:

-0.000122070312500000

In [33]:

```
x,y=10000,1/10000
z=(x+y)^3-(x^3+3*x^2*y+3*x*y^2+y^3)
z
```

Out[33]:

0

In [36]:

```
#数独游戏
sudoku?
```

In [39]:

```
# 精确求解方程
x = var('x')
solve(x^2 + 3*x + 2, x)
```

Out[39]:

```
[x == -2, x == -1]
```

In [40]:

```
x, b, c = var('x b c')
solve([x^2 + b*x + c == 0],x)
```

Out[40]:

```
[x == -1/2*b - 1/2*sqrt(b^2 - 4*c), x == -1/2*b + 1/2*sqrt(b^2 - 4*c)]
```

In [41]:

```
# 也可以求解多变量的方程（组）:
x, y = var('x, y')
solve([x+y==6, x-y==4], x, y)
```

Out[41]:

```
[[x == 5, y == 1]]
```

In [44]:

```
#Sage 求解非线性方程组的例子。我们先求方程组的符号解
var('x y p q')

eq1 = p+q==9
eq2 = q*y+p*x==-6
eq3 = q*y^2+p*x^2==24
solve([eq1,eq2,eq3,p==1],p,q,x,y)
```

Out[44]:

```
[[p == 1, q == 8, x == -4/3*sqrt(10) - 2/3, y == 1/6*sqrt(10) - 2/3], [p == 1, q =
= 8, x == 4/3*sqrt(10) - 2/3, y == -1/6*sqrt(10) - 2/3]]
```

In [46]:

```
# 要求解的近似值，可以这样:
solns = solve([eq1,eq2,eq3,p==1],p,q,x,y, solution_dict=True)
[[s[p].n(30), s[q].n(30), s[x].n(30), s[y].n(30)] for s in solns]

#函数 n 输出数值近似值，参数是以 bit 为单位的结果精度
```

Out[46]:

```
[[1.0000000, 8.0000000, -4.8830369, -0.13962039],
 [1.0000000, 8.0000000, 3.5497035, -1.1937129]]
```

In [48]:

```
# 但是我们可以用 find root 在区间 0 < ϕ < π/2 上寻找上述方程的解。
phi = var('phi')
find_root(cos(phi)==sin(phi),0,pi/2)
```

Out[48]:

0.7853981633974484

In [49]:

```
# 微积分
x, y = var('x,y')
f = x^2 + 17*y^2
f.diff(x)
f.diff(y)
```

Out[49]:

34*y

In [50]:

```
#计算 sin(x^2) 的 4 阶微分:
diff(sin(x^2), x, 4)
```

Out[50]:

16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)

In [52]:

```
# 再来看积分，定积分、不定积分都可以计算
integral(x*sin(x^2), x)
```

Out[52]:

-1/2*cos(x^2)

In [54]:

```
integral(x/(x^2+1), x, 0, 1)
```

Out[54]:

1/2*log(2)

In [56]:

```
#部分分式分解
f = 1/((1+x)*(x-1))
f.partial_fraction(x)
```

Out[56]:

-1/2/(x + 1) + 1/2/(x - 1)

In [57]:

```
#可以用 Sage 求解常微分方程组

t = var('t') # 定义变量 t
x = function('x',t) # 定义 x 是变量 t 的函数
DE = diff(x, t) + x - 1
desolve(DE, [x,t])
```

```
/opt/sagemath-8.2/local/lib/python2.7/site-packages/IPython/core/interactiveshell.
py:2882: DeprecationWarning: Calling function('f',x) is deprecated. Use function
('f')(x) instead.
See http://trac.sagemath.org/17447 for details.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[57]:

```
(_C + e^t)*e^(-t)
```

In [58]:

```
#计算 Laplace 变换
s = var("s")
t = var("t")
f = t^2*exp(t) - sin(t)
f.laplace(t,s)
```

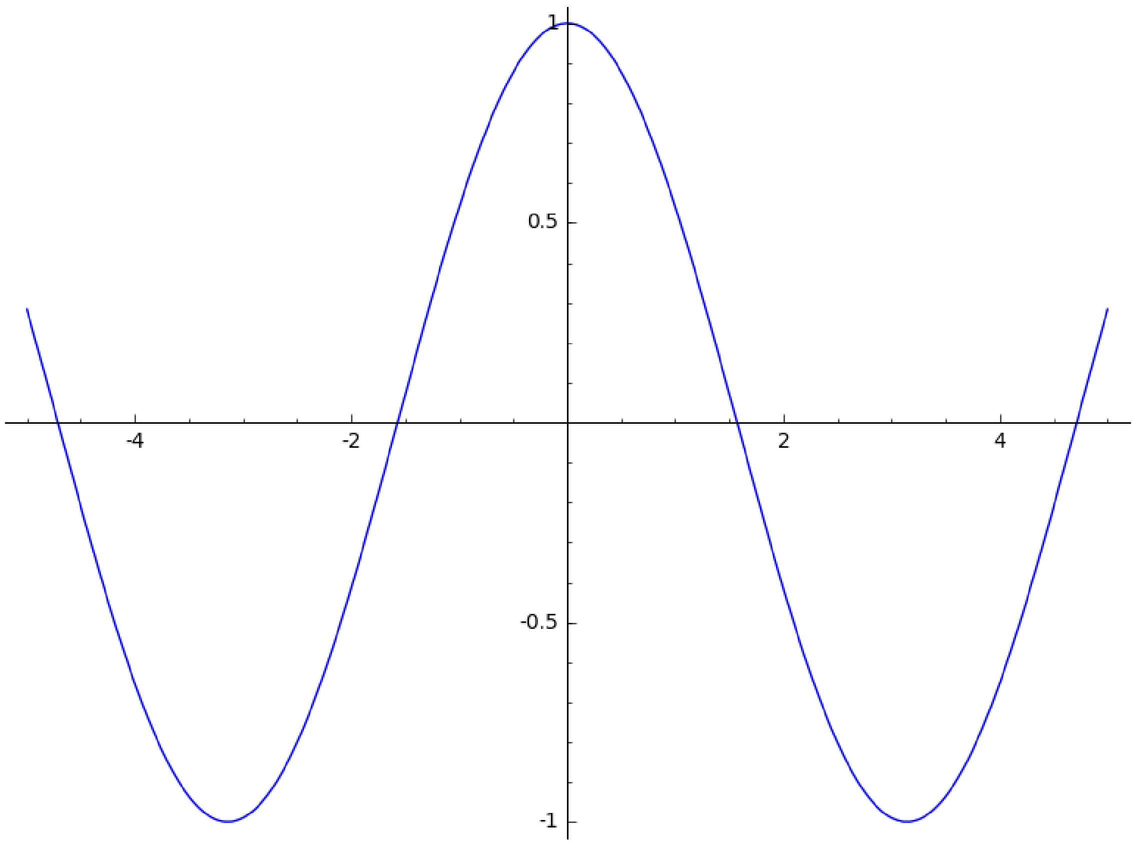
Out[58]:

```
-1/(s^2 + 1) + 2/(s - 1)^3
```

In [60]:

```
#绘制基本函数的图像是非常容易
plot(cos, (-5,5))
```

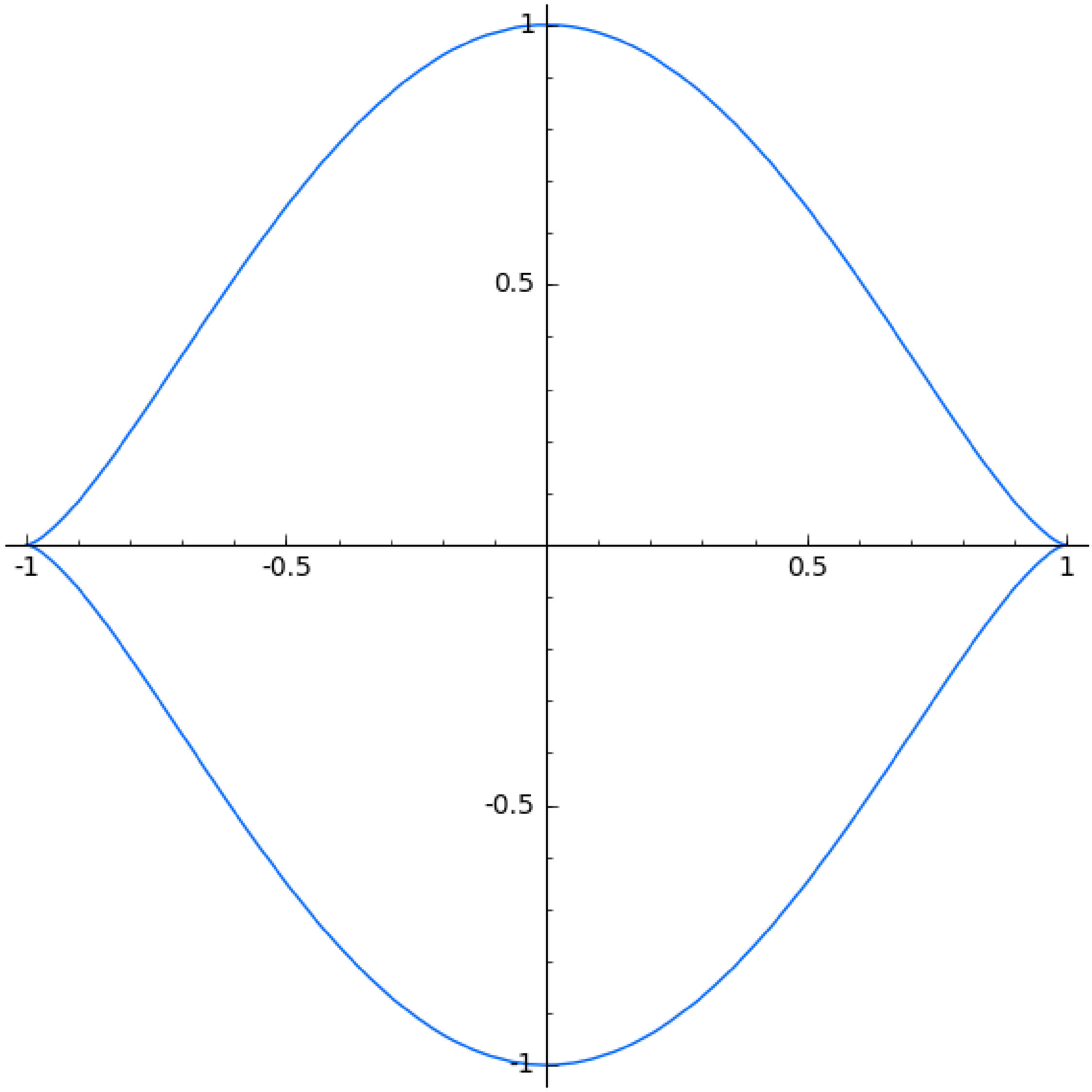
Out[60]:



In [62]:

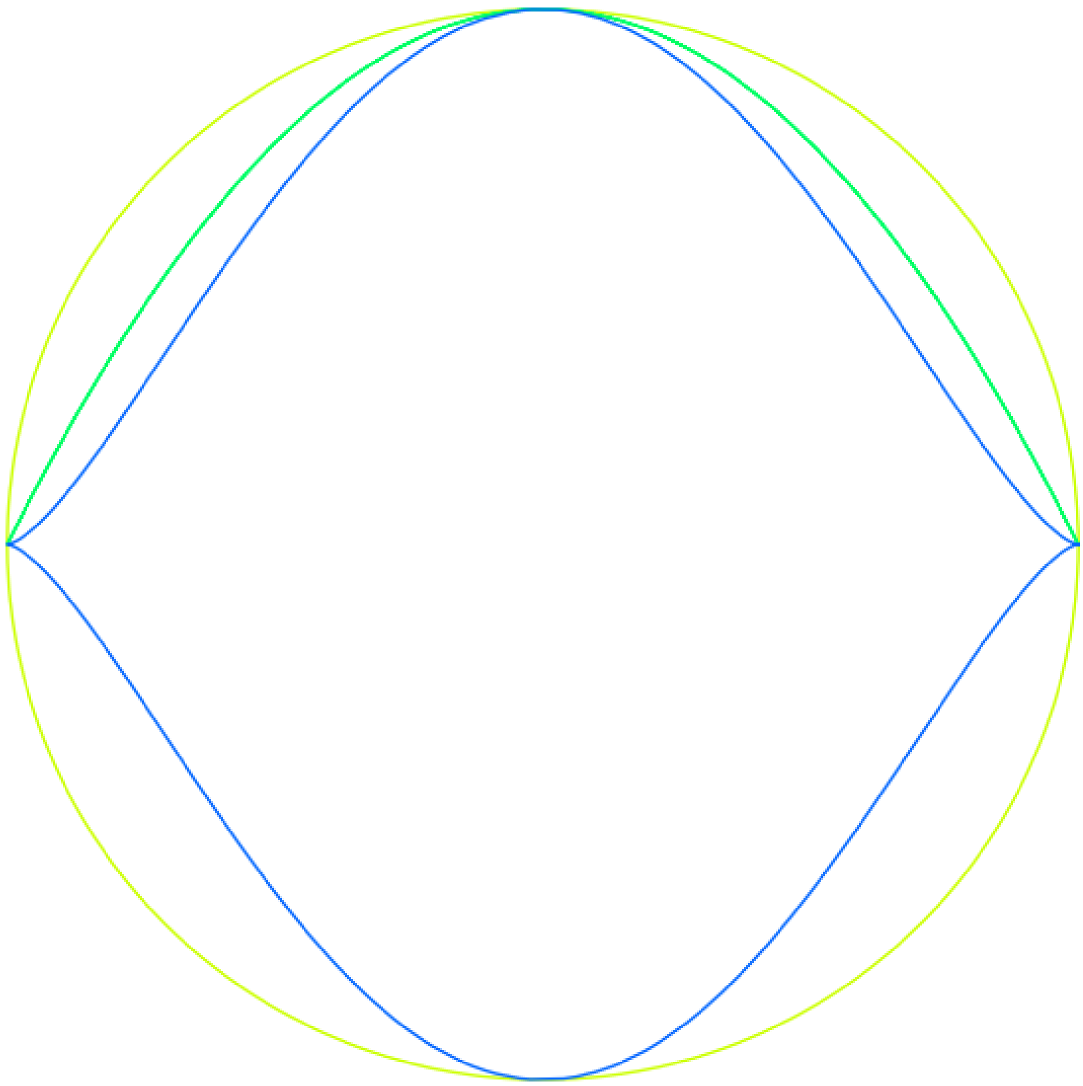
```
# 一旦指定了变量名，就可以绘制参数方程的图像
x = var('x')
parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
```

Out[62]:



In [64]:

```
#你可以把多个图像组合在一起:
x = var('x')
p1 = parametric_plot((cos(x),sin(x)),(x,0,2*pi),rgbcolor=hue(0.2))
p2 = parametric_plot((cos(x),sin(x)^2),(x,0,2*pi),rgbcolor=hue(0.4))
p3 = parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
show(p1+p2+p3, axes=false)
```



In [65]:

```
#使用 plot3d 绘制形如 f(x, y) = z 的函数图像：
x, y = var('x,y')
plot3d(x^2 + y^2, (x,-2,2), (y,-2,2))
```

Out[65]:

In [67]:

```
# 函数
def f(z): return z^2
type(f)
```

Out[67]:

<type 'function'>

In [68]:

```
f(3)
```

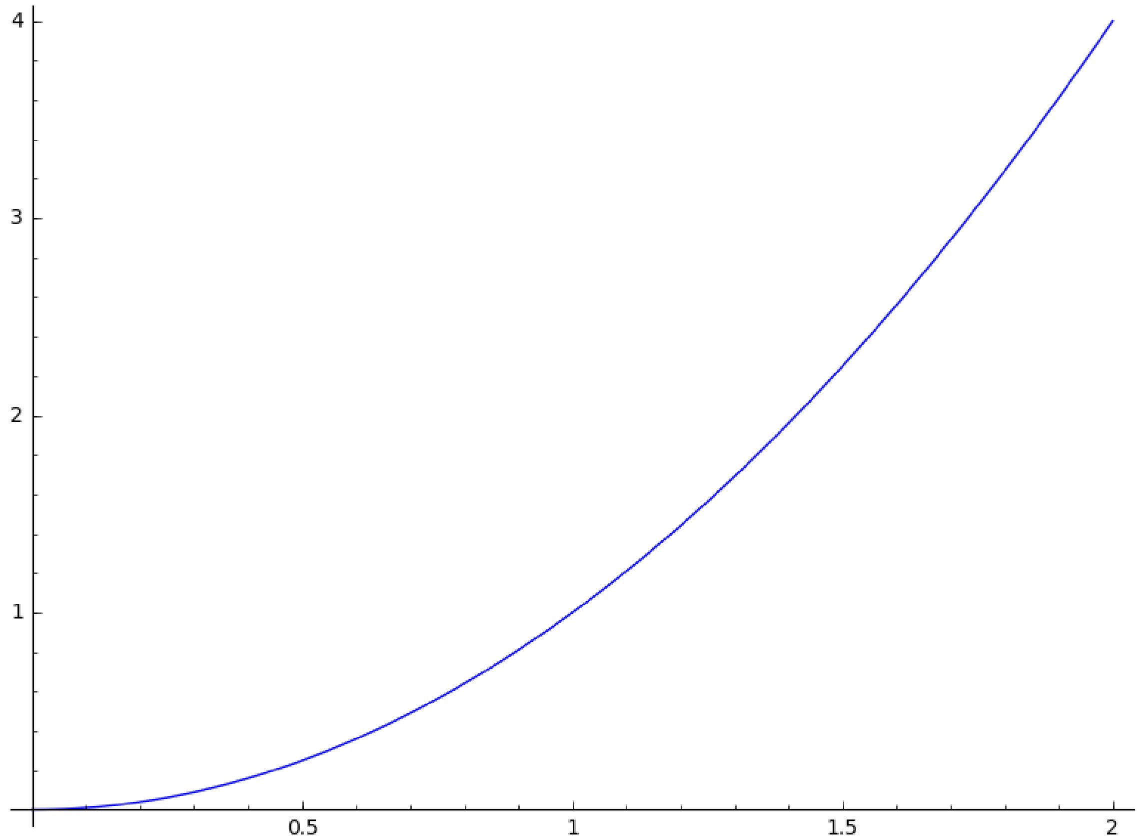
Out[68]:

9

In [69]:

```
plot(f, 0, 2)
```

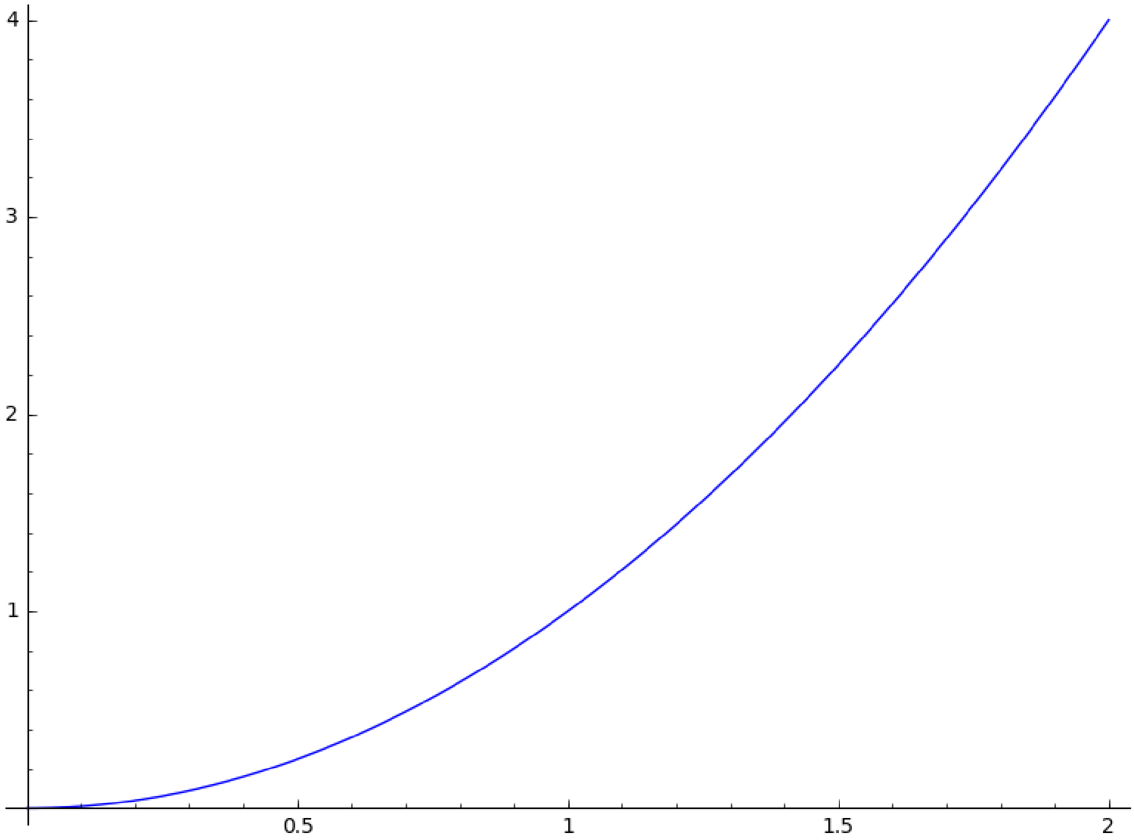
Out[69]:



In [70]:

```
var('z') # 定义 z 为一个变量
f(z)
plot(f(z), 0, 2)
# 这里, f(x) 是一个符号表达式，这是
```

Out[70]:



In [72]:

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

In [73]:

```
g
```

Out[73]:

```
x |--> x^2
```

In [74]:

```
g(3)
```

Out[74]:

9

In [75]:

```
Dg = g.derivative(); Dg
```

Out[75]:

x |--> 2*x

In [76]:

```
type(g)
```

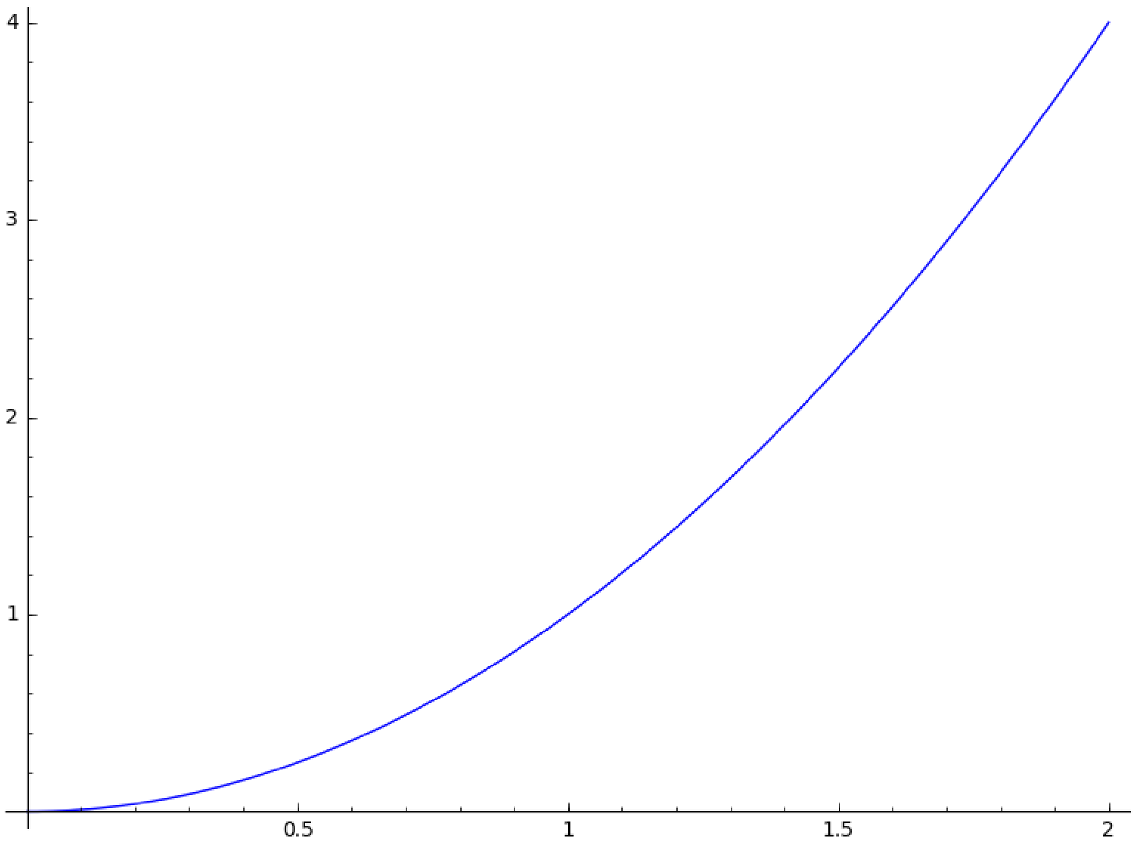
Out[76]:

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

In [77]:

```
plot(g, 0, 2)
# 注意 g 是可调用的符号表达式，g(x) 是一个与之相关，但是不同类型的对象，虽然 g(x) 也可用于绘图、
```

Out[77]:



In [78]:

```
g(x)
```

Out[78]:

x^2

In [79]:

```
type(g(x))
```

Out[79]:

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

In [80]:

```
g(x).derivative()
```

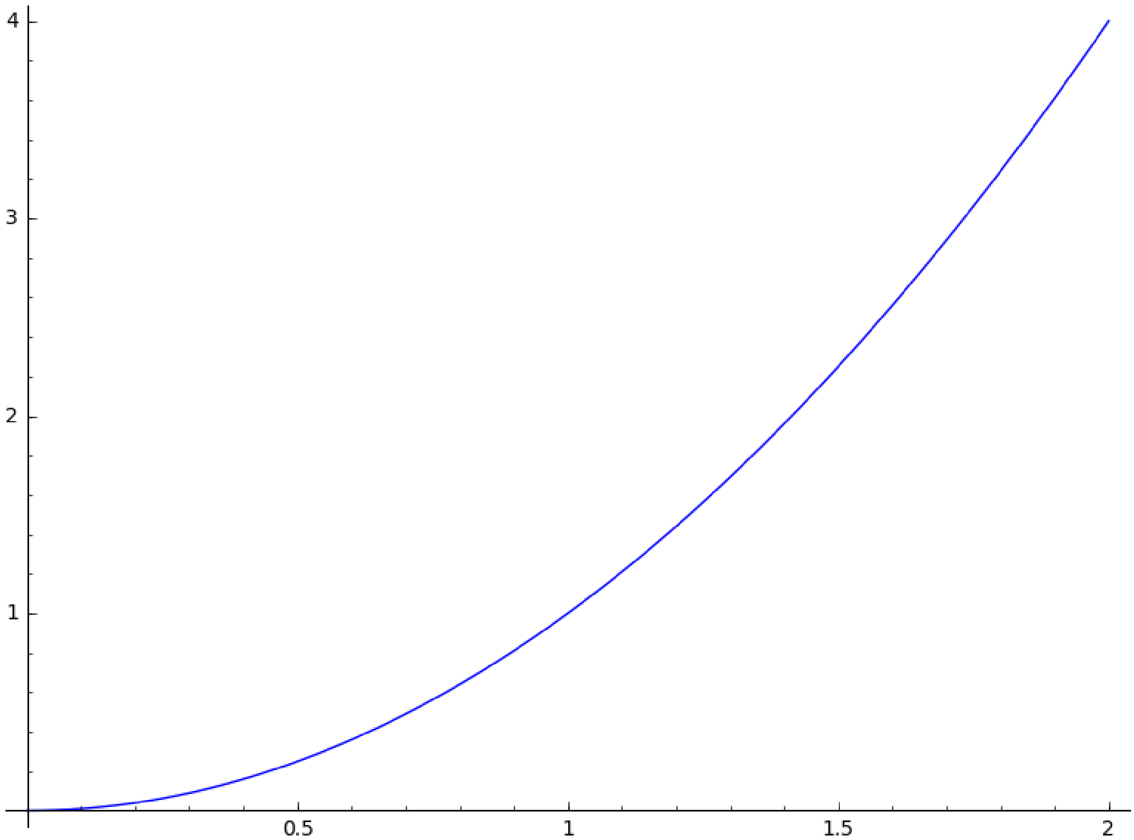
Out[80]:

2*x

In [81]:

```
plot(g(x), 0, 2)
```

Out[81]:



In [82]:

```
type(sin)
```

Out[82]:

<class 'sage.functions.trig.Function_sin'>

In [83]:

```
type(sin(x))
```

Out[83]:

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

In [85]:

```
x, y, z = var('x,y,z')
f(x, y, z) = 4*x^2 * (x^2 + y^2 + z^2 + z) + y^2 * (y^2 + z^2 - 1)
implicit_plot3d(f, (x, -0.5, 0.5), (y, -1, 1), (z, -1, 1))
```

Out[85]:

基本的环

当定义矩阵、向量或者多项式时，有时候需要，有时候是必须要指定其所在的“环”。环 是一个数学结 构，在其上定义的加法和乘法有很好的性质。如果你从来没有听说过这些概念，至少要了解以下这四个常用 的环：

• 整数环 {..., -1, 0, 1, 2, ...}, Sage 中叫 ZZ;

• 有理数环Q即，整数构成的分数，Sage 中叫 QQ;

• 实数环，Sage 中叫 RR;

• 复数环，Sage 中叫 CC.

你需要了解它们之间的区别，因为对于同一个多项式，处理方式完全取决于它定义在哪个环上。比如说， 多项式 x^2-2 有两个根, $\sqrt{2},-\sqrt{2}$,这些根不是有理数，所以如果你是讨论有理系数多项式，那么该多项式不能进行因式分解，如果是实系数，就可以。所以你可能要指定环以保证得到的结果是你所期望的。下面两个命 令定义有理系数多项式和实系数多项式集合。集合的名字分别是“ratpoly”和“realpoly”，这两个名字并不 重要，但是要注意变量的名字“.”和“.”的使用。

```
ratpoly.<t> = PolynomialRing(QQ)
realpoly.<z> = PolynomialRing(RR)
```

In [93]:

```
ratpoly.<t> = PolynomialRing(QQ)
factor(t^2-2)
```

Out[93]:

```
t^2 - 2
```

In [94]:

```
realpoly.<z> = PolynomialRing(RR)
factor(z^2-2)
```

Out[94]:

```
(z - 1.41421356237310) * (z + 1.41421356237310)
```

对于矩阵存在同样的问题：矩阵的行消去形式取决于其所定义的环，还有它的特征值和特征向量的计 算。更多关于多项式构造的内容请参见: 多项式, 更多关于矩阵的内容请参见线性代数. 符号 I 代表 1 的平方根; i 等同于 I. 当然这不是一个有理数

In [95]:

```
i
```

Out[95]:

```
38
```

In [96]:

```
reset('i')
```

In [97]:

```
i
```

Out[97]:

```
I
```

In [98]:

```
i in QQ
```

Out[98]:

```
False
```

下面是关于 Sage 中基本环的几个例子。上面已经提到有理数环可以用 QQ, 也可以用 RationalField() (域 是环的一种，乘法是可交换的，且所有非零元素均有乘法逆元。所有有理数可以构成一个域，但是整数不 行)

In [99]:

```
RationalField()
```

Out[99]:

```
Rational Field
```

In [100]:

```
QQ
```

Out[100]:

```
Rational Field
```

In [101]:

```
1/2 in QQ
```

Out[101]:

```
True
```

In [102]:

```
1.2 in QQ
```

Out[102]:

```
True
```

In [103]:

```
pi in QQ
```

Out[103]:

```
False
```

In [104]:

```
pi in RR
```

Out[104]:

```
True
```

In [105]:

```
sqrt(2) in QQ
```

Out[105]:

```
False
```

In [106]:

```
sqrt(2) in CC
```

Out[106]:

```
True
```

为用于高等数学，Sage 还知道其他的环，比如有限域, p-adic 整数，代数数环，多项式环和矩阵环。下面 是其中一些环的构造:

In [107]:

```
GF(3)
```

Out[107]:

```
Finite Field of size 3
```

In [108]:

```
GF(27, 'a') # need to name the generator if not a prime field
```

Out[108]:

```
Finite Field in a of size 3^3
```

线性代数 Sage 提供线性代数的标准构造，如矩阵的特征多项式，梯形格式，迹，分解等。 构造矩阵和矩阵的乘法都是很容易的，也是很自然的

In [109]:

```
A = Matrix([[1,2,3],[3,2,1],[1,1,1]])
w = vector([1,1,-4])
w*A
```

Out[109]:

```
(0, 0, 0)
```

In [110]:

```
A*w
```

Out[110]:

```
(-9, 1, -2)
```

基本的环 中提到，矩阵所在的环影响它的性质。下面 matrix 命令中的第一个参数告诉 Sage 这个矩阵 是整数环 (ZZ) 上的，有理数环 (QQ) 上的，还是实数环 (RR) 上的:

In [112]:

```
AZ = matrix(ZZ, [[2,0], [0,1]])
AZ
```

Out[112]:

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

In [115]:

```
AZ.echelon_form()
```

Out[115]:

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

In [113]:

```
AQ = matrix(QQ, [[2,0], [0,1]])
AQ
```

Out[113]:

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

In [116]:

```
AR = matrix(RR, [[2,0], [0,1]])
AR.echelon_form()
```

Out[116]:

```
[ 1.00000000000000 0.000000000000000]
[0.000000000000000  1.00000000000000]
```

一元多项式

有三种方法创建多项式环。

In [117]:

```
R = PolynomialRing(QQ, 't')
R
```

Out[117]:

```
Univariate Polynomial Ring in t over Rational Field
```

建立一个多项式环并告诉 Sage 在输出到屏幕时，使用字符 't' 作为不确定的量。但是这种方法没有定义 符号 t 在 Sage 中如何使用，你不能用它输入一个属于 R 的多项式 (如t^2+1)。 另一种方法是

In [118]:

```
S = QQ['t']
S == R
```

Out[118]:

```
True
```

这里的 t 有同样的问题。 第三种非常方便的方法是

In [119]:

```
R.<t> = PolynomialRing(QQ)
```

In [120]:

```
R.<t> = QQ['t']
```

In [121]:

```
# 或者，甚至是
R.<t> = QQ[]
```

这样变量 t 定义为多项式环的不定量，你可以很容易构造 R 中的元素，象下面一样。（注意，第三种方 式与 Magma 中的构造方法类似，但是在 Magma 中，可以用这种方法定义的对象有很多。）

In [122]:

```
poly = (t+1) * (t+2); poly
```

Out[122]:

```
t^2 + 3*t + 2
```

In [124]:

```
poly in R
```

Out[124]:

```
True
```

In [126]:

```
f = 2*t^7 + 3*t^2 - 15/19
f^2
```

Out[126]:

```
4*t^14 + 12*t^9 - 60/19*t^7 + 9*t^4 - 90/19*t^2 + 225/361
```

In [129]:

```
# 两个多项式相除将产生一个分式域中的元素（由 Sage 自动创建）。
x = QQ['x'].0
f = x^3 + 1; g = x^2 - 17
h = f/g; h
```

Out[129]:

```
(x^3 + 1)/(x^2 - 17)
```

In [130]:

```
h.parent()
```

Out[130]:

```
Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

多项式扩展的欧几里得算法

In [174]:

```
R.<x> = PolynomialRing(QQ)
A=x^8+x^6-3*x^4-3*x^3+8*x^2+2*x-5
B=3*x^6+5*x^4-4*x^2-9*x+21
D,U,V=xgcd(A,B) # 多项式环与整数环Z的性质类似
D
```

Out[174]:

```
1
```

In [175]:

```
U
```

Out[175]:

```
13989/130354*x^5 + 9225/65177*x^4 + 20281/65177*x^3 + 67125/130354*x^2 + 5149/1303
54*x - 1391/18622
```

In [176]:

```
V
```

Out[176]:

```
-4663/130354*x^7 - 3075/65177*x^6 - 5206/65177*x^5 - 18275/130354*x^4 + 4944/65177
*x^3 + 21579/130354*x^2 + 1910/65177*x + 3889/130354
```

虽然初始数据不大，但运算过程中膨胀的很快，跟计算方法有很大的关系，如果把有理数整数化，需要研究大整数运算性质。

In [131]:

```
# 多元多项式
# 要使用多元多项式，先要声明多项式环和变量。
R = PolynomialRing(GF(5),3,"z") # here, 3 = number of variables
R
```

Out[131]:

Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5

In [132]:

```
# 跟定义一元多项式一样，有多种方法：
GF(5)['z0, z1, z2']
```

Out[132]:

Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5

In [133]:

```
# 如果你希望变量的名字是单个字母，可以用下面的简短形式：
PolynomialRing(GF(5), 3, 'xyz')
```

Out[133]:

Multivariate Polynomial Ring in x, y, z over Finite Field of size 5

In [134]:

```
z = GF(5)['z0, z1, z2'].gens()
z
```

Out[134]:

(z0, z1, z2)

In [136]:

```
(z[0]+z[1]+z[2])^2
```

Out[136]:

z0^2 + 2*z0*z1 + z1^2 + 2*z0*z2 + 2*z1*z2 + z2^2

In [140]:

```
R, (x, y) = PolynomialRing(RationalField(), 2, 'xy').objgens()
R
```

Out[140]:

Multivariate Polynomial Ring in x, y over Rational Field

In [142]:

```
f = (x^3 + 2*y^2*x)^2
g = x^2*y^2
f.gcd(g)
```

Out[142]:

x^2

有限群，阿贝尔群

In [143]:

```
G = PermutationGroup(['(1,2,3)(4,5)', '(3,4)'])
G
```

Out[143]:

Permutation Group with generators [(3,4), (1,2,3)(4,5)]

In [144]:

```
G.order()
```

Out[144]:

120

In [146]:

```
G.is_abelian()
```

Out[146]:

False

In [147]:

```
G.random_element() # random output
```

Out[147]:

(1,4,3)

Sage 对数论有广泛的支持。例如我们可以在 Z/NZ 上进行运算：

In [149]:

```
R = IntegerModRing(97)
a = R(2) / R(3)
a
```

Out[149]:

33

In [150]:

```
b = R(47)
```

In [152]:

```
b^20052005
```

Out[152]:

50

In [153]:

```
b.modulus()
```

Out[153]:

97

In [155]:

```
b.is_square()
```

Out[155]:

True

In [156]:

```
# Sage 包含标准的数论函数。如:
gcd(515,2005)
```

Out[156]:

5

In [157]:

```
factor(2005)
```

Out[157]:

5 * 401

In [158]:

```

c = factorial(25); c
```

Out[158]:

15511210043330985984000000

In [159]:

```
next_prime(2005)
```

Out[159]:

2011

In [160]:

```
previous_prime(2005)
```

Out[160]:

2003

In [161]:

```
divisors(28); sum(divisors(28)); 2*28
```

Out[161]:

56

In [162]:

```
#Sage 的 sigma(n,k) 函数求 n 的所有因子的 k 次幂的和:
sigma(28,0); sigma(28,1); sigma(28,2)
```

Out[162]:

1050

In [163]:

```
#下面展示的是推广的 Euclidean 算法, Euler 的 ϕ 函数和中国剩余定理:
d,u,v = xgcd(12,15)
```

In [164]:

```
d == u*12 + v*15
```

Out[164]:

True

In [165]:

```
n = 2005
inverse_mod(3,n)
```

Out[165]:

1337

In [166]:

```
3 * 1337
```

Out[166]:

4011

In [168]:

```
euler_phi(n)
```

Out[168]:

1600

In [169]:

```
# 中国剩余定理
x = crt(2, 1, 3, 5); x
x % 3 # x mod 3 = 2
x % 5 # x mod 5 = 1
```

Out[169]:

1