Sage 教程

发行版本 *10.7.beta0*

The Sage Development Team

2025 年 04 月 10 日

Contents

Sage 是一个免费的开源数学软件，支持代数、几何、数论、密码学、数值计算及相关领域的研究和教学。Sage 的开发模式和技术特点极其强调开放性、社区性、合作性和协作性：我们是在造车，而不是在重新发明轮子。Sage 的总体目标是创建一个可行的、免费的、开源替代品，用来替代 Maple、Mathematica、Magma 和 MATLAB。

本教程是让你在短时间内熟悉 Sage 的最佳方式。你可以阅读 HTML 或 PDF 版本，也可以从 Sage notebook 中阅读（点击 `Help`，然后点击 `Tutorial` 以交互方式在 Sage 中完成教程）。

CHAPTER 1

介绍

完成本教程最多需要 3-4 小时。你可以阅读本教程的 HTML 或 PDF 版本，或者在 Sage Notebook 中点击 `Help`，然后点击 `Tutorial` 以交互方式在 Sage 中完成教程。

虽然 Sage 的大部分是使用 Python 实现的，但阅读本教程并不需要 Python 背景。可能你会在某个时点希望学习 Python（一门非常有趣的语言！），有很多优秀的免费资源可以帮助你：Python 初学者指南 [PyB] 列出了许多选择。如果你只是想快速试用 Sage，那么本教程是很好的起点。例如：

```
sage: 2 + 2
4
sage: factor(-2007)
-1 * 3^2 * 223

sage: A = matrix(4,4, range(16)); A
[ 0  1  2  3]
[ 4  5  6  7]
[ 8  9 10 11]
[12 13 14 15]

sage: factor(A.charpoly())
x^2 * (x^2 - 30*x - 80)

sage: m = matrix(ZZ,2, range(4))
sage: m[0,0] = m[0,0] - 3
sage: m
[-3  1]
[ 2  3]

sage: E = EllipticCurve([1,2,3,4,5]);
sage: E
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5
over Rational Field
sage: E.anlist(10)
[0, 1, 1, 0, -1, -3, 0, -1, -3, -3, -3]
sage: E.rank()
```

(续下页)

(接上页)

```
1

sage: k = 1/(sqrt(3)*I + 3/4 + sqrt(73)*5/9); k
36/(20*sqrt(73) + 36*I*sqrt(3) + 27)
sage: N(k)
0.165495678130644 - 0.0521492082074256*I
sage: N(k,30)       # 30 "bits"
0.16549568 - 0.052149208*I
sage: latex(k)
\frac{36}{20 \, \sqrt{73} + 36 i \, \sqrt{3} + 27}
```

1.1 安装

如果你的电脑没有安装 Sage，只是想尝试一些命令，可以在 http://sagecell.sagemath.org 上在线使用。

请参阅 Sage 主页 [SA] 文档中的安装指南，了解如何在你的电脑上安装 Sage。以下是一些简要说明。

1. Sage 下载文件附带所有所需组件。换句话说，虽然 Sage 使用 Python、IPython、PARI、GAP、Singular、Maxima、NTL、GMP 等，你不需要单独安装它们，因为它们已经包含在 Sage 发行版中。但是，要使用某些 Sage 功能，例如 Macaulay 或 KASH，你必须确保电脑已经安装了相关程序。
2. Sage 的预编译二进制版本（可以在 Sage 官网上找到）可能比源代码版本更容易和更快安装。只需解压文件并运行 `sage`。
3. 如果你想使用 SageTeX 包（允许你将 Sage 计算结果嵌入到 LaTeX 文件中），你需要让 TeX 发行版识别 SageTeX。请参阅 Sage 安装指南 中的“让 TeX 识别 SageTeX”章节（这个链接 ../installation/index.html 会为你打开安装指南）。其实非常简单；你只需要设置一个环境变量或复制一个文件到 TeX 的搜索目录中。

   如何使用 SageTeX 的文档位于 `$SAGE_ROOT/venv/share/texmf/tex/latex/sagetex/`，其中”`$SAGE_ROOT`” 指的是 Sage 的安装目录，例如 `/opt/sage-9.6`。

1.2 使用 Sage 的方法

Sage 可以通过多种方式使用：

- **Notebook 图形界面：** 运行 `sage -n jupyter`; 请参阅 Jupyter 在线文档,
- **交互式 Shell：** 请参阅交互式 *Shell*,
- **编写程序：** 在 Sage 中编写解释和编译的程序（请参阅加载和附加 *Sage* 文件 和创建编译代码)
- **编写脚本：** 编写使用 Sage 库的独立 Python 脚本（请参阅独立 *Python/Sage* 脚本).

1.3 Sage 的长期目标

- **实用：** Sage 的目标受众包括学习数学的学生（从高中到研究生）、教师和研究数学家。旨在提供可以用于探索和实验代数、几何、数论、微积分、数值计算等数学构造的软件。Sage 能够帮助用户更方便地进行数学对象的交互实验。
- **高效：** Sage 追求快速。它使用高度优化的成熟软件，如 GMP、PARI、GAP 和 NTL，因此在某些操作上非常快速。
- **免费开源：** 源代码必须免费提供且可读，用户可以了解系统的实际运作，并能够更容易地进行扩展。正如数学家通过仔细阅读或浏览证明来深入理解定理一样，用户应该能够通过阅读带有文档的源代码来

理解计算过程。如果在发表的论文中使用 Sage 进行计算，读者将始终可以免费访问 Sage 及其所有源代码，你甚至可以归档和重新分发你使用的 Sage 版本。

- **易于编译：** Sage 应该易于从源代码编译，适用于 Linux、OS X 和 Windows 用户，这使得用户修改系统更加灵活。
- **协作：**提供与大多数其他计算机代数系统的强大接口，包括 PARI、GAP、Singular、Maxima、KASH、Magma、Maple 和 Mathematica。Sage 旨在统一和扩展现有数学软件。
- **文档齐全：**提供教程、编程指南、参考手册和操作指南，包含大量示例和背景数学讨论。
- **可扩展：**能够定义新的数据类型或从内置类型派生，并能够使用多种编程语言编写的代码。
- **用户友好：**功能易于理解，文档和源代码易于查看，并且提供高水平的用户支持。

CHAPTER 2

导览

本节将带你了解 Sage 中的一些功能。更多示例请参考“Sage 构造”，该部分旨在回答“我如何构造...?”这样的常见问题。此外，你还可以查阅《Sage 参考手册》，其中包含数千个示例。请注意，你可以通过点击 `Help` 链接，在 Sage Notebook 中交互式地进行此导览。

（如果你在 Sage Notebook 中查看此教程，请按 `shift-enter` 来运行输入单元格。在按下 shift-enter 之前，你甚至可以编辑输入。在某些 Mac 上，你可能需要按 `shift-return` 而不是 `shift-enter`。）

2.1 赋值、等式和算术

Sage 基本上使用 Python 编程语言，因此大多数 Python 入门书籍都能帮助你学习 Sage。

Sage 使用 = 进行赋值。使用 ==, <=, >=, < 和 > 进行比较：

```
sage: a = 5
sage: a
5
sage: 2 == 2
True
sage: 2 == 3
False
sage: 2 < 3
True
sage: a == 5
True
```

Sage 提供所有基本的数学运算：

```
sage: 2**3    #  ** means exponent
8
sage: 2^3     #  ^ is a synonym for ** (unlike in Python)
8
sage: 10 % 3  #  for integer arguments, % means mod, i.e., remainder
1
```

(续下页)

(接上页)

```
sage: 10/4
5/2
sage: 10//4   #  for integer arguments, // returns the integer quotient
2
sage: 4 * (10 // 4) + 10 % 4 == 10
True
sage: 3^2*4 + 2%5
38
```

像 `3^2*4 + 2%5` 这样的表达式的计算取决于运算的顺序；算术二元运算符的优先级 中的 “运算符优先级表” 给出了明确的规定。

Sage 还提供了许多常见数学函数；以下是一些例子：

```
sage: sqrt(3.4)
1.84390889145858
sage: sin(5.135)
-0.912021158525540
sage: sin(pi/3)
1/2*sqrt(3)
```

如最后一个例子所示，一些数学表达式返回 “精确” 值，而不是近似值。要获得数值近似，可以使用函数 `N` 或方法 `n` （二者都有一个更长的名称 `numerical_approx`，函数 `N` 与 `n` 相同）。这些函数接受可选参数 `prec`，即请求的精度位数，以及 `digits`，即请求的十进制精度位数；默认精度为 53 位。

```
sage: exp(2)
e^2
sage: n(exp(2))
7.38905609893065
sage: sqrt(pi).numerical_approx()
1.77245385090552
sage: sin(10).n(digits=5)
-0.54402
sage: N(sin(10),digits=10)
-0.5440211109
sage: numerical_approx(pi, prec=200)
3.1415926535897932384626433832795028841971693993751058209749
```

Python 是动态类型语言，所以每个变量引用的值都有一个类型与之关联，但在给定作用域内，一个给定变量可以保存任意 Python 类型的值：

```
sage: a = 5   # a is an integer
sage: type(a)
<class 'sage.rings.integer.Integer'>
sage: a = 5/3  # now a is a rational number
sage: type(a)
<class 'sage.rings.rational.Rational'>
sage: a = 'hello'  # now a is a string
sage: type(a)
<... 'str'>
```

C 语言作为静态类型语言就非常不同；一个声明为 int 类型的变量在其作用域内只能保存 int。

2.2 获取帮助

Sage 有大量的内置文档，可以通过输入函数或常量的名称，然后加上问号来访问：

```
sage: tan?
Type:        <class 'sage.calculus.calculus.Function_tan'>
Definition:  tan( [noargspec] )
Docstring:

    The tangent function

    EXAMPLES:
        sage: tan(pi)
        0
        sage: tan(3.1415)
        -0.0000926535900581913
        sage: tan(3.1415/4)
        0.999953674278156
        sage: tan(pi/4)
        1
        sage: tan(1/2)
        tan(1/2)
        sage: RR(tan(1/2))
        0.546302489843790
sage: log2?
Type:        <class 'sage.functions.constants.Log2'>
Definition:  log2( [noargspec] )
Docstring:

    The natural logarithm of the real number 2.

    EXAMPLES:
        sage: log2
        log2
        sage: float(log2)
        0.69314718055994529
        sage: RR(log2)
        0.693147180559945
        sage: R = RealField(200); R
        Real Field with 200 bits of precision
        sage: R(log2)
        0.69314718055994530941723212145817656807550013436025525412068
        sage: l = (1-log2)/(1+log2); l
        (1 - log(2))/(log(2) + 1)
        sage: R(l)
        0.18123221829928249948761381864650311423330609774776013488056
        sage: maxima(log2)
        log(2)
        sage: maxima(log2).float()
        .6931471805599453
        sage: gp(log2)
        0.69314718055994530941723212145817656807   # 64-bit
        0.69314718055994530941723212145             # 32-bit
sage: sudoku?
File:        sage/local/lib/python2.5/site-packages/sage/games/sudoku.py
Type:        <... 'function'>
Definition:  sudoku(A)
Docstring:
```

(续下页)

(接上页)

```
    Solve the 9x9 Sudoku puzzle defined by the matrix A.

    EXAMPLE:
        sage: A = matrix(ZZ,9,[5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0,
    0,3,0, 0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0, 0,0,0, 0,0,0, 2,0,8, 0,0,0,
    0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4, 1,5,0,   0,3,0, 0,0,2,
    0,0,0, 4,9,0, 0,5,0, 0,0,3])
        sage: A
        [5 0 0 0 8 0 0 4 9]
        [0 0 0 5 0 0 0 3 0]
        [0 6 7 3 0 0 0 0 1]
        [1 5 0 0 0 0 0 0 0]
        [0 0 0 2 0 8 0 0 0]
        [0 0 0 0 0 0 0 1 8]
        [7 0 0 0 0 4 1 5 0]
        [0 3 0 0 0 2 0 0 0]
        [4 9 0 0 5 0 0 0 3]
        sage: sudoku(A)
        [5 1 3 6 8 7 2 4 9]
        [8 4 9 5 2 1 6 3 7]
        [2 6 7 3 4 9 5 8 1]
        [1 5 8 4 6 3 9 7 2]
        [9 7 4 2 1 8 3 6 5]
        [3 2 6 7 9 5 4 1 8]
        [7 8 2 9 3 4 1 5 6]
        [6 3 5 1 7 2 8 9 4]
        [4 9 1 8 5 6 7 2 3]
```

Sage 还提供了“Tab 补全”功能：输入函数的前几个字母，然后按下 `Tab` 键。例如，如果你输入 `ta` 然后按下 `Tab`，Sage 会显示 `tachyon, tan, tanh, taylor`。这是查找 Sage 中函数和其他结构名称的好方法。

2.3 函数、缩进和计数

在 Sage 中定义一个新函数，请使用 `def` 命令，并在变量名列表后加上冒号。例如：

```
sage: def is_even(n):
....:     return n%2 == 0
sage: is_even(2)
True
sage: is_even(3)
False
```

注意：根据你查看的教程版本，你可能会在本例的第二行看到三个点 `....:`。请勿输入它们，它们只是为了强调代码的缩进。在这种情况下，请在块末尾按 [Return/Enter] 以插入空行并结束函数定义。

你不需要指定输入参数的类型。你可以指定多个输入，每个输入都可以有一个可选的默认值。例如，如果未指定 `divisor`，则下面的函数默认值为 `divisor=2`。

```
sage: def is_divisible_by(number, divisor=2):
....:     return number%divisor == 0
sage: is_divisible_by(6,2)
True
sage: is_divisible_by(6)
True
```

(续下页)

(接上页)

```
sage: is_divisible_by(6, 5)
False
```

调用函数时，你还可以显式地指定一个或多个输入；如果你显式地指定输入，可以以任意顺序给出它们：

```
sage: is_divisible_by(6, divisor=5)
False
sage: is_divisible_by(divisor=2, number=6)
True
```

在 Python 中，代码块不是用大括号或其他语言中的开始和结束标记来表示的。相反，代码块由缩进来表示，缩进必须完全匹配。例如，以下是一个语法错误，因为 `return` 语句的缩进与上面的其他行不一致：

```
sage: def even(n):
....:     v = []
....:     for i in range(3,n):
....:         if i % 2 == 0:
....:             v.append(i)
....:    return v
Syntax Error:
       return v
```

如果你修复了缩进，函数就可以正常工作：

```
sage: def even(n):
....:     v = []
....:     for i in range(3,n):
....:         if i % 2 == 0:
....:             v.append(i)
....:     return v
sage: even(10)
[4, 6, 8]
```

行末不需要分号；在大多数情况下，行以换行符结束。但是，你可以在一行上放置多个语句，用分号间隔：

```
sage: a = 5; b = a + 3; c = b^2; c
64
```

如果你希望一行代码跨越多行，可以使用反斜杠：

```
sage: 2 + \
....:    3
5
```

在 Sage 中，你可以通过遍历整数区间来计数。例如，下面代码的第一行与 C++ 或 Java 中的 `for(i=0; i<3; i++)` 完全一样：

```
sage: for i in range(3):
....:     print(i)
0
1
2
```

下面代码的第一行与 `for(i=2;i<5;i++)` 等价。

```
sage: for i in range(2,5):
....:     print(i)
2
3
4
```

第三个参数控制步长，所以下面代码与 `for(i=1;i<6;i+=2)` 等价。

```
sage: for i in range(1,6,2):
....:     print(i)
1
3
5
```

通常你会希望创建一个漂亮的表格来显示你使用 Sage 计算的数字。一个简单的方法是使用格式化字符串。下面，我们创建三个宽度正好为 6 的列，并制作一个平方和立方的表格。

```
sage: for i in range(5):
....:     print('%6s %6s %6s' % (i, i^2, i^3))
     0      0      0
     1      1      1
     2      4      8
     3      9     27
     4     16     64
```

Sage 中最基本的数据结构是列表，顾名思义，就是一个任意对象的列表。例如，以下命令使用 `range` 创建一个列表:

```
sage: list(range(2,10))
[2, 3, 4, 5, 6, 7, 8, 9]
```

下面是一个更复杂的列表:

```
sage: v = [1, "hello", 2/3, sin(x^3)]
sage: v
[1, 'hello', 2/3, sin(x^3)]
```

如如许多编程语言一样，列表的索引是从 0 开始。

```
sage: v[0]
1
sage: v[3]
sin(x^3)
```

使用 `len(v)` 获取 `v` 的长度，使用 `v.append(obj)` 将新对象追加到 `v` 的末尾，使用 `del v[i]` 删除 `v` 的第 i 项:

```
sage: len(v)
4
sage: v.append(1.5)
sage: v
[1, 'hello', 2/3, sin(x^3), 1.50000000000000]
sage: del v[1]
sage: v
[1, 2/3, sin(x^3), 1.50000000000000]
```

另一个重要的数据结构是字典（或关联数组）。字典的工作方式类似于列表，但它可以用几乎任何对象来索引（索引必须是不可变的）:

```
sage: d = {'hi':-2,  3/8:pi,   e:pi}
sage: d['hi']
-2
sage: d[e]
pi
```

你还可以使用类定义新的数据类型。使用类封装数学对象是一种强大的技术，可以帮助简化和组织你的 Sage 程序。下面，我们定义一个表示不超过 n 的正偶数列表的类；它从内置类型 `list` 派生而来。

```
sage: class Evens(list):
....:     def __init__(self, n):
....:         self.n = n
....:         list.__init__(self, range(2, n+1, 2))
....:     def __repr__(self):
....:         return "Even positive numbers up to n."
```

`__init__` 方法在创建对象时调用以初始化对象；`__repr__` 方法打印对象。我们在 `__init__` 方法的第二行调用列表构造函数。下面我们创建 `Evens` 类的对象：

```
sage: e = Evens(10)
sage: e
Even positive numbers up to n.
```

注意，`e` 使用我们定义的 `__repr__` 方法打印。要查看底层数字列表，请使用 `list` 函数：

```
sage: list(e)
[2, 4, 6, 8, 10]
```

我们还可以访问属性 `n` 或像列表一样操作 `e`。

```
sage: e.n
10
sage: e[2]
6
```

2.4 基本代数和微积分

Sage 能够进行多种与基本代数和微积分相关的计算，例如求解方程、微分、积分和拉普拉斯变换。更多示例，请参阅 Sage Constructions 。

在所有这些示例中，函数中的变量都需要使用 `var(...)` 定义。例如：

```
sage: u = var('u')
sage: diff(sin(u), u)
cos(u)
```

如果遇到 `NameError` 错误，请检查是否拼写错误，或者是否忘记使用 `var(...)` 定义变量。

2.4.1 求解方程

精确求解方程

`solve` 函数用于求解方程。使用时，首先定义变量；然后将方程（或方程组）和需要求解的变量作为 `solve` 的参数：

```
sage: x = var('x')
sage: solve(x^2 + 3*x + 2, x)
[x == -2, x == -1]
```

你可以求解单变量方程，其他变量作为参数：

```
sage: x, b, c = var('x b c')
sage: solve([x^2 + b*x + c == 0],x)
[x == -1/2*b - 1/2*sqrt(b^2 - 4*c), x == -1/2*b + 1/2*sqrt(b^2 - 4*c)]
```

你也可以求解多变量方程：

```
sage: x, y = var('x, y')
sage: solve([x+y==6, x-y==4], x, y)
[[x == 5, y == 1]]
```

以下是由 Jason Grout 提供的使用 Sage 求解非线性方程组的示例：首先，我们符号化地求解该方程组：

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

对于解的数值近似，可以使用：

```
sage: solns = solve([eq1,eq2,eq3,p==1],p,q,x,y, solution_dict=True)
sage: [[s[p].n(30), s[q].n(30), s[x].n(30), s[y].n(30)] for s in solns]
[[1.0000000, 8.0000000, -4.8830369, -0.13962039],
 [1.0000000, 8.0000000, 3.5497035, -1.1937129]]
```

（函数 `n` 用于打印数值近似，参数是精度的位数。）

数值求解方程

很多时候，`solve` 无法找到指定方程或方程组的精确解。此时可以使用 `find_root` 找到数值解。例如，solve 对以下方程没有返回任何有意义的结果：

```
sage: theta = var('theta')
sage: solve(cos(theta)==sin(theta), theta)
[sin(theta) == cos(theta)]
```

另一方面，可以使用 `find_root` 在区间 $0 < \phi < \pi/2$ 内找到上述方程的解：

```
sage: phi = var('phi')
sage: find_root(cos(phi)==sin(phi),0,pi/2)
0.785398163397448...
```

2.4.2 微分、积分及其他

Sage 可以对许多函数进行微分和积分。例如，对 $\sin(u)$ 相对于 u 进行微分，可以这样做：

```
sage: u = var('u')
sage: diff(sin(u), u)
cos(u)
```

计算 $\sin(x^2)$ 的四阶导数：

```
sage: diff(sin(x^2), x, 4)
16*x^4*sin(x^2) - 48*x^2*cos(x^2) - 12*sin(x^2)
```

分别计算 $x^2 + 17y^2$ 相对于 x 和 y 的偏导数：

```
sage: x, y = var('x,y')
sage: f = x^2 + 17*y^2
sage: f.diff(x)
2*x
sage: f.diff(y)
34*y
```

接下来讨论积分，包括不定积分和定积分。计算 $\int x\sin(x^2)\,dx$ 和 $\int_0^1 \frac{x}{x^2+1}\,dx$

```
sage: integral(x*sin(x^2), x)
-1/2*cos(x^2)
sage: integral(x/(x^2+1), x, 0, 1)
1/2*log(2)
```

计算 $\frac{1}{x^2-1}$ 的部分分式分解：

```
sage: f = 1/((1+x)*(x-1))
sage: f.partial_fraction(x)
-1/2/(x + 1) + 1/2/(x - 1)
```

2.4.3 求解微分方程

你可以用 Sage 来研究常微分方程。求解方程 $x' + x - 1 = 0$：

```
sage: t = var('t')     # define a variable t
sage: x = function('x')(t)    # define x to be a function of that variable
sage: DE = diff(x, t) + x - 1
sage: desolve(DE, [x,t])
(_C + e^t)*e^(-t)
```

这里使用 Sage 与 Maxima [Max] 的接口，因此其输出可能与其他 Sage 输出有所不同。上面示例中，输出表示该微分方程的一般解是 $x(t) = e^{-t}(e^t + c)$。

你还可以计算拉普拉斯变换；计算 $t^2e^t - \sin(t)$ 的拉普拉斯变换如下：

```
sage: s = var("s")
sage: t = var("t")
sage: f = t^2*exp(t) - sin(t)
sage: f.laplace(t,s)
-1/(s^2 + 1) + 2/(s - 1)^3
```

这里是一个更复杂的示例。左侧连接到墙上的耦合弹簧的平衡位移

```
|------\/\/\/\/\---|mass1|----\/\/\/\/\/----|mass2|
         spring1                spring2
```

由二阶微分方程组建模

$$m_1 x_1'' + (k_1 + k_2)x_1 - k_2 x_2 = 0$$
$$m_2 x_2'' + k_2(x_2 - x_1) = 0,$$

其中 m_i 是物体 i 的质量，x_i 是质量 i 的平衡位移，k_i 是弹簧 i 的弹簧常数。

示例： 使用 Sage 求解上述问题，其中 $m_1 = 2, m_2 = 1, k_1 = 4, k_2 = 2, x_1(0) = 3, x_1'(0) = 0, x_2(0) = 3, x_2'(0) = 0.$

解：对第一个方程进行拉普拉斯变换（符号 $x = x_1, y = x_2$）：

```
sage: t,s = SR.var('t,s')
sage: x = function('x')
sage: y = function('y')
sage: f = 2*x(t).diff(t,2) + 6*x(t) - 2*y(t)
sage: f.laplace(t,s)
2*s^2*laplace(x(t), t, s) - 2*s*x(0) + 6*laplace(x(t), t, s) - 2*laplace(y(t), t, s) -␣
↪2*D[0](x)(0)
```

输出虽然难以阅读，但其表示

$$-2x'(0) + 2s^2 \cdot X(s) - 2sx(0) - 2Y(s) + 6X(s) = 0$$

（其中小写函数如 $x(t)$ 的拉普拉斯变换是大写函数 $X(s)$）。对第二个方程进行拉普拉斯变换：

```
sage: de2 = maxima("diff(y(t),t, 2) + 2*y(t) - 2*x(t)")
sage: lde2 = de2.laplace("t","s"); lde2.sage()
s^2*laplace(y(t), t, s) - s*y(0) - 2*laplace(x(t), t, s) + 2*laplace(y(t), t, s) - D[0](y)(0)
```

这表示

$$-Y'(0) + s^2 Y(s) + 2Y(s) - 2X(s) - sy(0) = 0.$$

代入初始条件 $x(0), x'(0), y(0),$ 和 $y'(0)$，并求解所得的两个方程：

```
sage: var('s X Y')
(s, X, Y)
sage: eqns = [(2*s^2+6)*X-2*Y == 6*s, -2*X +(s^2+2)*Y == 3*s]
sage: solve(eqns, X,Y)
[[X == 3*(s^3 + 3*s)/(s^4 + 5*s^2 + 4),
  Y == 3*(s^3 + 5*s)/(s^4 + 5*s^2 + 4)]]
```

此时进行逆拉普拉斯变换即可得到答案：

```
sage: var('s t')
(s, t)
sage: inverse_laplace((3*s^3 + 9*s)/(s^4 + 5*s^2 + 4),s,t)
cos(2*t) + 2*cos(t)
sage: inverse_laplace((3*s^3 + 15*s)/(s^4 + 5*s^2 + 4),s,t)
-cos(2*t) + 4*cos(t)
```

因此，解为

$$x_1(t) = \cos(2t) + 2\cos(t), \quad x_2(t) = 4\cos(t) - \cos(2t).$$

可以使用参数方式绘制函数图像

```
sage: t = var('t')
sage: P = parametric_plot((cos(2*t) + 2*cos(t), 4*cos(t) - cos(2*t) ),
....:     (t, 0, 2*pi), rgbcolor=hue(0.9))
sage: show(P)
```

也可以分开绘制两个函数的图像

```
sage: t = var('t')
sage: p1 = plot(cos(2*t) + 2*cos(t), (t,0, 2*pi), rgbcolor=hue(0.3))
sage: p2 = plot(4*cos(t) - cos(2*t), (t,0, 2*pi), rgbcolor=hue(0.6))
sage: show(p1 + p2)
```

有关绘图的更多信息，请参见绘图。有关微分方程的更多信息，请参见 [NagleEtAl2004] 的第 5.5 节。

2.4.4 欧拉法求解微分方程组

在下一个示例中，我们将演示欧拉法求解一阶和二阶常微分方程。首先回顾一下一阶方程的基本思想。给定初值问题的形式为

$$y' = f(x, y), \quad y(a) = c,$$

我们要找到解在 $x = b$ 处的近似值，其中 $b > a$。

回顾导数的定义

$$y'(x) \approx \frac{y(x+h) - y(x)}{h},$$

其中 $h > 0$ 是一个给定且极小的数。结合微分方程可以得到 $f(x, y(x)) \approx \frac{y(x+h)-y(x)}{h}$。现在求解 $y(x+h)$:

$$y(x+h) \approx y(x) + h \cdot f(x, y(x)).$$

如果我们把 $h \cdot f(x, y(x))$ 称为“校正项”（因为没有更好的名称），把 $y(x)$ 称为“y 的旧值”，把 $y(x+h)$ 称为“y 的新值”，那么这个近似可以重新表示为

$$y_{new} \approx y_{old} + h \cdot f(x, y_{old}).$$

如果我们将从 a 到 b 的区间分成 n 步，使得 $h = \frac{b-a}{n}$，那么我们可以在表中记录此方法的信息。

x	y	$h \cdot f(x, y)$
a	c	$h \cdot f(a, c)$
$a + h$	$c + h \cdot f(a, c)$	...
$a + 2h$	...	
...		
$b = a + nh$	???	...

我们的目标是逐行填满表中的所有空白，直到到达??? 条目，这就是欧拉法对 $y(b)$ 的近似值。

求解微分方程组的思想与之类似。

示例： 数值近似 $z(t)$ 在 $t = 1$ 处的值，使用欧拉法的 4 个步骤，其中 $z'' + tz' + z = 0, z(0) = 1, z'(0) = 0$。

我们必须将二阶常微分方程简化为两个一阶常微分方程组（使用 $x = z, y = z'$）并应用欧拉法：

```
sage: t,x,y = PolynomialRing(RealField(10),3,"txy").gens()
sage: f = y; g = -x - y * t
sage: eulers_method_2x2(f,g, 0, 1, 0, 1/4, 1)
      t                x            h*f(t,x,y)                y       h*g(t,x,y)
      0                1                  0.00                0           -0.25
    1/4              1.0                -0.062            -0.25           -0.23
    1/2             0.94                 -0.12            -0.48           -0.17
    3/4             0.82                 -0.16            -0.66          -0.081
      1             0.65                 -0.18            -0.74           0.022
```

因此，$z(1) \approx 0.65$.

我们还可以绘制点 (x, y) 以获得曲线的近似图。函数 `eulers_method_2x2_plot` 将执行此操作；为了使用它，我们需要定义函数 f 和 g，它们接受一个带有三个坐标的参数：$(t, x, 'y')$。

```
sage: f = lambda z: z[2]         # f(t,x,y) = y
sage: g = lambda z: -sin(z[1])   # g(t,x,y) = -sin(x)
sage: P = eulers_method_2x2_plot(f,g, 0.0, 0.75, 0.0, 0.1, 1.0)
```

此时，`P` 存储了两个图：`P[0]`, x 相对于 t 的图, 以及 `P[1]`, y 相对于 t 的图。我们可以通过如下代码绘制这两个图：

```
sage: show(P[0] + P[1])
```

（有关绘图的更多信息，请参见绘图。）

2.4.5 特殊函数

Sage 利用 PARI [GAP] 和 Maxima [Max] , 实现了多种正交多项式和特殊函数。这些函数在 Sage 参考手册的相应部分（“正交多项式”和“特殊函数”）中有详细文档。

```
sage: x = polygen(QQ, 'x')
sage: chebyshev_U(2,x)
4*x^2 - 1
sage: bessel_I(1,1).n(250)
0.56515910399248502720769602760986330732889962162109200948029448947925564096
sage: bessel_I(1,1).n()
0.565159103992485
sage: bessel_I(2,1.1).n()
0.167089499251049
```

此时，Sage 仅将这些函数包装用于数值使用。对于符号使用，请直接使用 Maxima 接口，如以下示例：

```
sage: maxima.eval("f:bessel_y(v, w)")
'bessel_y(v,w)'
sage: maxima.eval("diff(f,w)")
'(bessel_y(v-1,w)-bessel_y(v+1,w))/2'
```

2.4.6 向量微积分

参见 Vector Calculus Tutorial.

2.5 绘图

Sage 可以生成二维和三维图形。

2.5.1 二维图形

在二维中，Sage 可以绘制圆、线和多边形；在直角坐标系中绘制函数图形；还可以绘制极坐标图、轮廓图和矢量场图。本文档展示了若干这些图形的例子。有关使用 Sage 绘图的更多例子，请参见求解微分方程和*Maxima*，以及 Sage Constructions 文档。

该命令生成一个位于原点的半径为 1 的黄色圆：

```
sage: circle((0,0), 1, rgbcolor=(1,1,0))
Graphics object consisting of 1 graphics primitive
```

你还可以生成一个填充的圆：

```
sage: circle((0,0), 1, rgbcolor=(1,1,0), fill=True)
Graphics object consisting of 1 graphics primitive
```

你还可以通过将圆赋值给变量来创建圆；这样做不会将圆绘制出来：

```
sage: c = circle((0,0), 1, rgbcolor=(1,1,0))
```

要想绘制它，可以使用 `c.show()` 或 `show(c)`，如下所示：

```
sage: c.show()
```

或者，使用 `c.save('filename.png')` 将绘图保存到给定文件。

现在，这些“圆”看起来更像椭圆，因为坐标轴的比例不同。你可以这样修复这个问题：

```
sage: c.show(aspect_ratio=1)
```

命令 `show(c, aspect_ratio=1)` 可以完成同样的事情，或者你可以使用 `c.save('filename.png', aspect_ratio=1)` 保存图片。

绘制基本函数很容易：

```
sage: plot(cos, (-5,5))
Graphics object consisting of 1 graphics primitive
```

一旦你指定了变量名称，你还可以创建参数化图形：

```
sage: x = var('x')
sage: parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
Graphics object consisting of 1 graphics primitive
```

一定要注意，只有当原点在图形的视图范围内时，图形的轴才会相交，并且对于非常大的数值可能会使用科学计数法：

```
sage: plot(x^2,(x,300,500))
Graphics object consisting of 1 graphics primitive
```

你可以通过将多个图形相加来将他们组合在一起：

```
sage: x = var('x')
sage: p1 = parametric_plot((cos(x),sin(x)),(x,0,2*pi),rgbcolor=hue(0.2))
sage: p2 = parametric_plot((cos(x),sin(x)^2),(x,0,2*pi),rgbcolor=hue(0.4))
sage: p3 = parametric_plot((cos(x),sin(x)^3),(x,0,2*pi),rgbcolor=hue(0.6))
sage: show(p1+p2+p3, axes=false)
```

生成填充形状的一个好方法是生成点列表（示例中的 `L`），然后使用 `polygon` 命令绘制由这些点构成边界的形状。例如，下面是一个绿色的三角形：

```
sage: L = [[-1+cos(pi*i/100)*(1+cos(pi*i/100)),
....:      2*sin(pi*i/100)*(1-cos(pi*i/100))] for i in range(200)]
sage: p = polygon(L, rgbcolor=(1/8,3/4,1/2))
sage: p
Graphics object consisting of 1 graphics primitive
```

输入 `show(p, axes=false)` 来查看没有任何坐标轴的图形。

你可以向图形中添加文本：

```
sage: L = [[6*cos(pi*i/100)+5*cos((6/2)*pi*i/100),
....:      6*sin(pi*i/100)-5*sin((6/2)*pi*i/100)] for i in range(200)]
sage: p = polygon(L, rgbcolor=(1/8,1/4,1/2))
sage: t = text("hypotrochoid", (5,4), rgbcolor=(1,0,0))
sage: show(p+t)
```

微积分老师经常在黑板上绘制以下图形：arcsin 的多个周期：即 $y = \sin(x)$ 对于 x 在 -2π 和 2π 区间的图像，围绕 45 度线翻转。以下 Sage 命令构造此图形：

```
sage: v = [(sin(x),x) for x in srange(-2*float(pi),2*float(pi),0.1)]
sage: line(v)
Graphics object consisting of 1 graphics primitive
```

由于正切函数的值域比正弦函数大得多，如果你使用相同技巧绘制反正切的图像，你应该更改 x 轴的最大和最小坐标：

```
sage: v = [(tan(x),x) for x in srange(-2*float(pi),2*float(pi),0.01)]
sage: show(line(v), xmin=-20, xmax=20)
```

Sage 还能计算极坐标图、轮廓图和矢量场图（针对特殊类型的函数）。这里是一个轮廓图的例子：

```
sage: f = lambda x,y: cos(x*y)
sage: contour_plot(f, (-4, 4), (-4, 4))
Graphics object consisting of 1 graphics primitive
```

2.5.2 三维图形

Sage 还可以用于创建三维图形。在 notebook 和 REPL 中，这些图形将默认使用开源软件包 [ThreeJS] 显示，该软件包支持使用鼠标交互式旋转和缩放图形。

使用 `plot3d` 绘制形如 $f(x, y) = z$ 的函数图像：

```
sage: x, y = var('x,y')
sage: plot3d(x^2 + y^2, (x,-2,2), (y,-2,2))
Graphics3d Object
```

或者，你可以使用 `parametric_plot3d` 绘制参数曲面，其中每个 x, y, z 由一个或两个变量（通常是 u 和 v）的函数确定。前面的图形可以参数化地表达如下：

```
sage: u, v = var('u, v')
sage: f_x(u, v) = u
sage: f_y(u, v) = v
sage: f_z(u, v) = u^2 + v^2
sage: parametric_plot3d([f_x, f_y, f_z], (u, -2, 2), (v, -2, 2))
Graphics3d Object
```

在 Sage 中绘制 3D 曲面的第三种方法是 $implicit_plot3d$，它绘制形如 $f(x, y, z) = 0$ 的函数的轮廓（这定义了一组点）。我们使用经典公式绘制一个球体：

```
sage: x, y, z = var('x, y, z')
sage: implicit_plot3d(x^2 + y^2 + z^2 - 4, (x,-2, 2), (y,-2, 2), (z,-2, 2))
Graphics3d Object
```

下面是更多的例子：

Yellow Whitney's umbrella:

```
sage: u, v = var('u,v')
sage: fx = u*v
sage: fy = u
sage: fz = v^2
sage: parametric_plot3d([fx, fy, fz], (u, -1, 1), (v, -1, 1),
....:    frame=False, color="yellow")
Graphics3d Object
```

Cross cap:

```
sage: u, v = var('u,v')
sage: fx = (1+cos(v))*cos(u)
sage: fy = (1+cos(v))*sin(u)
sage: fz = -tanh((2/3)*(u-pi))*sin(v)
sage: parametric_plot3d([fx, fy, fz], (u, 0, 2*pi), (v, 0, 2*pi),
....:    frame=False, color="red")
Graphics3d Object
```

挠环面：

```
sage: u, v = var('u,v')
sage: fx = (3+sin(v)+cos(u))*cos(2*v)
sage: fy = (3+sin(v)+cos(u))*sin(2*v)
sage: fz = sin(u)+2*cos(v)
sage: parametric_plot3d([fx, fy, fz], (u, 0, 2*pi), (v, 0, 2*pi),
....:    frame=False, color="red")
Graphics3d Object
```

双纽线：

```
sage: x, y, z = var('x,y,z')
sage: f(x, y, z) = 4*x^2 * (x^2 + y^2 + z^2 + z) + y^2 * (y^2 + z^2 - 1)
sage: implicit_plot3d(f, (x, -0.5, 0.5), (y, -1, 1), (z, -1, 1))
Graphics3d Object
```

2.6 常见函数问题

定义函数的某些方面（例如，用于微分或绘图）可能会令人困惑。我们尝试在本节中解答一些相关问题。

以下是几种可以被称为“函数”的定义方法：

1. 定义一个 Python 函数，如函数、缩进和计数 中所述。这些函数可以被绘制，但不能被微分或积分。

```
sage: def f(z): return z^2
sage: type(f)
<... 'function'>
sage: f(3)
9
sage: plot(f, 0, 2)
Graphics object consisting of 1 graphics primitive
```

请注意最后一行的语法。使用 `plot(f(z), 0, 2)` 会报 `NameError`。因为 `z` 是 `f` 定义中的一个虚拟变量，在该定义之外未定义。为了能够在 plot 命令中使用 `f(z)`，需要将 `z`（或其他所需内容）定义为变量。我们可以使用下面的语法，或者采用我们给出的第二种方法。

```
sage: var('z')   # define z to be a variable
z
sage: f(z)
z^2
sage: plot(f(z), 0, 2)
Graphics object consisting of 1 graphics primitive
```

此时，`f(z)` 是一个符号表达式，即我们接下来要介绍的方法。

2. 定义一个“可调用的符号表达式”。这些表达式可以被绘制、微分和积分。

```
sage: g(x) = x^2
sage: g        # g sends x to x^2
x |--> x^2
sage: g(3)
9
sage: Dg = g.derivative(); Dg
x |--> 2*x
sage: Dg(3)
6
sage: type(g)
<class 'sage.symbolic.expression.Expression'>
sage: plot(g, 0, 2)
Graphics object consisting of 1 graphics primitive
```

注意，虽然 `g` 是一个可调用的符号表达式，但 `g(x)` 是一个相关但不同类型的对象，尽管存在一些问题，但它也可以被绘制、微分等：请参见下文中的第 5 点。

```
sage: g(x)
x^2
sage: type(g(x))
<class 'sage.symbolic.expression.Expression'>
sage: g(x).derivative()
2*x
sage: plot(g(x), 0, 2)
Graphics object consisting of 1 graphics primitive
```

3. 使用预定义的 Sage ’ 微积分函数’。这些函数可以被绘制，并且稍加辅助可以进行微分和积分。

```
sage: type(sin)
<class 'sage.functions.trig.Function_sin'>
sage: plot(sin, 0, 2)
Graphics object consisting of 1 graphics primitive
sage: type(sin(x))
<class 'sage.symbolic.expression.Expression'>
sage: plot(sin(x), 0, 2)
Graphics object consisting of 1 graphics primitive
```

单独使用 `sin` 不能被微分，至少不能得到 `cos`.

```
sage: f = sin
sage: f.derivative()
Traceback (most recent call last):
...
AttributeError: ...
```

用 `f = sin(x)` 替换 `sin` 就可以了，但更好的方法可能是使用 `f(x) = sin(x)` 来定义一个可调用的符号表达式。

```
sage: S(x) = sin(x)
sage: S.derivative()
x |--> cos(x)
```

以下是一些常见问题及其解释：

4. 非预期执行

```
sage: def h(x):
....:     if x<2:
....:         return 0
....:     else:
....:         return x-2
```

问题：`plot(h(x), 0, 4)` 绘制的是直线 $y = x-2$，而不是由 `h` 定义的分段函数。原因是，在命令 `plot(h(x), 0, 4)` 中，首先执行 `h(x)`，这意味着将符号变量 `x` 插入函数 `h` 中。因此，不等式 `x < 2` 首先执行得到 `False`，因此 `h(x)` 会执行 `x - 2`。可以通过以下方法看到这个过程

```
sage: bool(x < 2)
False
sage: h(x)
x - 2
```

注意，这里有两个不同的 `x`：用于定义函数 `h` 的 Python 变量（在其定义中是局部的）和 Sage 启动时可用的符号变量 `x`。

解决方案：不要使用 `plot(h(x), 0, 4)`；而是使用

```
sage: plot(h, 0, 4)
Graphics object consisting of 1 graphics primitive
```

5. 意外产生常数而非函数。

```
sage: f = x
sage: g = f.derivative()
sage: g
1
```

问题：以 `g(3)` 为例，会返回一个错误，提示”ValueError: the number of arguments must be less than or equal to 0.”。

```
sage: type(f)
<class 'sage.symbolic.expression.Expression'>
sage: type(g)
<class 'sage.symbolic.expression.Expression'>
```

`g` 不是函数，而是一个常数，所以它没有关联的变量，不能将任何内容插入其中。

解决方案：有几种选择。

- 将 `f` 定义为符号表达式。

```
sage: f(x) = x        # instead of 'f = x'
sage: g = f.derivative()
sage: g
x |--> 1
sage: g(3)
1
sage: type(g)
<class 'sage.symbolic.expression.Expression'>
```

- 或者保留 `f` 的原始定义，将 `g` 定义为符号表达式。

```
sage: f = x
sage: g(x) = f.derivative()  # instead of 'g = f.derivative()'
sage: g
x |--> 1
sage: g(3)
1
sage: type(g)
<class 'sage.symbolic.expression.Expression'>
```

- 抑或保留 `f` 和 `g` 的原始定义，指定要替换的变量。

```
sage: f = x
sage: g = f.derivative()
sage: g
1
sage: g(x=3)    # instead of 'g(3)'
1
```

最后，还有另一种方法可以区分 `f = x` 和 `f(x) = x` 的导数

```
sage: f(x) = x
sage: g = f.derivative()
sage: g.variables()  # the variables present in g
()
sage: g.arguments()  # the arguments which can be plugged into g
(x,)
sage: f = x
sage: h = f.derivative()
sage: h.variables()
()
sage: h.arguments()
()
```

正如上面例子试图说明的那样，`h` 不接受任何参数，这就是为什么 `h(3)` 会返回错误。

2.7 基本环

在定义矩阵、向量或多项式时，指定它们所定义的“环”非常有用，有时甚至是必须的。环是一种数学结构，具有良好的加法和乘法概念；如果你以前从未听说过它们，你可能只需要了解以下四种常用的环：

- 整数 $\{..., -1, 0, 1, 2, ...\}$，在 Sage 中称为 `ZZ`。
- 有理数 -- 即分数或整数的比率 -- 在 Sage 中称为 `QQ`。
- 实数，在 Sage 中称为 `RR`。
- 复数，在 Sage 中称为 `CC`。

了解这些区别是必要的，因为同一个多项式可能会根据它所定义的环而有所不同。例如，多项式 $x^2 - 2$ 有两个根，$\pm\sqrt{2}$。这些根不是有理数，所以如果你处理的是具有有理系数的多项式，那么这个多项式无法因式分解。但使用实系数，它便可以因式分解。因此，你可能需要指定环以确保获得预期的信息。以下两个命令分别定义了具有有理系数和实系数的多项式集。集合被命名为”ratpoly” 和”realpoly”，但这里并不重要；然而，请注意字符串”.<t>” 和”.<z>” 分别命名了两种情况下使用的 变量。：

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

现在我们来演示 $x^2 - 2$ 的因式分解:

```
sage: factor(t^2-2)
t^2 - 2
sage: factor(z^2-2)
(z - 1.41421356237310) * (z + 1.41421356237310)
```

类似的情况也适用于矩阵：矩阵的行简化形式可能取决于它所定义的环，以及它的特征值和特征向量。有关构造多项式的更多信息，请参见多项式，有关矩阵的更多信息，请参见线性代数。

符号 `I` 表示 -1 的平方根；`i` 是 `I` 的同义词。显然，它不是一个有理数:

```
sage: i  # square root of -1
I
sage: i in QQ
False
```

注意：如果变量 `i` 已被赋予其他值，例如，如果它被用作循环变量，则上述代码可能无法按预期工作。如果是这种情况，请输入:

```
sage: reset('i')
```

以获得 `i` 的原始复数值。

定义复数时有一个需要注意的地方：如上所述，符号 `i` 表示 -1 的平方根，但是它是 -1 的 * 形式 * 平方根，是一个代数数。调用 `CC(i)` 或 `CC.0` 或 `CC.gen(0)` 返回 -1 的 * 复数 * 平方根。通过所谓的强制转换，可以进行涉及不同类型数字的算术运算，请参见父结构、转换与强制转换。

```
sage: i = CC(i)        # floating point complex number
sage: i == CC.0
True
sage: a, b = 4/3, 2/3
sage: z = a + b*i
sage: z
1.33333333333333 + 0.666666666666667*I
sage: z.imag()         # imaginary part
0.666666666666667
```

(续下页)

(接上页)

```
sage: z.real() == a   # automatic coercion before comparison
True
sage: a + b
2
sage: 2*b == a
True
sage: parent(2/3)
Rational Field
sage: parent(4/2)
Rational Field
sage: 2/3 + 0.1       # automatic coercion before addition
0.766666666666667
sage: 0.1 + 2/3       # coercion rules are symmetric in Sage
0.766666666666667
```

以下是 Sage 中一些基本环的更多示例。如上所述，有理数环可以使用 `QQ` 或 `RationalField()` 来引用（域是满足乘法交换律的环，且每个非零元素在该环中都有一个倒数，因此有理数构成一个域，但整数不构成）：

```
sage: RationalField()
Rational Field
sage: QQ
Rational Field
sage: 1/2 in QQ
True
```

十进制数 `1.2` 被认为是 *QQ*‘ 中的数：也可以“强制转换”成有理数的十进制数被认为是有理数（参见父结构、转换与强制转换）。数字 π 和 $\sqrt{2}$ 不是有理数:

```
sage: 1.2 in QQ
True
sage: pi in QQ
False
sage: pi in RR
True
sage: sqrt(2) in QQ
False
sage: sqrt(2) in CC
True
```

为了在高等数学中使用，Sage 还具备其他环，例如有限域，*p*-adic 整数，代数数环，多项式环和矩阵环。以下是其中一些的构造:

```
sage: GF(3)
Finite Field of size 3
sage: GF(27, 'a')  # need to name the generator if not a prime field
Finite Field in a of size 3^3
sage: Zp(5)
5-adic Ring with capped relative precision 20
sage: sqrt(3) in QQbar # algebraic closure of QQ
True
```

2.8 线性代数

Sage 提供了线性代数中的标准构造，例如矩阵的特征多项式、阶梯形、迹、分解等。

创建矩阵和进行矩阵乘法非常简单自然：

```
sage: A = Matrix([[1,2,3],[3,2,1],[1,1,1]])
sage: w = vector([1,1,-4])
sage: w*A
(0, 0, 0)
sage: A*w
(-9, 1, -2)
sage: kernel(A)
Free module of degree 3 and rank 1 over Integer Ring
Echelon basis matrix:
[ 1  1 -4]
```

请注意，在 Sage 中，矩阵 A 的核是“左核”，即满足 $wA = 0$ 的向量空间 w。

求解矩阵方程非常简单，使用 `solve_right` 方法即可。运行 `A.solve_right(Y)` 将返回一个矩阵（或向量）X，使得 $AX = Y$：

```
sage: Y = vector([0, -4, -1])
sage: X = A.solve_right(Y)
sage: X
(-2, 1, 0)
sage: A * X   # checking our answer...
(0, -4, -1)
```

倘若无解，Sage 会返回错误：

```
sage: A.solve_right(w)
Traceback (most recent call last):
...
ValueError: matrix equation has no solutions
```

同理，可以使用 `A.solve_left(Y)` 来求解方程:$XA = Y$ 中的 X。

Sage 还可以计算特征值和特征向量:

```
sage: A = matrix([[0, 4], [-1, 0]])
sage: A.eigenvalues ()
[-2*I, 2*I]
sage: B = matrix([[1, 3], [3, 1]])
sage: B.eigenvectors_left()
[(4, [(1, 1)], 1), (-2, [(1, -1)], 1)]
```

（`eigenvectors_left` 的输出格式是一个包含三元组（特征值、特征向量、重数）的列表。）特征值和特征向量可以通过 Maxima 在有理数域 `QQ` 或实数域 `RR` 上计算（见下文的*Maxima*）。

如基本环 所述，矩阵定义的环会影响其某些性质。在下面的示例中，`matrix` 命令的第一个参数告诉 Sage 将矩阵视为整数矩阵（`ZZ`）、有理数矩阵（`QQ`）或实数矩阵（`RR`）

```
sage: AZ = matrix(ZZ, [[2,0], [0,1]])
sage: AQ = matrix(QQ, [[2,0], [0,1]])
sage: AR = matrix(RR, [[2,0], [0,1]])
sage: AZ.echelon_form()
[2 0]
[0 1]
```

(续下页)

(接上页)

```
sage: AQ.echelon_form()
[1 0]
[0 1]
sage: AR.echelon_form()
[ 1.00000000000000 0.000000000000000]
[0.000000000000000  1.00000000000000]
```

如果要计算浮点实数或复数矩阵的特征值和特征向量，矩阵应分别定义在 RDF （实双精度域）或 CDF （复双精度域）上。如果没有指定环并且使用浮点实数或复数，则默认情况下矩阵定义在 RR 或 CC 域上，这些域不支持所有情况的这些计算:

```
sage: ARDF = matrix(RDF, [[1.2, 2], [2, 3]])
sage: ARDF.eigenvalues()  # rel tol 8e-16
[-0.09317121994613098, 4.293171219946131]
sage: ACDF = matrix(CDF, [[1.2, I], [2, 3]])
sage: ACDF.eigenvectors_right()  # rel tol 3e-15
[(0.8818456983293743 - 0.8209140653434135*I, [(0.7505608183809549, -0.616145932704589 + 0.
↪2387941530333261*I)], 1),
(3.3181543016706256 + 0.8209140653434133*I, [(0.14559469829270957 + 0.3756690858502104*I, 0.
↪9152458258662108)], 1)]
```

2.8.1 矩阵空间

我们创建了一个定义在有理数域 $\mathbf{Q}$ 上的 3×3 矩阵空间 $\mathrm{Mat}_{3\times 3}(\mathbf{Q})$:

```
sage: M = MatrixSpace(QQ,3)
sage: M
Full MatrixSpace of 3 by 3 dense matrices over Rational Field
```

（要创建一个 3×4 矩阵空间，可以使用 `MatrixSpace(QQ,3,4)`。如果省略列数，则默认为行数，因此 `MatrixSpace(QQ,3)` 与 `MatrixSpace(QQ,3,3)` 意义相同。）矩阵空间有其规范基:

```
sage: B = M.basis()
sage: len(B)
9
sage: B[0,1]
[0 1 0]
[0 0 0]
[0 0 0]
```

我们创建一个矩阵作为 `M` 的元素。

```
sage: A = M(range(9)); A
[0 1 2]
[3 4 5]
[6 7 8]
```

接下来我们计算其简化行阶梯形和核。

```
sage: A.echelon_form()
[ 1  0 -1]
[ 0  1  2]
[ 0  0  0]
sage: A.kernel()
Vector space of degree 3 and dimension 1 over Rational Field
```

(续下页)

(接上页)

```
Basis matrix:
[ 1 -2  1]
```

接着我们来演示在有限域上定义的矩阵的计算：

```
sage: M = MatrixSpace(GF(2),4,8)
sage: A = M([1,1,0,0, 1,1,1,1, 0,1,0,0, 1,0,1,1,
....:        0,0,1,0, 1,1,0,1, 0,0,1,1, 1,1,1,0])
sage: A
[1 1 0 0 1 1 1 1]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 1 1 1 1 1 0]
sage: rows = A.rows()
sage: A.columns()
[(1, 0, 0, 0), (1, 1, 0, 0), (0, 0, 1, 1), (0, 0, 0, 1),
 (1, 1, 1, 1), (1, 0, 1, 1), (1, 1, 0, 1), (1, 1, 1, 0)]
sage: rows
[(1, 1, 0, 0, 1, 1, 1, 1), (0, 1, 0, 0, 1, 0, 1, 1),
 (0, 0, 1, 0, 1, 1, 0, 1), (0, 0, 1, 1, 1, 1, 1, 0)]
```

我们创建一个在有限域 $\mathbf{F}_2$ 上由上述行生成的子空间。

```
sage: V = VectorSpace(GF(2),8)
sage: S = V.subspace(rows)
sage: S
Vector space of degree 8 and dimension 4 over Finite Field of size 2
Basis matrix:
[1 0 0 0 0 1 0 0]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 0 1 0 0 1 1]
sage: A.echelon_form()
[1 0 0 0 0 1 0 0]
[0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1]
[0 0 0 1 0 0 1 1]
```

Sage 使用的 S 的基是通过生成矩阵的简化行阶梯形的非零行获得的。

2.8.2 稀疏线性代数

Sage 支持在主理想域 (PIDs) 上的稀疏线性代数。

```
sage: M = MatrixSpace(QQ, 100, sparse=True)
sage: A = M.random_element(density = 0.05)
sage: E = A.echelon_form()
```

Sage 中的多模算法适用于方阵（但不适用于非方阵）：

```
sage: M = MatrixSpace(QQ, 50, 100, sparse=True)
sage: A = M.random_element(density = 0.05)
sage: E = A.echelon_form()
sage: M = MatrixSpace(GF(2), 20, 40, sparse=True)
sage: A = M.random_element()
sage: E = A.echelon_form()
```

请注意，Python 是区分大小写的：

```
sage: M = MatrixSpace(QQ, 10,10, Sparse=True)
Traceback (most recent call last):
...
TypeError: ...__init__() got an unexpected keyword argument 'Sparse'...
```

2.9 多项式

在本节中，我们将介绍如何在 Sage 中创建和使用多项式。

2.9.1 单变量多项式

创建多项式环有三种方法。

```
sage: R = PolynomialRing(QQ, 't')
sage: R
Univariate Polynomial Ring in t over Rational Field
```

这会创建一个多项式环，并告诉 Sage 在显示时使用字符串't' 作为不定元。然而，这并没有定义符号 `t`，因此你不能用它来输入属于 `R` 的多项式（例如 t^2+1）。

另一种方法是

```
sage: S = QQ['t']
sage: S == R
True
```

这样做对于 `t` 也存在同样的问题。

第三种非常方便的方法是

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

或

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

甚至

```
sage: R.<t> = QQ[]
```

这样做还有一个额外的好处，即它定义了变量 `t` 作为多项式环的不定元，因此你可以轻松地构造 `R` 的元素，如下所示。（请注意，第三种方法与 Magma 中的构造符号非常相似，并且可以像在 Magma 中一样用于广泛的对象。）

```
sage: poly = (t+1) * (t+2); poly
t^2 + 3*t + 2
sage: poly in R
True
```

无论你使用哪种方法定义多项式环，你都可以通过 0^{th} 生成器恢复不定元：

```
sage: R = PolynomialRing(QQ, 't')
sage: t = R.0
sage: t in R
True
```

请注意，类似的构造方法适用于复数：复数可以被视为由符号 `i` 在实数上生成的，因此我们有以下内容：

```
sage: CC
Complex Field with 53 bits of precision
sage: CC.0  # 0th generator of CC
1.00000000000000*I
```

对于多项式环，你可以在创建环时同时获得环及其生成器，或者仅获得生成器，如下所示：

```
sage: R, t = QQ['t'].objgen()
sage: t    = QQ['t'].gen()
sage: R, t = objgen(QQ['t'])
sage: t    = gen(QQ['t'])
```

最后我们在 $\mathbf{Q}[t]$ 中进行一些算术运算。

```
sage: R, t = QQ['t'].objgen()
sage: f = 2*t^7 + 3*t^2 - 15/19
sage: f^2
4*t^14 + 12*t^9 - 60/19*t^7 + 9*t^4 - 90/19*t^2 + 225/361
sage: cyclo = R.cyclotomic_polynomial(7); cyclo
t^6 + t^5 + t^4 + t^3 + t^2 + t + 1
sage: g = 7 * cyclo * t^5 * (t^5 + 10*t + 2)
sage: g
7*t^16 + 7*t^15 + 7*t^14 + 7*t^13 + 77*t^12 + 91*t^11 + 91*t^10 + 84*t^9
       + 84*t^8 + 84*t^7 + 84*t^6 + 14*t^5
sage: F = factor(g); F
(7) * t^5 * (t^5 + 10*t + 2) * (t^6 + t^5 + t^4 + t^3 + t^2 + t + 1)
sage: F.unit()
7
sage: list(F)
[(t, 5), (t^5 + 10*t + 2, 1), (t^6 + t^5 + t^4 + t^3 + t^2 + t + 1, 1)]
```

注意，因式分解正确考虑并记录了单位部分。

如果你在某个研究项目中大量使用某个函数，例如 `R.cyclotomic_polynomial`，除了引用 Sage 之外，你还应该尝试找出 Sage 的哪个组件在实际计算分圆多项式并引用它。在这种情况下，如果你输入 `R.cyclotomic_polynomial??` 查看源代码，你很快会看到一行 `f = pari.polcyclo(n)`，这意味着 PARI 被用于计算分圆多项式。你的作品中也需要引用 PARI。

除以两个多项式会构造分数域的元素（Sage 会自动创建）。

```
sage: x = QQ['x'].0
sage: f = x^3 + 1; g = x^2 - 17
sage: h = f/g;  h
(x^3 + 1)/(x^2 - 17)
sage: h.parent()
Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

使用 Laurent 级数，可以在 `QQ[x]` 的分数域中计算级数展开：

```
sage: R.<x> = LaurentSeriesRing(QQ); R
Laurent Series Ring in x over Rational Field
sage: 1/(1-x) + O(x^10)
1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + O(x^10)
```

如果我们给变量不同的命名，我们会得到不同的单变量多项式环。

```
sage: R.<x> = PolynomialRing(QQ)
sage: S.<y> = PolynomialRing(QQ)
sage: x == y
False
sage: R == S
False
sage: R(y)
x
sage: R(y^2 - 17)
x^2 - 17
```

环由变量决定。请注意，使用名为 x 的变量创建另一个环不会返回不同的环。

```
sage: R = PolynomialRing(QQ, "x")
sage: T = PolynomialRing(QQ, "x")
sage: R == T
True
sage: R is T
True
sage: R.0 == T.0
True
```

Sage 还支持任意基环上的幂级数和 Laurent 级数环。在下面的示例中，我们创建了 $\mathbf{F}_7[[T]]$ 的一个元素，并通过相除创建 $\mathbf{F}_7((T))$ 的一个元素。

```
sage: R.<T> = PowerSeriesRing(GF(7)); R
Power Series Ring in T over Finite Field of size 7
sage: f = T  + 3*T^2 + T^3 + O(T^4)
sage: f^3
T^3 + 2*T^4 + 2*T^5 + O(T^6)
sage: 1/f
T^-1 + 4 + T + O(T^2)
sage: parent(1/f)
Laurent Series Ring in T over Finite Field of size 7
```

你也可以使用双括号简写来创建幂级数环：

```
sage: GF(7)[['T']]
Power Series Ring in T over Finite Field of size 7
```

2.9.2 多变量多项式

要处理多个变量的多项式，我们首先声明多项式环和变量。

```
sage: R = PolynomialRing(GF(5),3,"z") # here, 3 = number of variables
sage: R
Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5
```

与定义单变量多项式环一样，有多种方法：

```
sage: GF(5)['z0, z1, z2']
Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5
sage: R.<z0,z1,z2> = GF(5)[]; R
Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5
```

此外，如果你想让变量名为单个字母，你可以使用以下简写：

```
sage: PolynomialRing(GF(5), 3, 'xyz')
Multivariate Polynomial Ring in x, y, z over Finite Field of size 5
```

接下来让我们进行一些算术运算。

```
sage: z = GF(5)['z0, z1, z2'].gens()
sage: z
(z0, z1, z2)
sage: (z[0]+z[1]+z[2])^2
z0^2 + 2*z0*z1 + z1^2 + 2*z0*z2 + 2*z1*z2 + z2^2
```

你还可以使用更多数学符号来构造多项式环。

```
sage: R = GF(5)['x,y,z']
sage: x,y,z = R.gens()
sage: QQ['x']
Univariate Polynomial Ring in x over Rational Field
sage: QQ['x,y'].gens()
(x, y)
sage: QQ['x'].objgens()
(Univariate Polynomial Ring in x over Rational Field, (x,))
```

多变量多项式在 Sage 中使用 Python 字典和多项式的“分配表示”实现。Sage 使用了一些 Singular [Si]，例如，用于计算理想的最大公约数和 Gröbner 基。

```
sage: R, (x, y) = PolynomialRing(RationalField(), 2, 'xy').objgens()
sage: f = (x^3 + 2*y^2*x)^2
sage: g = x^2*y^2
sage: f.gcd(g)
x^2
```

接下来我们通过简单地将 `(f,g)` 乘以 `R` 来创建由 f 和 g 生成的理想 (f, g)，（也可以写做 `ideal([f,g])` 或 `ideal(f,g)`）。

```
sage: I = (f, g)*R; I
Ideal (x^6 + 4*x^4*y^2 + 4*x^2*y^4, x^2*y^2) of Multivariate Polynomial
Ring in x, y over Rational Field
sage: B = I.groebner_basis(); B
[x^6, x^2*y^2]
sage: x^2 in I
False
```

顺便说一句，上面的 Gröbner 基不是一个列表，而是一个不可变序列。这意味着它有全集，父结构，并且不可更改（这是好的，因为更改基会破坏使用 Gröbner 基的其他例程）。

```
sage: B.universe()
Multivariate Polynomial Ring in x, y over Rational Field
sage: B[1] = x
Traceback (most recent call last):
...
ValueError: object is immutable; please change a copy instead.
```

Sage 中有一些（没有我们想要的那么多）交换代数可用，通过 Singular 实现。例如，我们可以计算 I 的初等分解和相关素数：

```
sage: I.primary_decomposition()
[Ideal (x^2) of Multivariate Polynomial Ring in x, y over Rational Field,
```

(续下页)

(接上页)

```
 Ideal (y^2, x^6) of Multivariate Polynomial Ring in x, y over Rational Field]
sage: I.associated_primes()
[Ideal (x) of Multivariate Polynomial Ring in x, y over Rational Field,
 Ideal (y, x) of Multivariate Polynomial Ring in x, y over Rational Field]
```

2.10 父结构、转换与强制转换

这一节可能比前一节更技术化，但为了有效且高效地使用 Sage 中的环和其他代数结构，理解父结构和强制转换的意义非常重要。

请注意，我们在这里只解释概念，不展示具体实现。面向实现的教程可以参见 Sage thematic tutorial 。

2.10.1 元素

如果想在 Python 中实现一个环，第一步是创建一个类来表示该环的元素 `x`，并为其提供必要的双下划线方法，例如 `__add__`, `__sub__`, `__mul__`，同时确保环公理成立。

由于 Python 是一种强类型（但动态类型）语言，可能会想到为每个环实现一个 Python 类。毕竟，Python 有整数类型 `<int>` 和实数类型 `<float>` 等等。但这种方法很快就会失败：环的数量是无限的，无法实现无限多个类。

相反，可以创建一个类层次结构来实现常见的代数结构元素，例如群、环、斜域、交换环、域、代数等等。

但这意味着不同环的元素可以具有相同的类型。

```
sage: P.<x,y> = GF(3)[]
sage: Q.<a,b> = GF(4,'z')[]
sage: type(x)==type(a)
True
```

另一方面，也可以有不同的 Python 类来实现相同的数学结构（例如稠密矩阵与稀疏矩阵）

```
sage: P.<a> = PolynomialRing(ZZ)
sage: Q.<b> = PolynomialRing(ZZ, sparse=True)
sage: R.<c> = PolynomialRing(ZZ, implementation='NTL')
sage: type(a); type(b); type(c)
<class 'sage.rings.polynomial.polynomial_integer_dense_flint.Polynomial_integer_dense_flint'>
<class 'sage.rings.polynomial.polynomial_ring.PolynomialRing_integral_domain_with_category.
↪element_class'>
<class 'sage.rings.polynomial.polynomial_integer_dense_ntl.Polynomial_integer_dense_ntl'>
```

这带来了两个问题：一方面，如果两个元素是相同类的实例，可以预期它们的 `__add__` 方法能够相加；但如果这些元素属于非常不同的环，则不希望如此。另一方面，如果两个元素属于同一环的不同实现，想要相加，但如果它们属于不同的 Python 类，这并不容易实现。

解决这些问题的方法称为“强制转换”，将在下面解释。

然而，每个元素都必须知道它属于哪个父结构。这可以通过 `parent()` 方法获得：

```
sage: a.parent(); b.parent(); c.parent()
Univariate Polynomial Ring in a over Integer Ring
Sparse Univariate Polynomial Ring in b over Integer Ring
Univariate Polynomial Ring in c over Integer Ring (using NTL)
```

2.10.2 父结构与范畴

与代数结构元素的 Python 类层次结构类似，Sage 也提供包含这些元素的代数结构的类。在 Sage 中包含元素的结构称为“父结构”，并且有一个基类。大致上与数学概念的层次结构一致，有一系列类，例如集合、环、域等等：

```
sage: isinstance(QQ,Field)
True
sage: isinstance(QQ, Ring)
True
sage: isinstance(ZZ,Field)
False
sage: isinstance(ZZ, Ring)
True
```

在代数中，共享相同代数结构的对象被归类到所谓的“范畴”中。因此，Sage 中类层次结构与范畴层次结构之间有一个粗略的类比。然而，不应过分强调 Python 类与范畴的类比。毕竟，数学范畴也在 Sage 中实现：

```
sage: Rings()
Category of rings
sage: ZZ.category()
Join of Category of Dedekind domains
    and Category of euclidean domains
    and Category of noetherian rings
    and Category of infinite enumerated sets
    and Category of metric spaces
sage: ZZ.category().is_subcategory(Rings())
True
sage: ZZ in Rings()
True
sage: ZZ in Fields()
False
sage: QQ in Fields()
True
```

虽然 Sage 的类层次结构集中在实现细节上，但 Sage 的范畴框架更集中在数学结构上。可以在范畴中实现不依赖具体实现的通用方法和测试。

Sage 中的父结构应该是唯一的 Python 对象。例如，一旦创建了一个具有特定基环和特定生成器列表的多项式环，结果将被缓存：

```
sage: RR['x','y'] is RR['x','y']
True
```

2.10.3 类型与父结构

类型 `RingElement` 并不完全对应于数学概念中的环元素。例如，虽然方阵属于一个环，但它们不是 `RingElement` 的实例：

```
sage: M = Matrix(ZZ,2,2); M
[0 0]
[0 0]
sage: isinstance(M, RingElement)
False
```

虽然在 Sage 中父结构是唯一的，但在一个父结构中的相等元素不一定是相同的。这与 Python 对某些（虽然不是全部）整数的行为形成对比：

```
sage: int(1) is int(1) # Python int
True
sage: int(-15) is int(-15)
False
sage: 1 is 1           # Sage Integer
False
```

不同环的元素通常不是通过它们的类型区分，而是通过它们的父结构区分：

```
sage: a = GF(2)(1)
sage: b = GF(5)(1)
sage: type(a) is type(b)
True
sage: parent(a)
Finite Field of size 2
sage: parent(b)
Finite Field of size 5
```

因此，从代数的角度来看，**元素的父结构比它的类型更重要**。

2.10.4 转换与强制转换

在某些情况下，可以将一个父结构的元素转换为另一个父结构的元素。这样的转换可以是显式的也可以是隐式的（被称为 *强制转换*）。

读者可能知道例如 C 语言中的 *类型转换* 和 *类型强制转换* 的概念。Sage 中也有转换和强制转换的概念。但 Sage 中的概念集中在 *父结构* 上，而不是类型上。所以请不要将 C 语言中的类型转换与 Sage 中的转换混淆！

我们在这里给出一个相当简短的说明。详细描述和实现信息，请参阅参考手册中的强制转换章节以及 thematic tutorial.

关于在 *不同环的元素* 上进行算术运算的可能性，有两种极端观点：

- 不同的环是不同的世界，对不同环的元素进行加法或乘法没有任何意义；即使 `1 + 1/2` 也没有意义，因为第一个加数是整数，第二个是有理数。

或者

- 如果一个环 `R1` 的元素 `r1` 可以以某种方式在另一个环 `R2` 中解释，那么所有涉及 `r1` 和任意 `R2` 元素的算术运算都是允许的。乘法单位存在于所有域和许多环，它们应该都是相等的。

Sage 选择了一种折衷方案。如果 `P1` 和 `P2` 是父结构，`p1` 是 `P1` 的元素，那么用户可以显式请求将 `p1` 在 `P2` 中解释。这在所有情况下可能没有意义，或者对于 `P1` 的所有元素都没有定义，用户需要确保其合理性。我们称之为 **转换**：

```
sage: a = GF(2)(1)
sage: b = GF(5)(1)
sage: GF(5)(a) == b
True
sage: GF(2)(b) == a
True
```

然而，只有当这种转换可以彻底和一致地完成时，才会发生 *隐式*（或自动）转换。数学的严谨性在这一点上至关重要。

这种隐式转换称为 **强制转换**。如果定义了强制转换，那么它必须与转换一致。定义强制转换需要满足两个条件：

1. 从 `P1` 到 `P2` 的强制转换必须由结构保持映射给出（例如环同态）。仅仅一些 `P1` 的元素可以映射到 `P2` 是不够的，映射必须尊重 `P1` 的代数结构。

2. 这些强制转换映射的选择必须一致：如果 P3 是第三个父结构，那么从 P1 到 P2 的选定强制转换与从 P2 到 P3 的强制转换的组合必须与从 P1 到 P3 的选定强制转换一致。特别是，如果存在从 P1 到 P2 和从 P2 到 P1 的强制转换，则组合必须是 P1 的恒等映射。

因此，尽管可以将 GF(2) 的每个元素转换为 GF(5)，但不能强制转换，因为 GF(2) 和 GF(5) 之间没有环同态。

一致性方面更难解释。我们用多元多项式环来说明。在应用中，保留名称的强制转换最有意义。因此，我们有：

```
sage: R1.<x,y> = ZZ[]
sage: R2 = ZZ['y','x']
sage: R2.has_coerce_map_from(R1)
True
sage: R2(x)
x
sage: R2(y)
y
sage: R2.coerce(y)
y
```

如果没有保留名称的环同态，则不定义强制转换。然而，转换可能仍然是可能的，即通过根据生成器列表中的位置映射环生成器：

```
sage: R3 = ZZ['z','x']
sage: R3.has_coerce_map_from(R1)
False
sage: R3(x)
z
sage: R3(y)
x
sage: R3.coerce(y)
Traceback (most recent call last):
...
TypeError: no canonical coercion
from Multivariate Polynomial Ring in x, y over Integer Ring
to Multivariate Polynomial Ring in z, x over Integer Ring
```

但这种保留位置的转换不符合强制转换：通过组合从 ZZ['x','y'] 到 ZZ['y','x'] 的保留名称映射与从 ZZ['y','x'] 到 ZZ['a','b'] 的保留位置映射，将得到一个既不保留名称也不保留位置的映射，违反了一致性。

如果存在强制转换，它将用于比较不同环的元素或进行算术运算。这通常很方便，但用户应该意识将 == 关系扩展到不同父结构的边界可能很容易导致过度使用。例如，虽然 == 应该是 **同**一环元素上的等价关系，但如果涉及 不同环，则不一定如此。例如，ZZ 和有限域中的 1 被认为是相等的，因为从整数到任何有限域都有一个规范的强制转换。然而，通常两个不同的有限域之间没有强制转换。因此我们有：

```
sage: GF(5)(1) == 1
True
sage: 1 == GF(2)(1)
True
sage: GF(5)(1) == GF(2)(1)
False
sage: GF(5)(1) != GF(2)(1)
True
```

同理，我们有：

```
sage: R3(R1.1) == R3.1
True
sage: R1.1 == R3.1
False
sage: R1.1 != R3.1
True
```

一致性条件的另一个结果是强制转换只能从精确环（例如有理数 QQ）到不精确环（例如具有固定精度的实数 RR），而不能反过来。原因是从 QQ 到 RR 的强制转换与从 RR 到 QQ 的转换的组合应该是 QQ 上的恒等映射。但这是不可能的，因为在 RR 中一些不同的有理数可能被视为相等，如下例所示：

```
sage: RR(1/10^200+1/10^100) == RR(1/10^100)
True
sage: 1/10^200+1/10^100 == 1/10^100
False
```

当比较两个父结构 P1 和 P2 的元素时，可能没有两个环之间的强制转换，但有一个规范的父结构 P3 可选，使得 P1 和 P2 都强制转换到 P3。在这种情况下，也会发生强制转换。一个典型用例是有理数和具有整数系数的多项式之和，产生具有有理系数的多项式：

```
sage: P1.<x> = ZZ[]
sage: p = 2*x+3
sage: q = 1/2
sage: parent(p)
Univariate Polynomial Ring in x over Integer Ring
sage: parent(p+q)
Univariate Polynomial Ring in x over Rational Field
```

注意，原则上结果在 ZZ['x'] 的分数域中也有意义。然而，Sage 会尝试选择一个 规范的共同父结构，使得看起来最自然（在我们的例子中是 QQ['x']）。如果几个潜在的共同父结构看起来同样自然，为了获得可靠的结果，Sage 不会随机选择其中一个。该选择所基于的机制在 thematic tutorial 中进行了解释。

以下示例不会发生强制转换到共同父结构：

```
sage: R.<x> = QQ[]
sage: S.<y> = QQ[]
sage: x+y
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for +: 'Univariate Polynomial Ring in x over Rational␣
↪Field' and 'Univariate Polynomial Ring in y over Rational Field'
```

原因是 Sage 不会选择潜在候选结构 QQ['x']['y'], QQ['y']['x'], QQ['x','y'] 或 QQ['y','x'] 之一，因为所有这四个成对不同的结构看起来都是自然的共同父结构，并且没有明显的规范选择。

2.11 有限群与阿贝尔群

Sage 支持置换群、有限经典群（例如 $SU(n,q)$）、有限矩阵群（使用自定义生成器）和阿贝尔群（包括无限群）的计算。这些功能大部分是通过 GAP 接口实现的。

例如，要创建一个置换群，可以提供一个生成器列表，如下例所示：

```
sage: G = PermutationGroup(['(1,2,3)(4,5)', '(3,4)'])
sage: G
Permutation Group with generators [(3,4), (1,2,3)(4,5)]
sage: G.order()
```

(续下页)

(接上页)

```
120
sage: G.is_abelian()
False
sage: G.derived_series()           # random-ish output
[Permutation Group with generators [(1,2,3)(4,5), (3,4)],
 Permutation Group with generators [(1,5)(3,4), (1,5)(2,4), (1,3,5)]]
sage: G.center()
Subgroup generated by [()] of (Permutation Group with generators [(3,4), (1,2,3)(4,5)])
sage: G.random_element()           # random output
(1,5,3)(2,4)
sage: print(latex(G))
\langle (3,4), (1,2,3)(4,5) \rangle
```

在 Sage 中，你还可以获得特征表（LaTeX 格式）：

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

此外，Sage 还支持有限域上的经典群和矩阵群：

```
sage: MS = MatrixSpace(GF(7), 2)
sage: gens = [MS([[1,0],[-1,1]]),MS([[1,1],[0,1]])]
sage: G = MatrixGroup(gens)
sage: G.conjugacy_classes_representatives()
(
[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]
)
sage: G = Sp(4,GF(7))
sage: G
Symplectic Group of degree 4 over Finite Field of size 7
sage: G.random_element()             # random output
[5 5 5 1]
[0 2 6 3]
[5 0 1 0]
[4 6 3 4]
sage: G.order()
276595200
```

你还可以计算阿贝尔群（包括有限群和无限群）：

```
sage: F = AbelianGroup(5, [5,5,7,8,9], names='abcde')
sage: (a, b, c, d, e) = F.gens()
sage: d * b**2 * c**3
b^2*c^3*d
sage: F = AbelianGroup(3,[2]*3); F
Multiplicative Abelian group isomorphic to C2 x C2 x C2
sage: H = AbelianGroup([2,3], names="xy"); H
```

(续下页)

(接上页)

```
Multiplicative Abelian group isomorphic to C2 x C3
sage: AbelianGroup(5)
Multiplicative Abelian group isomorphic to Z x Z x Z x Z x Z
sage: AbelianGroup(5).order()
+Infinity
```

2.12 数论

Sage 具有丰富的数论功能。例如，我们可以在 $\mathbf{Z}/N\mathbf{Z}$ 中进行算术运算：

```
sage: R = IntegerModRing(97)
sage: a = R(2) / R(3)
sage: a
33
sage: a.rational_reconstruction()
2/3
sage: b = R(47)
sage: b^20052005
50
sage: b.modulus()
97
sage: b.is_square()
True
```

Sage 包含标准的数论函数，例如：

```
sage: gcd(515,2005)
5
sage: factor(2005)
5 * 401
sage: c = factorial(25); c
15511210043330985984000000
sage: [valuation(c,p) for p in prime_range(2,23)]
[22, 10, 6, 3, 2, 1, 1, 1]
sage: next_prime(2005)
2011
sage: previous_prime(2005)
2003
sage: divisors(28); sum(divisors(28)); 2*28
[1, 2, 4, 7, 14, 28]
56
56
```

完美！

Sage 的 `sigma(n,k)` 函数累加 `n` 的除数的 k 次幂：

```
sage: sigma(28,0); sigma(28,1); sigma(28,2)
6
56
1050
```

下面展示扩展的欧几里得算法、欧拉 ϕ 函数和中国剩余定理：

```
sage: d,u,v = xgcd(12,15)
sage: d == u*12 + v*15
True
sage: n = 2005
sage: inverse_mod(3,n)
1337
sage: 3 * 1337
4011
sage: prime_divisors(n)
[5, 401]
sage: phi = n*prod([1 - 1/p for p in prime_divisors(n)]); phi
1600
sage: euler_phi(n)
1600
sage: prime_to_m_part(n, 5)
401
```

接下来验证有关 $3n+1$ 问题的一些内容：

```
sage: n = 2005
sage: for i in range(1000):
....:     n = 3*odd_part(n) + 1
....:     if odd_part(n)==1:
....:         print(i)
....:         break
38
```

最后，展示中国剩余定理：

```
sage: x = crt(2, 1, 3, 5); x
11
sage: x % 3  # x mod 3 = 2
2
sage: x % 5  # x mod 5 = 1
1
sage: [binomial(13,m) for m in range(14)]
[1, 13, 78, 286, 715, 1287, 1716, 1716, 1287, 715, 286, 78, 13, 1]
sage: [binomial(13,m)%2 for m in range(14)]
[1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1]
sage: [kronecker(m,13) for m in range(1,13)]
[1, -1, 1, 1, -1, -1, -1, -1, 1, 1, -1, 1]
sage: n = 10000; sum([moebius(m) for m in range(1,n)])
-23
sage: Partitions(4).list()
[[4], [3, 1], [2, 2], [2, 1, 1], [1, 1, 1, 1]]
```

2.12.1 p-进数

Sage 中实现了 p-进数域。注意，一旦创建了 p-进数域，就不能改变其精度。

```
sage: K = Qp(11); K
11-adic Field with capped relative precision 20
sage: a = K(211/17); a
4 + 4*11 + 11^2 + 7*11^3 + 9*11^5 + 5*11^6 + 4*11^7 + 8*11^8 + 7*11^9
  + 9*11^10 + 3*11^11 + 10*11^12 + 11^13 + 5*11^14 + 6*11^15 + 2*11^16
  + 3*11^17 + 11^18 + 7*11^19 + O(11^20)
```

(续下页)

(接上页)

```
sage: b = K(3211/11^2); b
10*11^-2 + 5*11^-1 + 4 + 2*11 + O(11^18)
```

在 *p*-进数域和数域中实现整数环已经做了大量工作。感兴趣的读者可以阅读 sage.rings.padics.tutorial，并向 `sage-support` Google 讨论组的专家咨询更多详细信息。

在 NumberField 类中已经实现了许多相关方法。

```
sage: R.<x> = PolynomialRing(QQ)
sage: K = NumberField(x^3 + x^2 - 2*x + 8, 'a')
sage: K.integral_basis()
[1, 1/2*a^2 + 1/2*a, a^2]
```

```
sage: K.galois_group()
Galois group 3T2 (S3) with order 6 of x^3 + x^2 - 2*x + 8
```

```
sage: K.polynomial_quotient_ring()
Univariate Quotient Polynomial Ring in a over Rational Field with modulus
x^3 + x^2 - 2*x + 8
sage: K.units()
(-3*a^2 - 13*a - 13,)
sage: K.discriminant()
-503
sage: K.class_group()
Class group of order 1 of Number Field in a with
defining polynomial x^3 + x^2 - 2*x + 8
sage: K.class_number()
1
```

2.13 一些更高级的数学

2.13.1 代数几何

在 Sage 中可以定义任意代数簇，但有时复杂的功能仅限于在 **Q** 或有限域上的环。例如，我们可以计算两个仿射平面曲线的并集，然后将曲线恢复成该并集的不可约分量。

```
sage: x, y = AffineSpace(2, QQ, 'xy').gens()
sage: C2 = Curve(x^2 + y^2 - 1)
sage: C3 = Curve(x^3 + y^3 - 1)
sage: D = C2 + C3
sage: D
Affine Plane Curve over Rational Field defined by
   x^5 + x^3*y^2 + x^2*y^3 + y^5 - x^3 - y^3 - x^2 - y^2 + 1
sage: D.irreducible_components()
[Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
   x^2 + y^2 - 1,
 Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
   x^3 + y^3 - 1]
```

我们还可以通过相交这两条曲线并计算其不可约分量来找到它们的所有交点。

```
sage: V = C2.intersection(C3)
sage: V.irreducible_components()
[Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
```

(续下页)

(接上页)

```
  y - 1,
  x,
 Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  y,
  x - 1,
 Closed subscheme of Affine Space of dimension 2 over Rational Field defined by:
  x + y + 2,
  2*y^2 + 4*y + 3]
```

例如，$(1,0)$ 和 $(0,1)$ 都在两条曲线上（显而易见），还有一些（二次）点，它们的 y 坐标满足 $2y^2+4y+3=0$。

Sage 可以计算三维射影空间中扭曲三次曲线的环理想：

```
sage: R.<a,b,c,d> = PolynomialRing(QQ, 4)
sage: I = ideal(b^2-a*c, c^2-b*d, a*d-b*c)
sage: F = I.groebner_fan(); F
Groebner fan of the ideal:
Ideal (b^2 - a*c, c^2 - b*d, -b*c + a*d) of Multivariate Polynomial Ring
in a, b, c, d over Rational Field
sage: F.reduced_groebner_bases ()
[[-c^2 + b*d, -b*c + a*d, -b^2 + a*c],
 [-b*c + a*d, -c^2 + b*d, b^2 - a*c],
 [-c^3 + a*d^2, -c^2 + b*d, b*c - a*d, b^2 - a*c],
 [-c^2 + b*d, b^2 - a*c, b*c - a*d, c^3 - a*d^2],
 [-b*c + a*d, -b^2 + a*c, c^2 - b*d],
 [-b^3 + a^2*d, -b^2 + a*c, c^2 - b*d, b*c - a*d],
 [-b^2 + a*c, c^2 - b*d, b*c - a*d, b^3 - a^2*d],
 [c^2 - b*d, b*c - a*d, b^2 - a*c]]
sage: F.polyhedralfan()
Polyhedral fan in 4 dimensions of dimension 4
```

2.13.2 椭圆曲线

Sage 的椭圆曲线功能包括 PARI 的大部分椭圆曲线功能、访问 Cremona 在线表中的数据（需要可选数据库包）、mwrank 功能（即计算全 Mordell-Weil 群的 2 次下降）、SEA 算法、计算所有同源、许多关于 **Q** 上曲线的新代码，以及 Denis Simon 的一些代数下降软件。

创建椭圆曲线的命令 `EllipticCurve` 有多种形式：

- EllipticCurve($[a_1, a_2, a_3, a_4, a_6]$): 返回如下椭圆曲线

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

  其中 a_i 被转换为 a_1 的父结构。如果所有 a_i 的父结构都是 **Z**，它们将被转换为 **Q**。

- EllipticCurve($[a_4, a_6]$): 与上面相同，但 $a_1 = a_2 = a_3 = 0$。
- EllipticCurve(label): 返回来自 Cremona 数据库的椭圆曲线，使用给定的（新的!）Cremona 标签。标签是一个字符串，例如 `"11a"` 或 `"37b2"`。字母必须是小写（以区分旧标签）。
- EllipticCurve(j): 返回具有 j-不变量 j 的椭圆曲线。
- EllipticCurve(R, $[a_1, a_2, a_3, a_4, a_6]$): 创建定义在环 R 上的椭圆曲线，给定的 a_i 同上。

我们将展示每一个构造函数：

```
sage: EllipticCurve([0,0,1,-1,0])
Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field
```

(续下页)

(接上页)

```
sage: EllipticCurve([GF(5)(0),0,1,-1,0])
Elliptic Curve defined by y^2 + y = x^3 + 4*x over Finite Field of size 5

sage: EllipticCurve([1,2])
Elliptic Curve defined by y^2  = x^3 + x + 2 over Rational Field

sage: EllipticCurve('37a')
Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field

sage: EllipticCurve_from_j(1)
Elliptic Curve defined by y^2 + x*y = x^3 + 36*x + 3455 over Rational Field

sage: EllipticCurve(GF(5), [0,0,1,-1,0])
Elliptic Curve defined by y^2 + y = x^3 + 4*x over Finite Field of size 5
```

点 $(0,0)$ 是椭圆曲线 E 上的一点，定义为 $y^2+y=x^3-x$。要在 Sage 中创建该点，输入 `E([0,0])`。Sage 可以在此椭圆曲线上添加点（椭圆曲线支持一个加法群结构，其中无穷远点为零元素，曲线上三个共线点之和为零）：

```
sage: E = EllipticCurve([0,0,1,-1,0])
sage: E
Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field
sage: P = E([0,0])
sage: P + P
(1 : 0 : 1)
sage: 10*P
(161/16 : -2065/64 : 1)
sage: 20*P
(683916417/264517696 : -18784454671297/4302115807744 : 1)
sage: E.conductor()
37
```

复数域上的椭圆曲线由 j-不变量参数化。Sage 计算 j-不变量如下：

```
sage: E = EllipticCurve([0,0,0,-4,2]); E
Elliptic Curve defined by y^2 = x^3 - 4*x + 2 over Rational Field
sage: E.conductor()
2368
sage: E.j_invariant()
110592/37
```

如果我们创建一个具有与 E 相同 j-不变量的曲线，它不一定与 E 同构。在以下示例中，这些曲线不相同，因为它们的导数不同。

```
sage: F = EllipticCurve_from_j(110592/37)
sage: F.conductor()
37
```

然而，通过对 F 进行 2 次扭转可以得到一个与其同构的曲线。

```
sage: G = F.quadratic_twist(2); G
Elliptic Curve defined by y^2 = x^3 - 4*x + 2 over Rational Field
sage: G.conductor()
2368
sage: G.j_invariant()
110592/37
```

我们可以计算椭圆曲线的 L-级数或模形式 $\sum_{n=0}^{\infty} a_n q^n$ 的系数 a_n。此计算使用 PARI C 库：

```
sage: E = EllipticCurve([0,0,1,-1,0])
sage: E.anlist(30)
[0, 1, -2, -3, 2, -2, 6, -1, 0, 6, 4, -5, -6, -2, 2, 6, -4, 0, -12, 0, -4,
 3, 10, 2, 0, -1, 4, -9, -2, 6, -12]
sage: v = E.anlist(10000)
```

对于 $n \leq 10^5$，计算所有 a_n 仅需几秒：

```
sage: %time v = E.anlist(100000)
CPU times: user 0.98 s, sys: 0.06 s, total: 1.04 s
Wall time: 1.06
```

椭圆曲线可以使用它们的 Cremona 标签构造。这会预加载椭圆曲线的秩、Tamagawa 数、调节器等信息。

```
sage: E = EllipticCurve("37b2")
sage: E
Elliptic Curve defined by y^2 + y = x^3 + x^2 - 1873*x - 31833 over Rational
Field
sage: E = EllipticCurve("389a")
sage: E
Elliptic Curve defined by y^2 + y = x^3 + x^2 - 2*x  over Rational Field
sage: E.rank()
2
sage: E = EllipticCurve("5077a")
sage: E.rank()
3
```

我们也可以直接访问 Cremona 数据库。

```
sage: db = sage.databases.cremona.CremonaDatabase()
sage: db.curves(37)
{'a1': [[0, 0, 1, -1, 0], 1, 1], 'b1': [[0, 1, 1, -23, -50], 0, 3]}
sage: db.allcurves(37)
{'a1': [[0, 0, 1, -1, 0], 1, 1],
 'b1': [[0, 1, 1, -23, -50], 0, 3],
 'b2': [[0, 1, 1, -1873, -31833], 0, 1],
 'b3': [[0, 1, 1, -3, 1], 0, 3]}
```

从数据库返回的对象不是 `EllipticCurve` 类型。它们是数据库中的元素，只有几个字段而已。Cremona 数据库有一个小型版本，默认随 Sage 一起分发，包含有关导子 (conductor) ≤ 10000 的椭圆曲线的有限信息。还有一个大型可选版本，包含有关所有导子不超过 120000 的曲线的大量数据（截至 2005 年 10 月）。Sage 还有一个巨大的（2GB）可选数据库包，包含 Stein-Watkins 数据库中数亿条椭圆曲线数据。

2.13.3 狄利克雷特征

Dirichlet 特征是同态 $(\mathbf{Z}/N\mathbf{Z})^* \to R^*$ 的扩展，对于某个环 R，可以通过将满足 $\gcd(N, x) > 1$ 的整数 x 映射到 0 从而得到一个 $\mathbf{Z} \to R$ 的映射。

```
sage: G = DirichletGroup(12)
sage: G.list()
[Dirichlet character modulo 12 of conductor 1 mapping 7 |--> 1, 5 |--> 1,
Dirichlet character modulo 12 of conductor 4 mapping 7 |--> -1, 5 |--> 1,
Dirichlet character modulo 12 of conductor 3 mapping 7 |--> 1, 5 |--> -1,
Dirichlet character modulo 12 of conductor 12 mapping 7 |--> -1, 5 |--> -1]
sage: G.gens()
```

(续下页)

(接上页)

```
(Dirichlet character modulo 12 of conductor 4 mapping 7 |--> -1, 5 |--> 1,
Dirichlet character modulo 12 of conductor 3 mapping 7 |--> 1, 5 |--> -1)
sage: len(G)
4
```

创建该群之后，我们继续创建一个元素并进行计算。

```
sage: G = DirichletGroup(21)
sage: chi = G.1; chi
Dirichlet character modulo 21 of conductor 7 mapping 8 |--> 1, 10 |--> zeta6
sage: chi.values()
[0, 1, zeta6 - 1, 0, -zeta6, -zeta6 + 1, 0, 0, 1, 0, zeta6, -zeta6, 0, -1,
 0, 0, zeta6 - 1, zeta6, 0, -zeta6 + 1, -1]
sage: chi.conductor()
7
sage: chi.modulus()
21
sage: chi.order()
6
sage: chi(19)
-zeta6 + 1
sage: chi(40)
-zeta6 + 1
```

还可以计算伽罗瓦群 $\mathrm{Gal}(\mathbf{Q}(\zeta_N)/\mathbf{Q})$ 对这些特征的作用，以及对应于模数分解的直积分解。

```
sage: chi.galois_orbit()
[Dirichlet character modulo 21 of conductor 7 mapping 8 |--> 1, 10 |--> -zeta6 + 1,
 Dirichlet character modulo 21 of conductor 7 mapping 8 |--> 1, 10 |--> zeta6]

sage: go = G.galois_orbits()
sage: [len(orbit) for orbit in go]
[1, 2, 2, 1, 1, 2, 2, 1]

sage: G.decomposition()
[Group of Dirichlet characters modulo 3 with values in Cyclotomic Field of order 6 and degree 2,
 Group of Dirichlet characters modulo 7 with values in Cyclotomic Field of order 6 and degree 2]
```

接下来，我们构造模 20 的狄利克雷特征群，但其值在 $\mathbf{Q}(i)$ 中：

```
sage: K.<i> = NumberField(x^2+1)
sage: G = DirichletGroup(20,K)
sage: G
Group of Dirichlet characters modulo 20 with values in Number Field in i with defining␣
↪polynomial x^2 + 1
```

接下来我们计算 `G` 的几个不变量：

```
sage: G.gens()
(Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1,
Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> i)

sage: G.unit_gens()
(11, 17)
sage: G.zeta()
i
```

(续下页)

(接上页)

```
sage: G.zeta_order()
4
```

下面这个例子中，我们创建了一个值在数域中的狄利克雷特征。通过 DirichletGroup 的第三个参数明确指定了选择的单位根。

```
sage: x = polygen(QQ, 'x')
sage: K = NumberField(x^4 + 1, 'a'); a = K.0
sage: b = K.gen(); a == b
True
sage: K
Number Field in a with defining polynomial x^4 + 1
sage: G = DirichletGroup(5, K, a); G
Group of Dirichlet characters modulo 5 with values in the group of order 8 generated by a in␣
↪Number Field in a with defining polynomial x^4 + 1
sage: chi = G.0; chi
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> a^2
sage: [(chi^i)(2) for i in range(4)]
[1, a^2, -1, -a^2]
```

这里 NumberField(x^4 + 1, 'a') 告诉 Sage 在打印 K 时使用符号"a"（一个定义多项式 x^4+1 的数域）。此时名称"a" 尚未声明。一旦执行 a = K.0 （或等价的 a = K.gen()），符号"a" 就代表生成多项式 x^4+1 的一个根。

2.13.4 模形式

Sage 可以进行一些与模形式相关的计算，包括计算维度、模符号空间、Hecke 算子和分解。

有几个函数可以用来计算模形式空间的维度。例如，

```
sage: from sage.modular.dims import dimension_cusp_forms
sage: dimension_cusp_forms(Gamma0(11),2)
1
sage: dimension_cusp_forms(Gamma0(1),12)
1
sage: dimension_cusp_forms(Gamma1(389),2)
6112
```

接下来我们展示如何在权重 12 和级别 1 的模符号空间上计算 Hecke 算子。

```
sage: M = ModularSymbols(1,12)
sage: M.basis()
([X^8*Y^2,(0,0)], [X^9*Y,(0,0)], [X^10,(0,0)])
sage: t2 = M.T(2)
sage: t2
Hecke operator T_2 on Modular Symbols space of dimension 3 for Gamma_0(1)
of weight 12 with sign 0 over Rational Field
sage: t2.matrix()
[ -24    0    0]
[   0  -24    0]
[4860    0 2049]
sage: f = t2.charpoly('x'); f
x^3 - 2001*x^2 - 97776*x - 1180224
sage: factor(f)
(x - 2049) * (x + 24)^2
sage: M.T(11).charpoly('x').factor()
(x - 285311670612) * (x - 534612)^2
```

我们还可以创建 $\Gamma_0(N)$ 和 $\Gamma_1(N)$ 的模符号空间。

```
sage: ModularSymbols(11,2)
Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign
 0 over Rational Field
sage: ModularSymbols(Gamma1(11),2)
Modular Symbols space of dimension 11 for Gamma_1(11) of weight 2 with
sign 0 over Rational Field
```

让我们计算一些特征多项式和 q 展开式。

```
sage: M = ModularSymbols(Gamma1(11),2)
sage: M.T(2).charpoly('x')
x^11 - 8*x^10 + 20*x^9 + 10*x^8 - 145*x^7 + 229*x^6 + 58*x^5 - 360*x^4
     + 70*x^3 - 515*x^2 + 1804*x - 1452
sage: M.T(2).charpoly('x').factor()
(x - 3) * (x + 2)^2 * (x^4 - 7*x^3 + 19*x^2 - 23*x + 11)
        * (x^4 - 2*x^3 + 4*x^2 + 2*x + 11)
sage: S = M.cuspidal_submodule()
sage: S.T(2).matrix()
[-2  0]
[ 0 -2]
sage: S.q_expansion_basis(10)
[q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 - 2*q^9 + O(q^10)]
```

我们甚至可以计算带有特征的模符号空间。

```
sage: G = DirichletGroup(13)
sage: e = G.0^2
sage: M = ModularSymbols(e,2); M
Modular Symbols space of dimension 4 and level 13, weight 2, character
[zeta6], sign 0, over Cyclotomic Field of order 6 and degree 2
sage: M.T(2).charpoly('x').factor()
(x - zeta6 - 2) * (x - 2*zeta6 - 1) * (x + zeta6 + 1)^2
sage: S = M.cuspidal_submodule(); S
Modular Symbols subspace of dimension 2 of Modular Symbols space of
dimension 4 and level 13, weight 2, character [zeta6], sign 0, over
Cyclotomic Field of order 6 and degree 2
sage: S.T(2).charpoly('x').factor()
(x + zeta6 + 1)^2
sage: S.q_expansion_basis(10)
[q + (-zeta6 - 1)*q^2 + (2*zeta6 - 2)*q^3 + zeta6*q^4 + (-2*zeta6 + 1)*q^5 + (-2*zeta6 + 4)*q^6␣
↪+ (2*zeta6 - 1)*q^8 - zeta6*q^9 + O(q^10)]
```

以下是 Sage 如何计算 Hecke 算子在模形式空间上的作用的另一个例子。

```
sage: T = ModularForms(Gamma0(11),2)
sage: T
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of
weight 2 over Rational Field
sage: T.degree()
2
sage: T.level()
11
sage: T.group()
Congruence Subgroup Gamma0(11)
sage: T.dimension()
2
```

(续下页)

(接上页)

```
sage: T.cuspidal_subspace()
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 2 for
Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
sage: T.eisenstein_subspace()
Eisenstein subspace of dimension 1 of Modular Forms space of dimension 2
for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
sage: M = ModularSymbols(11); M
Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign
0 over Rational Field
sage: M.weight()
2
sage: M.basis()
((1,0), (1,8), (1,9))
sage: M.sign()
0
```

设 T_p 表示通常的 Hecke 算子 (p 是质数)。Hecke 算子 T_2, T_3, T_5 如何在模符号空间上作用?

```
sage: M.T(2).matrix()
[ 3  0 -1]
[ 0 -2  0]
[ 0  0 -2]
sage: M.T(3).matrix()
[ 4  0 -1]
[ 0 -1  0]
[ 0  0 -1]
sage: M.T(5).matrix()
[ 6  0 -1]
[ 0  1  0]
[ 0  0  1]
```

CHAPTER 3

交互式 Shell

在本教程的大部分内容中，我们假定你使用 `sage` 命令启动 Sage 解释器。这将启动一个定制版的 IPython Shell，并导入许多函数和类，使它们可以直接从命令提示符使用。可以通过编辑 `$SAGE_ROOT/ipythonrc` 文件进行进一步的自定义。启动 Sage 后，会输出以下类似内容：

```
┌────────────────────────────────────────────────────────────────────┐
| SageMath version 9.7, Release Date: 2022-01-10                     |
| Using Python 3.10.4. Type "help()" for help.                       |
└────────────────────────────────────────────────────────────────────┘


sage:
```

要退出 Sage 只需按 Ctrl-D 或输入 `quit` 或 `exit`。

```
sage: quit
Exiting Sage (CPU time 0m0.00s, Wall time 0m0.89s)
```

Wall time 指的是墙上的挂钟走过的时间。因为 CPU 时间不会跟踪子进程（如 GAP 或 Singular）消耗的时间。

（请避免在终端中使用 `kill -9` 杀死 Sage 进程，因为 Sage 可能无法终止子进程，例如 Maple 进程，或清理 `$HOME/.sage/tmp` 中的临时文件。）

3.1 Sage 会话

会话是从 Sage 启动到退出期间的输入输出序列。Sage 通过 IPython 记录所有 Sage 输入。实际上，如果你使用的是交互式 Shell（而不是 notebook 界面），你可以随时输入 `%history` （或 `%hist`）来列出迄今为止输入的所有命令行。在 Sage 提示符下输入 `?` 可以了解有关 IPython 的更多信息，例如，“IPython 提供带编号的提示符... 并缓存输入和输出。所有输入都会保存，并且可以作为变量检索（除了常用的箭头键召回外）。以下全局变量始终存在（所以不要覆盖它们！）”：

```
_:  上一次输入 (交互式 SHell 和 notebook 均适用)
__: 上两次输入 (仅交互式 Shell 适用)
_oh : 所有输入的列表 (仅交互式 Shell 适用)
```

例如：

```
sage: factor(100)
 _1 = 2^2 * 5^2
sage: kronecker_symbol(3,5)
 _2 = -1
sage: %hist   # This only works from the interactive shell, not the notebook.
1: factor(100)
2: kronecker_symbol(3,5)
3: %hist
sage: _oh
 _4 = {1: 2^2 * 5^2, 2: -1}
sage: _i1
 _5 = 'factor(ZZ(100))\n'
sage: eval(_i1)
 _6 = 2^2 * 5^2
sage: %hist
1: factor(100)
2: kronecker_symbol(3,5)
3: %hist
4: _oh
5: _i1
6: eval(_i1)
7: %hist
```

我们在本教程和其他 Sage 文档中均省略了输出编号。

你还可以在会话中将输入列表储存在宏中。

```
sage: E = EllipticCurve([1,2,3,4,5])
sage: M = ModularSymbols(37)
sage: %hist
1: E = EllipticCurve([1,2,3,4,5])
2: M = ModularSymbols(37)
3: %hist
sage: %macro em 1-2
Macro `em` created. To execute, type its name (without quotes).
```

```
sage: E
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5 over
Rational Field
sage: E = 5
sage: M = None
sage: em
Executing Macro...
sage: E
Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5 over
Rational Field
```

在使用交互式 Shell 时，任何 UNIX Shell 命令都可以通过在 Sage 前面加上感叹号 ! 来执行。例如：

```
sage: !ls
auto  example.sage glossary.tex  t  tmp  tut.log  tut.tex
```

返回当前目录的列表。

`PATH` 变量将 Sage 的 bin 目录放在最前端，因此如果运行 `gp`, `gap`, `singular`, `maxima` 等等，你会得到随 Sage 附带的版本。

```
sage: !gp
Reading GPRC: /etc/gprc ...Done.

                          GP/PARI CALCULATOR Version 2.2.11 (alpha)
                  i686 running linux (ix86/GMP-4.1.4 kernel) 32-bit version
...
sage: !singular
                     SINGULAR                                /  Development
 A Computer Algebra System for Polynomial Computations   /   version 3-0-1
                                                       0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann        \   October 2005
FB Mathematik der Universitaet, D-67653 Kaiserslautern    \
```

3.2 记录输入和输出

记录 Sage 会话不同于保存会话（参见保存和加载完整会话）。要记录输入（和可选输出），请使用 `logstart` 命令。输入 `logstart?` 了解更多详情。你可以使用这个命令记录你输入的所有内容、所有输出，甚至可以在未来的会话中重现输入（通过重新加载日志文件）。

```
was@form:~$ sage
┌────────────────────────────────────────────────────────────────────┐
│ SageMath version 9.7, Release Date: 2022-01-10                     │
│ Using Python 3.10.4. Type "help()" for help.                       │
└────────────────────────────────────────────────────────────────────┘

sage: logstart setup
Activating auto-logging. Current session state plus future input saved.
Filename       : setup
Mode           : backup
Output logging : False
Timestamping   : False
State          : active
sage: E = EllipticCurve([1,2,3,4,5]).minimal_model()
sage: F = QQ^3
sage: x,y = QQ['x,y'].gens()
sage: G = E.gens()
sage:
Exiting Sage (CPU time 0m0.61s, Wall time 0m50.39s).
was@form:~$ sage
┌────────────────────────────────────────────────────────────────────┐
│ SageMath version 9.7, Release Date: 2022-01-10                     │
│ Using Python 3.10.4. Type "help()" for help.                       │
└────────────────────────────────────────────────────────────────────┘

sage: load("setup")
Loading log file <setup> one line at a time...
Finished replaying log file <setup>
sage: E
Elliptic Curve defined by y^2 + x*y  = x^3 - x^2 + 4*x + 3 over Rational
Field
sage: x*y
x*y
sage: G
[(2 : 3 : 1)]
```

如果你在 Linux KDE 终端 `konsole` 中使用 Sage，那么可以按照以下步骤保存会话：在 `konsole` 中启动 Sage

后，选择“设置”，然后“历史记录...”，然后“设置为无限制”。当你准备保存会话时，选择“编辑”，然后“保存历史记录为...”，并输入一个名称将会话的文本保存到你的计算机。保存这个文件后，你可以将其加载到编辑器（例如 xemacs）并打印出来。

3.3 粘贴忽略提示符

假设你正在阅读 Sage 或 Python 计算的会话，并希望将它们复制到 Sage 中。但是有 `>>>` 或 `sage:` 提示符很烦人。实际上，你可以将包含提示符的示例复制并粘贴到 Sage 中。换句话说，默认情况下，Sage 解析器在传递给 Python 之前会删除任何前导 `>>>` 或 `sage:` 提示符。例如：

```
sage: 2^10
1024
sage: sage: sage: 2^10
1024
sage: >>> 2^10
1024
```

3.4 命令计时

如果你在输入的开头放置 `%time` 命令，那么命令执行的时间将显示在输出后。例如，我们可以比较几种幂运算的运行时间。这些计时在你电脑上可能会有很大不同，甚至在不同版本的 Sage 之间也会有所不同。首先是原生 Python：

```
sage: %time a = int(1938)^int(99484)
CPU times: user 0.66 s, sys: 0.00 s, total: 0.66 s
Wall time: 0.66
```

这意味着总共耗时 0.66 秒，”Wall time” 即墙上挂钟的时间为 0.66 秒。如果你的计算机负载较重，wall time 可能比 CPU 时间长很多。

还可以使用 `timeit` 函数来尝试在大量迭代命令下获取时间。这提供了稍微不同的信息，并且需要输入命令字符串来计时。

```
sage: timeit("int(1938)^int(99484)")
5 loops, best of 3: 44.8 ms per loop
```

接下来我们使用原生 Sage Integer 类型，它是用 Cython 调用 GMP 库实现的：

```
sage: %time a = 1938^99484
CPU times: user 0.04 s, sys: 0.00 s, total: 0.04 s
Wall time: 0.04
```

使用 PARI 的 C 语言接口：

```
sage: %time a = pari(1938)^pari(99484)
CPU times: user 0.05 s, sys: 0.00 s, total: 0.05 s
Wall time: 0.05
```

GMP 表现稍好（预料之中，因为为 Sage 构建的 PARI 版本使用 GMP 进行整数运算）。

还可以使用 `cputime` 命令计时一组命令块，如下所示：

```
sage: t = cputime()
sage: a = int(1938)^int(99484)
sage: b = 1938^99484
```

(续下页)

(接上页)

```
sage: c = pari(1938)^pari(99484)
sage: cputime(t)                          # somewhat random output
0.64
```

```
sage: cputime?
...
    Return the time in CPU second since Sage started, or with optional
    argument t, return the time since time t.
    INPUT:
        t -- (optional) float, time in CPU seconds
    OUTPUT:
        float -- time in CPU seconds
```

walltime 命令的行为与 cputime 命令类似，只是它计算的是挂钟时间。

我们也可以用 Sage 包含的计算机代数系统计算上面的幂。以下每种情况下，我们执行一个简单命令以启动该程序的服务器。最相关的时间是挂钟时间。然而，如果挂钟时间和 CPU 时间之间存在显著差异，则可能表明存在值得优化的性能问题。

```
sage: time 1938^99484;
CPU times: user 0.01 s, sys: 0.00 s, total: 0.01 s
Wall time: 0.01
sage: gp(0)
0
sage: time g = gp('1938^99484')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.04
sage: maxima(0)
0
sage: time g = maxima('1938^99484')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.30
sage: kash(0)
0
sage: time g = kash('1938^99484')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.04
sage: mathematica(0)
        0
sage: time g = mathematica('1938^99484')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.03
sage: maple(0)
0
sage: time g = maple('1938^99484')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 0.11
sage: libgap(0)
0
sage: time g = libgap.eval('1938^99484;')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 1.02
```

注意，在这项测试中 GAP 和 Maxima 最慢（运行在 sage.math.washington.edu 机器上）。由于 pexpect 接口的开销，将它们与最快的 Sage 相比可能不太公平。

3.5 其他 IPython 技巧

如上文所述，Sage 使用 IPython 作为前端，因此你可以使用任何 IPython 的命令和功能。你可以阅读 完整的 IPython 文档 。下面是一些有趣的技巧 -- 在 IPython 中，这些被称为"Magic 命令"：

- 如果你想输入一些复杂代码，可以使用 `%edit` （或 `%ed` 或 `ed`）打开一个编辑器。在启动 Sage 之前，请确保 `EDITOR` 环境变量设置为你喜欢的编辑器（通过在适当位置如 `.profile` 文件中放置 `export EDITOR=/usr/bin/emacs` 或 `export EDITOR=/usr/bin/vim` 等）。在 Sage 提示符下执行 `%edit` 会打开指定的编辑器。然后在编辑器中你可以定义一个函数：

  ```
  def some_function(n):
      return n**2 + 3*n + 2
  ```

  保存并退出编辑器。在剩下的 Sage 会话期间，你可以使用 `some_function`。如果你想修改它，可以在 Sage 提示符下输入 `%edit some_function`。

- 如果你有一个计算，并且想修改其输出以便用于其他用途，可执行计算并输入 `%rep`：这会将上一个命令的输出放置到 Sage 提示符，供你编辑。：

  ```
  sage: f(x) = cos(x)
  sage: f(x).derivative(x)
  -sin(x)
  ```

  此时如果你在 Sage 提示符下输入 `%rep`, 你会得到一个新的 Sage 提示符，后面跟着 `-sin(x)`, 光标在行尾。

要了解更多信息，请输入 `%quickref` 以获得 IPython 快速参考指南。截止本文撰写时间（2011 年 4 月），Sage 使用的 IPython 版本为 0.9.1，Magic 命令文档 可以在线访问。各种较为高级的 Magic 命令系统的内容记载在 这里 。

3.6 错误与异常

出现问题时，通常会看到 Python "异常"。Python 甚至会尝试给出引发异常的原因。通常可以看到异常的名称，例如：`NameError` 或 `ValueError` （详细异常列表请参见 Python 库参考 [PyLR] ）。例如：

```
sage: 3_2
------------------------------------------------------------
   File "<console>", line 1
     ZZ(3)_2
           ^
SyntaxError: invalid ...

sage: EllipticCurve([0,infinity])
------------------------------------------------------------
Traceback (most recent call last):
...
TypeError: Unable to coerce Infinity (<class 'sage...Infinity'>) to Rational
```

有时交互式调试器对理解问题很有用。可以使用 `%pdb` 切换它（默认是关闭的）。如果打开调试器，出现异常时会出现提示符 `ipdb>`。在调试器中，可以打印任意局部变量的状态，并在执行栈中上下移动。例如：

```
sage: %pdb
Automatic pdb calling has been turned ON
sage: EllipticCurve([1,infinity])
---------------------------------------------------------------------------
<class 'exceptions.TypeError'>             Traceback (most recent call last)
```

(续下页)

(接上页)

```
...

ipdb>
```

在 ipdb> 提示符下输入 ? 以获取调试器命令列表:

```
ipdb> ?

Documented commands (type help <topic>):
========================================
EOF     break   commands   debug    h        l     pdef   quit    tbreak
a       bt      condition  disable  help     list  pdoc   r       u
alias   c       cont       down     ignore   n     pinfo  return  unalias
args    cl      continue   enable   j        next  pp     s       up
b       clear   d          exit     jump     p     q      step    w
whatis  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv
```

输入 Ctrl-D 或 quit 返回 Sage。

3.7 反向搜索与 Tab 补全

反向搜索：输入命令的开头，然后按 Ctrl-p （或直接按上箭头键）查看以前输入的以该命令开头的命令行。即使你完全退出 Sage 并稍后重新启动，这些功能仍然可以使用。也可以使用 Ctrl-r 通过历史记录进行反向搜索。所有这些功能均使用 readline 软件包，可在大多数 Linux 版本中使用。

为了演示 Tab 补全，首先创建三维向量空间 $V = \mathbf{Q}^3$ 如下:

```
sage: V = VectorSpace(QQ,3)
sage: V
Vector space of dimension 3 over Rational Field
```

也可以使用如下更简洁的表示法:

```
sage: V = QQ^3
```

然后可以很容易地使用 Tab 补全列出 V 的所有成员函数。只需输入 V.，然后按键盘上的 Tab 键:

```
sage: V.[tab key]
V._VectorSpace_generic__base_field
...
V.ambient_space
V.base_field
V.base_ring
V.basis
V.coordinates
...
V.zero_vector
```

如果输入函数的前几个字母，然后按 Tab 键，只会显示以这些字母开头的函数。

```
sage: V.i[tab key]
V.is_ambient  V.is_dense    V.is_full     V.is_sparse
```

如果想知道某函数的作用，例如 coordinates 函数，输入 V.coordinates? 来获取帮助或 V.coordinates?? 查看源码，如下一节所述。

3.8 集成帮助系统

Sage 拥有集成帮助系统。输入函数名后跟? 可以查看该函数的文档。

```
sage: V = QQ^3
sage: V.coordinates?
Type:           instancemethod
Base Class:     <class 'instancemethod'>
String Form:    <bound method FreeModule_ambient_field.coordinates of Vector
space of dimension 3 over Rational Field>
Namespace:      Interactive
File:           /home/was/s/local/lib/python2.4/site-packages/sage/modules/f
ree_module.py
Definition:     V.coordinates(self, v)
Docstring:
    Write v in terms of the basis for self.

    Returns a list c such that if B is the basis for self, then

            sum c_i B_i = v.

    If v is not in self, raises an ArithmeticError exception.

    EXAMPLES:
        sage: M = FreeModule(IntegerRing(), 2); M0,M1=M.gens()
        sage: W = M.submodule([M0 + M1, M0 - 2*M1])
        sage: W.coordinates(2*M0-M1)
        [2, -1]
```

如上所示，输出告诉你对象的类型，定义它的文件，以及有用的函数描述及示例，可以将这些示例粘贴到当前会话中。几乎所有这些示例都会定期自动测试，以确保它们正常工作并完全按照描述运行。

另一个非常符合 Sage 开源精神的功能是，如果 f 是一个 Python 函数，那么输入 f?? 会显示定义 f 的源代码。例如：

```
sage: V = QQ^3
sage: V.coordinates??
Type:           instancemethod
...
Source:
def coordinates(self, v):
        """
        Write $v$ in terms of the basis for self.
        ...
        """
        return self.coordinate_vector(v).list()
```

这告诉我们 coordinates 函数所做的就是调用 coordinate_vector 函数并将结果转换为列表。coordinate_vector 函数做什么？

```
sage: V = QQ^3
sage: V.coordinate_vector??
...
def coordinate_vector(self, v):
        ...
        return self.ambient_vector_space()(v)
```

coordinate_vector 函数将其输入强制转化环绕空间，其效果是以 V 的形式计算 v 的系数向量。空间 V 已经是环绕空间，因为它就是 $\mathbf{Q}^3$。子空间也有 coordinate_vector 函数，它是不同的。我们创建一个子空间并看到：

```
sage: V = QQ^3; W = V.span_of_basis([V.0, V.1])
sage: W.coordinate_vector??
...
def coordinate_vector(self, v):
        """
         ...
        """
        # First find the coordinates of v wrt echelon basis.
        w = self.echelon_coordinate_vector(v)
        # Next use transformation matrix from echelon basis to
        # user basis.
        T = self.echelon_to_user_matrix()
        return T.linear_combination_of_rows(w)
```

（如果你认为实现效率低下，请注册以帮助优化线性代数。）

你也可以输入 help(command_name) 或 help(class) 来获取给定类的帮助文档（类似 manpage ）。

```
sage: help(VectorSpace)
Help on function VectorSpace in module sage.modules.free_module:

VectorSpace(K, dimension_or_basis_keys=None, sparse=False, inner_product_matrix=None, *,
            with_basis='standard', dimension=None, basis_keys=None, **args)
EXAMPLES:

The base can be complicated, as long as it is a field.

::

    sage: V = VectorSpace(FractionField(PolynomialRing(ZZ,'x')),3)
    sage: V
    Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring in x
     over Integer Ring
    sage: V.basis()
    [
    (1, 0, 0),
    (0, 1, 0),
--More--
```

当你输入 q 退出帮助系统时，你的会话内容将保持不变。帮助列表不会使你的会话变得杂乱，而 function_name? 的输出有时会造成这种情况。输入 help(module_name) 特别有用。例如，向量空间在 sage.modules.free_module 中定义，输入 help(sage.modules.free_module) 即可获得有关整个模块的文档。使用帮助查看文档时，可以通过输入 / 进行搜索，也可以通过输入 ? 反向搜索。

3.9 保存和加载单个对象

假设你计算出一个矩阵或更复杂的模符号空间，并希望将其保存以供日后使用。你要怎么办呢？计算机代数系统采用多种方法来保存单个对象。

1. **保存游戏：**仅支持保存和加载完整会话（如 GAP、Magma）。
2. **统一输入输出：**使每个对象都以可读的方式打印（GP/PARI）。
3. **Eval:** 轻松在解释器中计算任意代码（如 Singular、PARI）。

由于 Sage 使用 Python，因此采用不同的方法，即每个对象都可以序列化，转化为一个可以从中恢复该对象的字符串。这与 PARI 的统一输入输出方法精神相似，只不过对象打印到屏幕的方式不会过于复杂。此外，保存和加载在大多数情况下是完全自动的，不需要额外编程；这是 Python 的设计特性。

几乎所有 Sage 对象 x 都可以以压缩形式保存到磁盘，使用 `save(x, filename)`（或在许多情况下 `x.save(filename)`）。要加载对象，使用 `load(filename)`。

```
sage: A = MatrixSpace(QQ,3)(range(9))^2
sage: A
[ 15  18  21]
[ 42  54  66]
[ 69  90 111]
sage: save(A, 'A')
```

现在你应该退出 Sage 并重新启动。然后便可以恢复 `A`：

```
sage: A = load('A')
sage: A
[ 15  18  21]
[ 42  54  66]
[ 69  90 111]
```

可以使用同样的方法处理更复杂的对象，如椭圆曲线。缓存对象的所有数据都与对象一同保存。例如：

```
sage: E = EllipticCurve('11a')
sage: v = E.anlist(100000)               # takes a while
sage: save(E, 'E')
sage: quit
```

`E` 的存储版占 153K 字节，因为它储存了前 100000 个 a_n.

```
~/tmp$ ls -l E.sobj
-rw-r--r--  1 was was 153500 2006-01-28 19:23 E.sobj
~/tmp$ sage [...]
sage: E = load('E')
sage: v = E.anlist(100000)               # instant!
```

（在 Python 中，保存和加载使用 `cPickle` 模块实现。具体来说，Sage 对象 `x` 可以通过 `cPickle.dumps(x, 2)` 保存。注意 `2`!）

Sage 无法保存和加载某些其它计算机代数系统（例如 GAP、Singular、Maxima）创建的单个对象。它们重新加载时状态显示为“无效 (invalid)”。在 GAP 中，虽然许多对象的打印方式可以重新构建，但很多对象却不行，因此特意不允许从其打印表示进行重建。

```
sage: a = libgap(2)
sage: a.save('a')
sage: load('a')
Traceback (most recent call last):
```

(续下页)

(接上页)

```
...
ValueError: The session in which this object was defined is no longer
running.
```

GP/PARI 对象可以保存和加载，因为它们的打印表示足以重构它们。

```
sage: a = gp(2)
sage: a.save('a')
sage: load('a')
2
```

保存的对象稍后可以在不同架构或操作系统的计算机上重新加载，例如，你可以在 32 位 OS X 上保存一个大矩阵，然后在 64 位 Linux 上重新加载它，计算阶梯形式，然后再保存回去。此外，在许多情况下，即使在不同版本的 Sage 中也能加载对象，只要该对象的代码没有太大差异。对象的所有属性，以及定义对象的类（但不包括源代码）都会被保存。如果该类在新版本的 Sage 中不再存在，那么该对象就无法在新版本中重新加载。但你可以在老版本中加载它，获取其对象字典（使用 `x.__dict__`），保存该字典，并将其加载到新版本中。

3.9.1 保存为文本

你还可以将对象的 ASCII 文本表示保存到纯文本文件中，只需以写入模式打开文件并写入对象的字符串表示即可（你也可以通过这种方式写入许多对象）。写完对象后，关闭文件。

```
sage: R.<x,y> = PolynomialRing(QQ,2)
sage: f = (x+y)^7
sage: o = open('file.txt','w')
sage: o.write(str(f))
sage: o.close()
```

3.10 保存和加载完整会话

Sage 对于保存和加载完整会话有非常灵活的支持。

`save_session(sessionname)` 命令将所有在当前会话中定义的变量保存为给定 `sessionname` 的字典。（在少数情况下，如果某个变量不支持保存，则不会保存到字典。）生成的文件为 `.sobj` 文件，可以像保存的其它对象一样加载。加载会话保存的对象时，会得到一个字典，字典的键为变量名，值为对象。

可以使用 `load_session(sessionname)` 命令将 `sessionname` 中定义的变量加载到当前会话。注意，这不会清除当前会话中已经定义的变量；而是合并两个会话。

首先启动 Sage 并定义一些变量。

```
sage: E = EllipticCurve('11a')
sage: M = ModularSymbols(37)
sage: a = 389
sage: t = M.T(2003).matrix(); t.charpoly().factor()
 _4 = (x - 2004) * (x - 12)^2 * (x + 54)^2
```

接下来保存会话，将上面定义的每个变量保存至文件。然后查看文件，大小约为 3K。

```
sage: save_session('misc')
Saving a
Saving M
Saving t
Saving E
```

(续下页)

(接上页)

```
sage: quit
was@form:~/tmp$ ls -l misc.sobj
-rw-r--r--  1 was was 2979 2006-01-28 19:47 misc.sobj
```

最后重新启动 Sage，定义一个额外的变量，并加载保存的会话。

```
sage: b = 19
sage: load_session('misc')
Loading a
Loading M
Loading E
Loading t
```

每个保存的变量再次可用。此外，变量 b 没有被覆盖。

```
sage: M
Full Modular Symbols space for Gamma_0(37) of weight 2 with sign 0
and dimension 5 over Rational Field
sage: E
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational
Field
sage: b
19
sage: a
389
```

CHAPTER 4

接口

Sage 的一个核心功能是它支持在通用接口和简洁的编程语言下，使用来自多个不同计算机代数系统的对象进行计算。

接口的 console 和 interact 方法的作用非常不同。例如，以 GAP 为例：

1. `gap.console()`: 这会打开 GAP 控制台 - 将控制权转移给 GAP。在这里，Sage 只是充当一个方便的程序启动器，类似于 Linux 的 bash shell。
2. `gap.interact()`: 这是与正在运行的 GAP 实例交互的便捷方式，该实例可能“装满了”Sage 对象。你可以将 Sage 对象导入到这个 GAP 会话中（甚至可以从交互界面中导入）等等。

4.1 GP/PARI

PARI 是一款小巧紧凑、非常成熟、高度优化的 C 程序，其主要关注点是数论。Sage 中有两个截然不同的接口可供使用：

- `gp` -- **PARI** 解释器
- `pari` -- **PARI** C 库

例如，以下是同一任务的两种实现方法。它们看起来一样，但输出结果实际上是不同的，并且幕后发生的事情也截然不同。

```
sage: gp('znprimroot(10007)')
Mod(5, 10007)
sage: pari('znprimroot(10007)')
Mod(5, 10007)
```

在第一种情况下，会启动一个单独的 GP 解释器副本作为服务器，并将字符串 `'znprimroot(10007)'` 发送给它，经 GP 计算后，结果被赋予 GP 中的一个变量（该变量占用子 GP 进程内存中的空间，不会被释放）。然后显示该变量的值。在第二种情况下，没有启动单独的程序，并且字符串 `'znprimroot(10007)'` 被某个 PARI C 库函数计算。结果存储在 Python 的堆内存中，当该变量不再被引用时，该内存将被释放。对象具有不同的类型：

```
sage: type(gp('znprimroot(10007)'))
<class 'sage.interfaces.gp.GpElement'>
sage: type(pari('znprimroot(10007)'))
<class 'cypari2.gen.Gen'>
```

那么应该使用哪一种呢？这取决于你的需求。GP 接口可以完成在通常的 GP/PARI 命令行程序中你可以做的任何任务，尤其是你可以加载复杂的 PARI 程序并运行它们。而使用 PARI 接口（通过 C 库）限制要多得多。首先，所有的成员函数尚未完全实现。其次，许多代码，例如涉及数值积分的代码，通过 PARI 接口无法工作。话虽如此，PARI 接口显著比 GP 接口更快、更稳健。

（如果 GP 接口在计算给定输入时内存耗尽，它会静默地自动将堆栈大小加倍并重试该输入。因此，如果你没有正确预估所需的内存，你的计算也不会崩溃。这是通常的 GP 解释器似乎不提供的一个不错的技巧。对于 PARI C 库接口，它会立即将每个创建的对象从 PARI 堆栈中复制出来，因此堆栈不会增长。然而，每个对象的大小不得超过 100MB，否则在创建对象时堆栈将溢出。这个额外的复制会导致一定的性能损耗。）

总的来说，Sage 使用 PARI C 库提供了与 GP/PARI 解释器类似的功能，不同之处在于具有不同的复杂内存管理和 Python 编程语言。

首先，我们从 Python 列表创建一个 PARI 列表。

```
sage: v = pari([1,2,3,4,5])
sage: v
[1, 2, 3, 4, 5]
sage: type(v)
<class 'cypari2.gen.Gen'>
```

每个 PARI 对象的类型都是 `Gen`。底层对象的 PARI 类型可以使用 `type` 成员函数来获取。

```
sage: v.type()
't_VEC'
```

在 PARI 中，要创建一个椭圆曲线，我们输入 `ellinit([1,2,3,4,5])`。与 Sage 类似，除了 `ellinit` 是一个可以在任何 PARI 对象上调用的方法，例如我们的 `t_VEC` v。

```
sage: e = v.ellinit()
sage: e.type()
't_VEC'
sage: pari(e)[:13]
[1, 2, 3, 4, 5, 9, 11, 29, 35, -183, -3429, -10351, 6128487/10351]
```

现在我们有了一个椭圆曲线对象，我们可以计算关于它的一些信息。

```
sage: e.elltors()
[1, [], []]
sage: e.ellglobalred()
[10351, [1, -1, 0, -1], 1, [11, 1; 941, 1], [[1, 5, 0, 1], [1, 5, 0, 1]]]
sage: f = e.ellchangecurve([1,-1,0,-1])
sage: f[:5]
[1, -1, 0, 4, 3]
```

4.2 GAP

Sage 附带用于计算离散数学，尤其是群论的 GAP。

以下是 GAP 的 `IdGroup` 函数的例子。

```
sage: G = gap('Group((1,2,3)(4,5), (3,4))')
sage: G
Group( [ (1,2,3)(4,5), (3,4) ] )
sage: G.Center()
Group( () )
sage: G.IdGroup()
[ 120, 34 ]
sage: G.Order()
120
```

我们可以在 Sage 中执行相同的计算，而无需显式调用 GAP 接口，如下所示：

```
sage: G = PermutationGroup([[(1,2,3),(4,5)],[(3,4)]])
sage: G.center()
Subgroup generated by [()] of (Permutation Group with generators [(3,4), (1,2,3)(4,5)])
sage: G.group_id()
[120, 34]
sage: n = G.order(); n
120
```

对于某些 GAP 功能，你需要安装可选的 Sage 软件包。可以通过如下命令完成:

```
sage -i gap_packages
```

4.3 Singular

Singular 提供了一个庞大且成熟的库，用于处理 Gröbner 基、多元多项式最大公因数、平面曲线上的 Riemann-Roch 空间基，以及因式分解等。我们将使用 Sage 接口来展示多元多项式的因式分解（请勿输入 ....:）:

```
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: R1
polynomial ring, over a field, global ordering
// coefficients: QQ...
// number of vars : 2
//        block   1 : ordering dp
//                  : names    x y
//        block   2 : ordering C
sage: f = singular('9*y^8 - 9*x^2*y^7 - 18*x^3*y^6 - 18*x^5*y^6 +'
....:     '9*x^6*y^4 + 18*x^7*y^5 + 36*x^8*y^4 + 9*x^10*y^4 - 18*x^11*y^2 -'
....:     '9*x^12*y^3 - 18*x^13*y^2 + 9*x^16')
```

现在我们已经定义了 f，我们输出它并进行因式分解。

```
sage: f
9*x^16-18*x^13*y^2-9*x^12*y^3+9*x^10*y^4-18*x^11*y^2+36*x^8*y^4+18*x^7*y^5-18*x^5*y^6+9*x^6*y^4-
↪18*x^3*y^6-9*x^2*y^7+9*y^8
sage: f.parent()
Singular
sage: F = f.factorize(); F
[1]:
   _[1]=9
   _[2]=x^6-2*x^3*y^2-x^2*y^3+y^4
   _[3]=-x^5+y^2
[2]:
```

(续下页)

(接上页)

```
   1,1,2
sage: F[1][2]
x^6-2*x^3*y^2-x^2*y^3+y^4
```

与*GAP* 中的 GAP 示例一样，我们可以计算上述因式分解而无需显式调用 Singular 接口（然而，Sage 实际上在幕后使用 Singular 接口来进行实际计算）。请勿输入 ....:

```
sage: x, y = QQ['x, y'].gens()
sage: f = (9*y^8 - 9*x^2*y^7 - 18*x^3*y^6 - 18*x^5*y^6 + 9*x^6*y^4
....:      + 18*x^7*y^5 + 36*x^8*y^4 + 9*x^10*y^4 - 18*x^11*y^2 - 9*x^12*y^3
....:      - 18*x^13*y^2 + 9*x^16)
sage: factor(f)
(9) * (-x^5 + y^2)^2 * (x^6 - 2*x^3*y^2 - x^2*y^3 + y^4)
```

4.4 Maxima

Maxima 包括在 Sage 中，采用 Lisp 实现。gnuplot 包（Maxima 默认用于绘图）作为 Sage 的可选包分发。除其他功能外，Maxima 还可以进行符号操作。Maxima 可以符号化积分和微分函数，求解一阶常微分方程（ODE），大部分线性二阶常微分方程，并且已经实现了对任意阶线性常微分方程的拉普拉斯变换方法。Maxima 还了解各种特殊函数，拥有通过 gnuplot 进行绘图的能力，并且具有求解和操作矩阵（如行化简、特征值和特征向量），以及多项方程的方法。

我们通过构造一个矩阵来说明 Sage/Maxima 接口。对于 $i, j = 1, \ldots, 4$，该矩阵的 i, j 项为 i/j。

```
sage: f = maxima.eval('ij_entry[i,j] := i/j')
sage: A = maxima('genmatrix(ij_entry,4,4)'); A
matrix([1,1/2,1/3,1/4],[2,1,2/3,1/2],[3,3/2,1,3/4],[4,2,4/3,1])
sage: A.determinant()
0
sage: A.echelon()
matrix([1,1/2,1/3,1/4],[0,0,0,0],[0,0,0,0],[0,0,0,0])
sage: A.eigenvalues()
[[0,4],[3,1]]
sage: A.eigenvectors().sage()
[[[0, 4], [3, 1]], [[1, 0, 0, -4], [0, 1, 0, -2], [0, 0, 1, -4/3]], [[1, 2, 3, 4]]]
```

下面是另一个例子:

```
sage: A = maxima("matrix ([1, 0, 0], [1, -1, 0], [1, 3, -2])")
sage: eigA = A.eigenvectors()
sage: V = VectorSpace(QQ,3)
sage: eigA
[[[-2,-1,1],[1,1,1]],[[[0,0,1]],[[0,1,3]],[[1,1/2,5/6]]]]
sage: v1 = V(sage_eval(repr(eigA[1][0][0]))); lambda1 = eigA[0][0][0]
sage: v2 = V(sage_eval(repr(eigA[1][1][0]))); lambda2 = eigA[0][0][1]
sage: v3 = V(sage_eval(repr(eigA[1][2][0]))); lambda3 = eigA[0][0][2]

sage: M = MatrixSpace(QQ,3,3)
sage: AA = M([[1,0,0],[1, - 1,0],[1,3, - 2]])
sage: b1 = v1.base_ring()
sage: AA*v1 == b1(lambda1)*v1
True
sage: b2 = v2.base_ring()
sage: AA*v2 == b2(lambda2)*v2
True
```

(续下页)

(接上页)

```
sage: b3 = v3.base_ring()
sage: AA*v3 == b3(lambda3)*v3
True
```

最后，我们给出一个使用 Sage 进行 openmath 绘图的例子。其中许多内容都是根据 Maxima 参考手册改编而来。

绘制多个函数的二维图像（请勿输入 ....:）：

```
sage: maxima.plot2d('[cos(7*x),cos(23*x)^4,sin(13*x)^3]','[x,0,1]',  # not tested
....:     '[plot_format,openmath]')
```

可以用鼠标移动的“动态”三维图（请勿输入 ....:）：

```
sage: maxima.plot3d ("2^(-u^2 + v^2)", "[u, -3, 3]", "[v, -2, 2]",  # not tested
....:     '[plot_format, openmath]')
sage: maxima.plot3d("atan(-x^2 + y^3/4)", "[x, -4, 4]", "[y, -4, 4]",  # not tested
....:     "[grid, 50, 50]",'[plot_format, openmath]')
```

接下来的绘图是著名的莫比乌斯带（请勿输入 ....:）：

```
sage: maxima.plot3d("[cos(x)*(3 + y*cos(x/2)), sin(x)*(3 + y*cos(x/2)), y*sin(x/2)]",  # not␣
↪tested
....:     "[x, -4, 4]", "[y, -4, 4]", '[plot_format, openmath]')
```

接下来的绘图是著名克莱因瓶（请勿输入 ....:）：

```
sage: maxima("expr_1: 5*cos(x)*(cos(x/2)*cos(y) + sin(x/2)*sin(2*y)+ 3.0) - 10.0")
5*cos(x)*(sin(x/2)*sin(2*y)+cos(x/2)*cos(y)+3.0)-10.0
sage: maxima("expr_2: -5*sin(x)*(cos(x/2)*cos(y) + sin(x/2)*sin(2*y)+ 3.0)").sage()
-5*(cos(1/2*x)*cos(y) + sin(1/2*x)*sin(2*y) + 3.0)*sin(x)
sage: maxima("expr_3: 5*(-sin(x/2)*cos(y) + cos(x/2)*sin(2*y))")
5*(cos(x/2)*sin(2*y)-sin(x/2)*cos(y))
sage: maxima.plot3d ("[expr_1, expr_2, expr_3]", "[x, -%pi, %pi]",  # not tested
....:     "[y, -%pi, %pi]", "['grid, 40, 40]", '[plot_format, openmath]')
```

CHAPTER 5

Sage, LaTeX 及其朋友们

Sage 与 TeX 的 LaTeX 方言之间存在着密切的协同关系。本节旨在介绍各种交互方式，从最基本的开始，然后介绍一些不常见的用法。

5.1 基本使用

Sage 中的每个“对象”都必须有 LaTeX 表示。你可以通过执行 `latex(foo)` 来获取这种表示，其中 `foo` 是 Sage 中的某个对象。输出是一个字符串，当在 TeX 的数学模式中使用时（例如，包围在一对单美元符号之间），该字符串应该能够准确地呈现 `foo`。以下是一些示例。

```
sage: var('z')
z
sage: latex(z^12)
z^{12}
sage: latex(sqrt(z^2 + 1/2))
\sqrt{z^{2} + \frac{1}{2}}
sage: latex('a string')
\text{\texttt{a{ }string}}
sage: latex(QQ)
\Bold{Q}
sage: latex(ZZ['x'])
\Bold{Z}[x]
sage: latex(matrix(QQ, 2, 3, [[2,4,6],[-1,-1,-1]]))
\left(\begin{array}{rrr}
2 & 4 & 6 \\
-1 & -1 & -1
\end{array}\right)
```

通过这种方式，Sage 可以有效地用于构建 LaTeX 文档的各个部分：在 Sage 中创建或计算一个对象 `foo`，对该对象执行 `latex(foo)`，然后将 LaTeX 字符串剪切并粘贴到你的文档中。

命令 `view(foo)` 会显示对象 `foo` 的渲染后的 LaTeX 表示。在后台，该命令会运行 `latex(foo)` 并将 LaTeX 字符串合并到一个简单的 LaTeX 文档中，用系统范围内的 TeX 安装处理该文档，然后调用合适的查看器来显示输出。

在 Jupyter Notebook 中，你可以自动看到输入命令输出的渲染 LaTeX 表示。你可以通过执行 `%display latex` 来启动自动渲染（并通过执行 `%display plain` 停止）。

Jupyter Notebook 使用 MathJax 在网页浏览器中清晰地渲染数学内容。MathJax 是一个开源的 JavaScript 数学显示引擎，可以在所有现代浏览器中使用。它能够渲染大部分 LaTex，但并不支持完整的 LaTeX，是 LaTex 的子集。它不支持复杂表格、分段或文档管理，因为它主要用于准确渲染 LaTeX 数学片段。

在 Jupyter Notebook 中自动 LaTeX 渲染（启用 `%display latex`）是通过 `sage.misc.html.MathJax` 类内部实现的。该类的对象将 Sage 对象通过 `latex()` 转换为 MathJax 需要的 HTML 形式，然后将其包装在 HTML 中。

```
sage: from sage.misc.html import MathJax
sage: mj = MathJax()
sage: var('z')
z
sage: mj(z^12)
<html>\[z^{12}\]</html>
sage: mj(sqrt(z^2 + 1/2))
<html>\[\sqrt{z^{2} + \frac{1}{2}}\]</html>
sage: mj('a string')
<html>\[\verb|a|\verb| |\verb|string|\]</html>
sage: mj(QQ)
<html>\[\newcommand{\Bold}[1]{\mathbf{#1}}\Bold{Q}\]</html>
sage: mj(ZZ['x'])
<html>\[\newcommand{\Bold}[1]{\mathbf{#1}}\Bold{Z}[x]\]</html>
sage: mj(matrix(QQ, 2, 3, [[2,4,6],[-1,-1,-1]]))
<html>\[\left(\begin{array}{rrr}
2 & 4 & 6 \\
-1 & -1 & -1
\end{array}\right)\]</html>
```

如果你需要了解 Sage 对象的 LaTeX 渲染，那么了解这一点很有用。

5.2 自定义 LaTeX 生成

有几种方法可以自定义由 `latex()` 命令生成的实际 LaTeX 代码。预定义对象 `latex` 包含多个方法，可以通过输入 `latex.`（注意这里有一个点）后按 `Tab` 键来列出这些方法。

`latex.matrix_delimiters` 方法是一个很好的例子。它可以用来更改矩阵周围的符号 -- 大括号、方括号、花括号、竖线。不强制执行任何样式，你可以随意混合搭配。注意，LaTeX 所需的反斜杠在 Python 字符串中需要额外加一个斜杠以便正确转义。

```
sage: A = matrix(ZZ, 2, 2, range(4))
sage: latex(A)
\left(\begin{array}{rr}
0 & 1 \\
2 & 3
\end{array}\right)
sage: latex.matrix_delimiters(left='[', right=']')
sage: latex(A)
\left[\begin{array}{rr}
0 & 1 \\
2 & 3
\end{array}\right]
sage: latex.matrix_delimiters(left='\\{', right='\\}')
sage: latex(A)
\left\{\begin{array}{rr}
```

(续下页)

(接上页)

```
0 & 1 \\
2 & 3
\end{array}\right\}
```

latex.vector_delimiters 方法的工作原理与之类似。

常见环和域（整数、有理数、实数等）的排版方式可以通过 latex.blackboard_bold 方法来控制。这些集合默认以粗体排版，但有时可以选择以双重划线格式书写，如某些书面作品所做的那样。这可以通过重新定义 Sage 内置的 \Bold{} 宏来实现。

```
sage: latex(QQ)
\Bold{Q}
sage: from sage.misc.html import MathJax
sage: mj = MathJax()
sage: mj(QQ)
<html>\[\newcommand{\Bold}[1]{\mathbf{#1}}\Bold{Q}\]</html>
sage: latex.blackboard_bold(True)
sage: mj(QQ)
<html>\[\newcommand{\Bold}[1]{\mathbb{#1}}\Bold{Q}\]</html>
sage: latex.blackboard_bold(False)
```

可以通过加入新的宏来利用 LaTeX 的可扩展性。可以添加单个宏，以便在 MathJax 解释 LaTeX 片段时使用。

```
sage: latex.add_macro(r"\newcommand{\sqrt}[1]{(#1)^\frac{1}{2}}")
sage: latex.extra_macros()
'\\newcommand{\\sqrt}[1]{(#1)^\\frac{1}{2}}'
sage: var('x y')
(x, y)
sage: latex(sqrt(x+y))
\sqrt{x + y}
sage: from sage.misc.html import MathJax
sage: mj = MathJax()
sage: mj(sqrt(x + y))
<html>\[\newcommand{\sqrt}[1]{(#1)^\frac{1}{2}}\sqrt{x + y}\]</html>
sage: latex.extra_macros('')
```

5.3 自定义 LaTeX 处理

系统范围内的 TeX 被调用来处理完整的 LaTeX 文档，例如，当你 view(foo) 时，其中 foo 是一个复杂的 Sage 对象，太复杂以至于 MathJax 无法处理。命令 latex_extra_preamble 用于构建完整 LaTeX 文档的导言部分，下面将展示如何完成这项工作。如往常一样，请注意 Python 字符串中需要双反斜杠。

```
sage: latex.extra_macros('')
sage: latex.extra_preamble('')
sage: from sage.misc.latex import latex_extra_preamble
sage: print(latex_extra_preamble())
\newcommand{\ZZ}{\Bold{Z}}
...
\newcommand{\Bold}[1]{\mathbf{#1}}
sage: latex.add_macro("\\newcommand{\\foo}{bar}")
sage: print(latex_extra_preamble())
\newcommand{\ZZ}{\Bold{Z}}
...
\newcommand{\Bold}[1]{\mathbf{#1}}
\newcommand{\foo}{bar}
```

同样，对于更大或更复杂的 LaTeX 表达式，可以将包（或其他任意内容）添加到 LaTeX 文件的导言部分。任意内容都可以通过 `latex.add_to_preamble` 命令加入导言部分，专用命令 `latex.add_package_to_preamble_if_available` 会首先检查某个包是否实际存在，然后尝试将其添加到导言部分。

这里我们将几何包添加到导言部分并用它来设置 TeX 将在页面上使用的区域尺寸（有效地设置边距）。如往常一样，请注意 Python 字符串中需要双反斜杠。

```
sage: from sage.misc.latex import latex_extra_preamble
sage: latex.extra_macros('')
sage: latex.extra_preamble('')
sage: latex.add_to_preamble('\\usepackage{geometry}')
sage: latex.add_to_preamble('\\geometry{letterpaper,total={8in,10in}}')
sage: latex.extra_preamble()
'\\usepackage{geometry}\\geometry{letterpaper,total={8in,10in}}'
sage: print(latex_extra_preamble())
\usepackage{geometry}\geometry{letterpaper,total={8in,10in}}
\newcommand{\ZZ}{\Bold{Z}}
...
\newcommand{\Bold}[1]{\mathbf{#1}}
```

可以通过检查其存在性来添加特定包，以下示例展示了这种情况。作为示例，我们将尝试向导言部分添加一个可能不存在的包。

```
sage: latex.extra_preamble('')
sage: latex.extra_preamble()
''
sage: latex.add_to_preamble('\\usepackage{foo-bar-unchecked}')
sage: latex.extra_preamble()
'\\usepackage{foo-bar-unchecked}'
sage: latex.add_package_to_preamble_if_available('foo-bar-checked')
sage: latex.extra_preamble()
'\\usepackage{foo-bar-unchecked}'
```

使用哪种 TeX 方言，以及输出和相关查看器的性质，也可以定制。

备注

Sage 几乎包括了构建和使用 Sage 所需的一切，但一个重要的例外是 TeX 本身。因此，在以下情况下，你需要安装完整的 TeX 系统以及一些相关的转换工具。许多版本的 Linux 都有基于 TeXLive 的软件包，macOS 有 MacTeX，Windows 有 MiKTeX。

可以使用 `latex.engine()` 命令控制是否使用系统范围内的 `latex`, `pdflatex` 或 `xelatex` 可执行文件。当调用 `view()` 并且引擎设置为 `latex` 时，会生成一个 dvi 文件，Sage 会使用 dvi 查看器（如 xdvi）来显示结果。相比之下，当引擎设置为 `pdflatex` 时，调用 `view()` 会生成 PDF 文件，并且 Sage 会调用系统的 PDF 文件查看工具（如 acrobat, okular, evince 等）。

对于使用这些工具的练习，有一些预先打包好的示例。要使用这些示例，需要导入 `sage.misc.latex.latex_examples` 对象，这是 `sage.misc.latex.LatexExamples` 类的一个实例，如下所示。目前该类有交换图、组合图、扭结理论和 pstricks 的示例，分别使用以下包：xy，tkz-graph，xypic，pstricks。导入后，对 `latex_examples` 使用 tab 补全查看内置示例。调用每个示例会返回一些关于如何正确呈现该示例的说明。要实际查看示例，需要使用 `view(foo)`（导言部分、引擎等均设置正确）。

```
sage: from sage.misc.latex import latex_examples
sage: foo = latex_examples.diagram()
sage: foo
```

(续下页)

(接上页)

```
LaTeX example for testing display of a commutative diagram produced
by xypic.

To use, try to view this object -- it will not work.  Now try
'latex.add_to_preamble("\\usepackage[matrix,arrow,curve,cmtip]{xy}")',
and try viewing again. You should get a picture (a part of the diagram arising
from a filtered chain complex).
```

为了展示如何处理复杂的 LaTeX 表达式，让我们看一下使用 `tkz-graph` LaTeX 包的组合图示例。

备注

`tkz-graph` LaTeX 包建立在 `pgf` 库的 `tikz` 前端之上。渲染组合图需要 `pgf` 库以及文件 `tkz-graph.sty` 和 `tkz-berge.sty`。它们很可能已经是系统范围内 TeX 安装的一部分。即使不是，也应当很容易找到安装指南。

首先，我们通过将相关包添加到 LaTeX 文档的导言部分来确保它们被包含在内。

```
sage: latex.extra_preamble('\\usepackage{tikz}\n\\usepackage{tkz-graph}\n'
....:                      '\\usepackage{tkz-berge}\n\\usetikzlibrary{arrows,shapes}')
```

当使用 dvi 文件作为中间格式时，图形无法正确生成，因此最好将 LaTeX 引擎设置为 `pdflatex` 可执行文件。

```
sage: latex.engine('pdflatex')
```

此时，像 `view(graphs.CompleteGraph(4))` 这样的命令应该生成一个带有完整图 K_4 适当图像的 PDF。

实际上，可以省略前面的步骤，因为导言部分会自动正确设置，并且 `pdflatex` 是 Sage 的默认 LaTeX 引擎。重新启动 Sage 后再次尝试该命令。

注意，通过 `tkz-graph` 有多种选项可以影响 LaTeX 中图形的呈现方式，这超出了本节的范围。请参阅参考手册 sage.graphs.graph_latex 章节获取指令和详细信息。

5.4 SageTeX

SageTeX 是一个可以进一步集成 TeX 和 Sage 的程序。它是一组 TeX 宏，允许 LaTeX 文档包含指令，让 Sage 计算各种对象并使用 `latex()` 格式化对象。更多信息请参见*使用 SageTeX*。

CHAPTER 6

编程

6.1 加载和附加 Sage 文件

接下来我们说明如何将写在单独文件中的程序加载到 Sage 中。创建一个名为 `example.sage` 的文件，并写入以下内容：

```
print("Hello World")
print(2^3)
```

你可以使用 `load` 命令读取并执行 `example.sage` 文件。

```
sage: load("example.sage")
Hello World
8
```

你也可以使用 `attach` 命令将 Sage 文件附加到运行的会话中：

```
sage: attach("example.sage")
Hello World
8
```

现在如果你修改 `example.sage` 文件并在 Sage 中输入一个空行（即按下回车键），那么 `example.sage` 的内容将会自动重新加载到 Sage 中。

特别是，`attach` 命令会在文件更改时自动重新加载文件，这在调试代码时非常方便，而 `load` 命令仅加载文件一次。

当 Sage 加载 `example.sage` 时，它会将其转换为 Python，然后由 Python 解释器执行。此转换非常简单；它主要是将整型字面量包装在 `Integer()` 中，将浮点型字面量包装在 `RealNumber()` 中，将 `^` 替换为 `**`，并将例如 `R.2` 替换为 `R.gen(2)`。转换后的 `example.sage` 版本包含在与 `example.sage` 相同的目录中，名为 `example.sage.py`。该文件包含以下代码：

```
print("Hello World")
print(Integer(2)**Integer(3))
```

整型字面量被包装，^ 被替换为 **。（在 Python 中，^ 表示“异或”，而 ** 表示“幂运算”。）

（这种预解析由 sage/misc/interpreter.py 模块实现。）

只要有换行符来创建新块（在文件中则无需如此），你就可以将多行缩进代码粘贴到 Sage 中。然而，更好的方式是将这些代码保存到文件中，并如上所述使用 attach 命令来加载。

6.2 创建编译代码

速度在数学计算中至关重要，因为更快的计算可以大大提高效率。尽管 Python 是一种非常方便的高级语言，但如果使用静态类型的编译型语言实现某些计算，其速度可以比用 Python 实现快几个数量级。如果 Sage 完全用 Python 编写，那么在某些方面速度会过于缓慢。为了应对这种情况，Sage 支持一种编译“版本”的 Python，称为 Cython ([Cyt] 和 [Pyr])。Cython 类似于 Python 和 C 语言。大多数 Python 结构，包括列表推导式、条件表达式、类似 += 这样的代码都支持；你还可以导入其他 Python 模块中编写的代码。此外，你还可以声明任意的 C 变量，并直接调用任意的 C 库函数。生成的代码会转换为 C，并使用 C 编译器进行编译。

为了创建你自己的编译 Sage 代码，请将文件命名为 .spyx 扩展名（而非 .sage）。如果使用命令行界面，你可以像处理解释代码一样附加和加载编译代码（目前，Notebook 界面不支持附加和加载 Cython 代码）。实际编译是在“幕后”完成的，你无需进行任何显式操作。编译后的共享对象库存储在 $HOME/.sage/temp/hostname/pid/spyx 中。这些文件将在退出 Sage 时删除。

Sage 预解析不适用于 spyx 文件，例如，1/3 在 spyx 文件中结果为 0，而不是有理数 1/3。如果 foo 是 Sage 库中的一个函数，要想在 spyx 文件中使用它，请导入 sage.all 并使用 sage.all.foo。

```
import sage.all
def foo(n):
    return sage.all.factorial(n)
```

6.2.1 访问单独文件中的 C 函数

访问定义在单独 *.c 文件中的 C 函数也很容易。以下是一个示例。在同一目录下创建文件 test.c 和 test.spyx，内容如下：

纯 C 代码：test.c

```
int add_one(int n) {
  return n + 1;
}
```

Cython 代码：test.spyx:

```
cdef extern from "test.c":
    int add_one(int n)

def test(n):
    return add_one(n)
```

然后进行以下操作：

```
sage: attach("test.spyx")
Compiling (...)/test.spyx...
sage: test(10)
11
```

如果需要额外的库 foo 来编译从 Cython 文件生成的 C 代码，在 Cython 源代码中添加 clib foo。类似地，可以使用声明 cfile bar 将额外的 C 文件 bar 包含在编译中。

6.3 独立 Python/Sage 脚本

以下独立 Sage 脚本可以分解整数、多项式等：

```
#!/usr/bin/env sage

import sys

if len(sys.argv) != 2:
    print("Usage: %s <n>" % sys.argv[0])
    print("Outputs the prime factorization of n.")
    sys.exit(1)

print(factor(sage_eval(sys.argv[1])))
```

为了使用此脚本，`SAGE_ROOT` 必须包含在 PATH 中。如果将上述脚本命名为 `factor`，则以下是使用示例：

```
$ ./factor 2006
2 * 17 * 59
```

6.4 数据类型

在 Sage 中，每个对象都有一个明确的类型。Python 有各种基本内置类型，而 Sage 库还增加了更多类型。Python 内置类型包括字符串、列表、元组、整型和浮点型等，如下所示：

```
sage: s = "sage"; type(s)
<... 'str'>
sage: s = 'sage'; type(s)       # you can use either single or double quotes
<... 'str'>
sage: s = [1,2,3,4]; type(s)
<... 'list'>
sage: s = (1,2,3,4); type(s)
<... 'tuple'>
sage: s = int(2006); type(s)
<... 'int'>
sage: s = float(2006); type(s)
<... 'float'>
```

除此之外，Sage 还添加了许多其他类型。例如，向量空间：

```
sage: V = VectorSpace(QQ, 1000000); V
Vector space of dimension 1000000 over Rational Field
sage: type(V)
<class 'sage.modules.free_module.FreeModule_ambient_field_with_category'>
```

只有某些函数可以在 `V` 上调用。在其他数学软件系统中，这些函数可以使用“函数”符号 `foo(V,...)` 来调用。在 Sage 中，某些函数附加到 `V` 类型（或类），并使用与 Java 或 C++ 类似的面向对象语法调用，例如 `V.foo(...)`。这种方式有助于保持全局命名空间的整洁，并允许名称相同但行为不同的函数存在，而无需通过参数类型检查（或 case 语句）来决定调用哪个函数。此外，如果你重复使用函数名，该函数仍然可用（例如，如果你调用某个函数 `zeta`，那么要计算 Riemann-Zeta 函数在 0.5 处的值，可以输入 `s=.5; s.zeta()`）。

```
sage: zeta = -1
sage: s=.5; s.zeta()
-1.46035450880959
```

在某些非常常见的情况下，为了方便起见，同时避免使用面向对象符号可能导致数学表达式看起来令人困惑，Sage 也支持常规的函数符号。这里有一些例子。

```
sage: n = 2; n.sqrt()
sqrt(2)
sage: sqrt(2)
sqrt(2)
sage: V = VectorSpace(QQ,2)
sage: V.basis()
    [(1, 0), (0, 1)]
sage: basis(V)
    [(1, 0), (0, 1)]
sage: M = MatrixSpace(GF(7), 2); M
Full MatrixSpace of 2 by 2 dense matrices over Finite Field of size 7
sage: A = M([1,2,3,4]); A
[1 2]
[3 4]
sage: A.charpoly('x')
x^2 + 2*x + 5
sage: charpoly(A, 'x')
x^2 + 2*x + 5
```

要列出 A 的所有成员函数，请使用 tab 补全功能。只需输入 `A.`，然后在键盘上按 `[tab]` 键即可，如*反向搜索与 Tab 补全* 中所述。

6.5 列表、元组和序列

列表数据类型具有存储任意类型元素的功能。和 C、C++ 等语言类似（但与大多数标准的计算机代数系统不同），列表元素的索引是从 0 开始的：

```
sage: v = [2, 3, 5, 'x', SymmetricGroup(3)]; v
[2, 3, 5, 'x', Symmetric group of order 3! as a permutation group]
sage: type(v)
<... 'list'>
sage: v[0]
2
sage: v[2]
5
```

（在列表中进行索引时，不一定需要使用 Python 的整型作为索引！）Sage 整数（或有理数，或任何具有 `__index__` 方法的对象）都可以正常使用。

```
sage: v = [1,2,3]
sage: v[2]
3
sage: n = 2       # Sage Integer
sage: v[n]        # Perfectly OK!
3
sage: v[int(n)]   # Also OK.
3
```

`range` 函数创建一个包含 Python 整型（而不是 Sage 整数）元素的列表：

```
sage: list(range(1, 15))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

该函数在使用列表推导式构造列表时非常有用：

```
sage: L = [factor(n) for n in range(1, 15)]
sage: L
[1, 2, 3, 2^2, 5, 2 * 3, 7, 2^3, 3^2, 2 * 5, 11, 2^2 * 3, 13, 2 * 7]
sage: L[12]
13
sage: type(L[12])
<class 'sage.structure.factorization_integer.IntegerFactorization'>
sage: [factor(n) for n in range(1, 15) if is_odd(n)]
[1, 3, 5, 7, 3^2, 11, 13]
```

有关如何使用列表推导式创建列表的更多内容，请参考 [PyT] 。

列表切片是一个非常好的功能。如果 `L` 是一个列表，那么 `L[m:n]` 返回从第 m 个元素开始到第 $(n-1)$ 个元素结束的子列表，如下所示：

```
sage: L = [factor(n) for n in range(1, 20)]
sage: L[4:9]
[5, 2 * 3, 7, 2^3, 3^2]
sage: L[:4]
[1, 2, 3, 2^2]
sage: L[14:4]
[]
sage: L[14:]
[3 * 5, 2^4, 17, 2 * 3^2, 19]
```

元组与列表类似，只不过它是不可变的，一旦创建便不能更改。

```
sage: v = (1,2,3,4); v
(1, 2, 3, 4)
sage: type(v)
<... 'tuple'>
sage: v[1] = 5
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
```

序列是第三种面向列表的 Sage 类型。与列表和元组不同，序列不是 Python 内置类型。默认情况下，序列是可变的，但可以使用 `Sequence` 类方法 `set_immutable` 将其设置为不可变，如以下例子所示。序列的所有元素都属于同一个父对象，称为序列的领域 (universe)。

```
sage: v = Sequence([1,2,3,4/5])
sage: v
[1, 2, 3, 4/5]
sage: type(v)
<class 'sage.structure.sequence.Sequence_generic'>
sage: type(v[1])
<class 'sage.rings.rational.Rational'>
sage: v.universe()
Rational Field
sage: v.is_immutable()
False
sage: v.set_immutable()
sage: v[0] = 3
Traceback (most recent call last):
...
ValueError: object is immutable; please change a copy instead.
```

序列派生自列表，可以在任何需要列表的地方使用：

```
sage: v = Sequence([1,2,3,4/5])
sage: isinstance(v, list)
True
sage: list(v)
[1, 2, 3, 4/5]
sage: type(list(v))
<... 'list'>
```

另一个例子是，向量空间的基是不可变序列，因为不能改变它们至关重要。

```
sage: V = QQ^3; B = V.basis(); B
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
sage: type(B)
<class 'sage.structure.sequence.Sequence_generic'>
sage: B[0] = B[1]
Traceback (most recent call last):
...
ValueError: object is immutable; please change a copy instead.
sage: B.universe()
Vector space of dimension 3 over Rational Field
```

6.6 字典

字典（有时也被称为关联数组）是从“可哈希”对象（例如字符串、数字和元组等；详情请参见 Python 文档 http://docs.python.org/tut/node7.html 和 http://docs.python.org/lib/typesmapping.html ）到任意对象的映射。

```
sage: d = {1:5, 'sage':17, ZZ:GF(7)}
sage: type(d)
<... 'dict'>
sage: list(d.keys())
[1, 'sage', Integer Ring]
sage: d['sage']
17
sage: d[ZZ]
Finite Field of size 7
sage: d[1]
5
```

第三个键说明字典的索引可以很复杂，例如整数环。

你可以将上述字典转换为具有相同数据的列表：

```
sage: list(d.items())
[(1, 5), ('sage', 17), (Integer Ring, Finite Field of size 7)]
```

一种常见用法是遍历字典中的键值对：

```
sage: d = {2:4, 3:9, 4:16}
sage: [a*b for a, b in d.items()]
[8, 27, 64]
```

正如最后的输出所示，字典是无序的。

6.7 集合

Python 有内建的集合类型。它提供的主要功能是快速查找元素是否在集合中，以及标准集合论运算。

```
sage: X = set([1,19,'a']);   Y = set([1,1,1, 2/3])
sage: X   # random sort order
{1, 19, 'a'}
sage: X == set(['a', 1, 1, 19])
True
sage: Y
{2/3, 1}
sage: 'a' in X
True
sage: 'a' in Y
False
sage: X.intersection(Y)
{1}
```

Sage 也有自己的集合类型（在某些情况下使用 Python 内建集合类型实现），但具有一些与 Sage 相关的额外功能。使用 `Set(...)` 来创建 Sage 集合。例如：

```
sage: X = Set([1,19,'a']);   Y = Set([1,1,1, 2/3])
sage: X   # random sort order
{'a', 1, 19}
sage: X == Set(['a', 1, 1, 19])
True
sage: Y
{1, 2/3}
sage: X.intersection(Y)
{1}
sage: print(latex(Y))
\left\{1, \frac{2}{3}\right\}
sage: Set(ZZ)
Set of elements of Integer Ring
```

6.8 迭代器

迭代器是 Python 最近添加的功能，在数学应用中特别有用。这里有几个例子；详情请参见 [PyT] 。我们创建一个非负整数平方的迭代器，上限为 10000000。

```
sage: v = (n^2 for n in range(10000000))
sage: next(v)
0
sage: next(v)
1
sage: next(v)
4
```

我们创建一个 $4p+1$ 形式的素数迭代器，其中 p 也是素数，并查看前几个值。

```
sage: w = (4*p + 1 for p in Primes() if is_prime(4*p+1))
sage: w         # in the next line, 0xb0853d6c is a random 0x number
<generator object at 0xb0853d6c>
sage: next(w)
13
sage: next(w)
```

(续下页)

(接上页)

```
29
sage: next(w)
53
```

某些环，例如有限域和整数环有与之关联的迭代器：

```
sage: [x for x in GF(7)]
[0, 1, 2, 3, 4, 5, 6]
sage: W = ((x,y) for x in ZZ for y in ZZ)
sage: next(W)
(0, 0)
sage: next(W)
(0, 1)
sage: next(W)
(0, -1)
```

6.9 循环、函数、控制语句和比较

我们已经看过了一些常见的 `for` 循环用法示例。在 Python 中，`for` 循环具有缩进结构，例如：

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
```

请注意 for 语句末尾的冒号（不像 GAP 或 Maple 中有”do” 或”od”），以及循环体（即 `print(i)`）前的缩进。这个缩进非常重要。在 Sage 中，当你在”:” 后按下 `enter` 时，会自动添加缩进，如下所示。

```
sage: for i in range(5):
....:     print(i)  # now hit enter twice
....:
0
1
2
3
4
```

符号 = 用于赋值。符号 == 用于检查相等：

```
sage: for i in range(15):
....:     if gcd(i,15) == 1:
....:         print(i)
....:
1
2
4
7
8
11
13
14
```

请牢记缩进如何决定 `if`, `for` 和 `while` 语句的块结构：

```
sage: def legendre(a,p):
....:     is_sqr_modp=-1
....:     for i in range(p):
....:         if a % p == i^2 % p:
....:             is_sqr_modp=1
....:     return is_sqr_modp

sage: legendre(2,7)
1
sage: legendre(3,7)
-1
```

当然，这不是勒让德符号 (Legendre symbol) 的高效实现！它只是为了说明 Python/Sage 编程的各个方面。Sage 附带的函数 {kronecker}，可以通过调用 PARI 的 C 库高效地计算勒让德符号。

最后，我们注意到数字之间的比较，如 ==, !=, <=, >=, >, <，会自动将两个数字转换为相同类型（如果可能的话）：

```
sage: 2 < 3.1; 3.1 <= 1
True
False
sage: 2/3 < 3/2;   3/2 < 3/1
True
True
```

使用 bool 来判断符号不等式：

```
sage: x < x + 1
x < x + 1
sage: bool(x < x + 1)
True
```

在比较不同类型的对象时，在大多数情况下，Sage 会尝试找到两者的共同复结构（参见父结构、转换与强制转换 了解更多细节）。如果成功，比较将在强制转换的对象之间进行；如果不成功，则认为对象不相等。要测试两个变量是否引用同一个对象，请使用 `is`。在下面这个示例中我们将看到，Python 整型 `1` 是唯一的，而 Sage 整型 `1` 则不是：

```
sage: 1 is 2/2
False
sage: 1 is 1
False
sage: 1 == 2/2
True
```

在以下两行代码中，第一个等式为 `False`，因为没有从 $\mathbf{Q} \to \mathbf{F}_5$ 的标准同态，因此无法将 $\mathbf{F}_5$ 中的 1 与 $1 \in \mathbf{Q}$ 进行比较。相反，由于存在从 $\mathbf{Z} \to \mathbf{F}_5$ 的标准映射，因此第二个比较为 `True`。需要注意的是，顺序不影响结果。

```
sage: GF(5)(1) == QQ(1); QQ(1) == GF(5)(1)
False
False
sage: GF(5)(1) == ZZ(1); ZZ(1) == GF(5)(1)
True
True
sage: ZZ(1) == QQ(1)
True
```

警告: Sage 中的比较比 Magma 更严格，Magma 会声明 $1 \in \mathbf{F}_5$ 等于 $1 \in \mathbf{Q}$。

```
sage: magma('GF(5)!1 eq Rationals()!1')            # optional - magma
true
```

6.10 性能分析

> “过早优化乃万恶之源。” - Donald Knuth

节作者：*Martin Albrecht <malb@informatik.uni-bremen.de>*

有时检查代码中的瓶颈有助于了解哪些部分占用最多的计算时间；这可以很好地了解哪些部分需要优化。Python（以及 Sage）提供了几种性能分析工具和方法，这个过程称为性能分析。

最简单的方式是使用交互式 shell 中的 `prun` 命令。它会返回一个总结，描述哪些函数花了多少计算时间。例如，要分析有限域上的矩阵乘法（版本 1.0 当前很慢!），可以这样做：

```
sage: k,a = GF(2**8, 'a').objgen()
sage: A = Matrix(k,10,10,[k.random_element() for _ in range(10*10)])
```

```
sage: %prun B = A*A
       32893 function calls in 1.100 CPU seconds

Ordered by: internal time

ncalls tottime percall cumtime percall filename:lineno(function)
 12127  0.160   0.000   0.160  0.000 :0(isinstance)
  2000  0.150   0.000   0.280  0.000 matrix.py:2235(__getitem__)
  1000  0.120   0.000   0.370  0.000 finite_field_element.py:392(__mul__)
  1903  0.120   0.000   0.200  0.000 finite_field_element.py:47(__init__)
  1900  0.090   0.000   0.220  0.000 finite_field_element.py:376(__compat)
   900  0.080   0.000   0.260  0.000 finite_field_element.py:380(__add__)
     1  0.070   0.070   1.100  1.100 matrix.py:864(__mul__)
  2105  0.070   0.000   0.070  0.000 matrix.py:282(ncols)
  ...
```

这里 `ncalls` 是调用次数，`tottime` 是给定函数花费的总时间（不包括调用子函数的时间），`percall` 是 `tottime` 除以 `ncalls` 的商。`cumtime` 是该函数及所有子函数花费的总时间（即，从调用到退出），`percall` 是 `cumtime` 除以原始调用次数的商，`filename:lineno(function)` 提供了每个函数的相关数据。性能分析中的经验法则是：列表越靠前的函数，其代价越高，因而更需要进行优化。

与以往一样，`prun?` 命令提供了使用性能分析器和理解输出详细信息的帮助。

性能分析数据还可以保存到一个对象中，以便进行更详细的检查：

```
sage: %prun -r A*A
sage: stats = _
sage: stats?
```

注意：输入 `stats = prun -r A\*A` 会显示语法错误消息，因为 prun 是 IPython shell 命令而不是常规函数。

为了更好地以图形化方式呈现分析数据，你可以使用 hotshot 分析器，`hotshot2cachetree` 脚本，以及 `kcachegrind` 程序（仅限 Unix）。以下是使用 hotshot 分析器的示例：

```
sage: k,a = GF(2**8, 'a').objgen()
sage: A = Matrix(k,10,10,[k.random_element() for _ in range(10*10)])
sage: import hotshot
```

(续下页)

(接上页)

```
sage: filename = "pythongrind.prof"
sage: prof = hotshot.Profile(filename, lineevents=1)
```

```
sage: prof.run("A*A")
<hotshot.Profile instance at 0x414c11ec>
sage: prof.close()
```

这会在当前工作目录中生成一个 `pythongrind.prof` 文件。现在可以将其转换为 cachegrind 格式进行可视化展示。

在系统终端中，输入

```
$ hotshot2calltree -o cachegrind.out.42 pythongrind.prof
```

现在，输出文件 `cachegrind.out.42` 可以用 `kcachegrind` 查看。请注意，需要遵守命名约定 `cachegrind.out.XX`。

CHAPTER 7

使用 SageTeX

SageTeX 包允许你将 Sage 计算结果嵌入到 LaTeX 文档中。要使用它，需要先“安装”它（请参阅让 *TeX 识别 SageTeX*）。

7.1 示例

以下是一个非常简短的 SageTeX 使用示例。完整文档可以在 `SAGE_ROOT/venv/share/doc/sagetex` 中找到，其中 `SAGE_ROOT` 是 Sage 安装目录。该目录包含文档和示例文件。请参阅 `SAGE_ROOT/venv/share/texmf/tex/latex/sagetex` 以获取一些可能有用的 Python 脚本。

想要了解 SageTeX 的工作原理，请按照 SageTeX 的安装说明（在让 *TeX 识别 SageTeX* 中）操作，并将以下文本复制到一个名为 `st_example.tex` 的文件中：

警告

如果你在“实时”帮助中查看此内容，下面的文本会有几个未知控制序列的错误。请使用静态版查看正确的文本。

```
\documentclass{article}
\usepackage{sagetex}

\begin{document}

Using Sage\TeX, one can use Sage to compute things and put them into
your \LaTeX{} document. For example, there are
$\sage{number_of_partitions(1269)}$ integer partitions of $1269$.
You don't need to compute the number yourself, or even cut and paste
it from somewhere.

Here's some Sage code:

\begin{sageblock}
```

(续下页)

(接上页)

```
    f(x) = exp(x) * sin(2*x)
\end{sageblock}

The second derivative of $f$ is

\[
  \frac{\mathrm{d}^{2}}{\mathrm{d}x^{2}} \sage{f(x)} =
  \sage{diff(f, x, 2)(x)}.
\]

Here's a plot of $f$ from $-1$ to $1$:

\sageplot{plot(f, -1, 1)}

\end{document}
```

像往常一样在 `st_example.tex` 上运行 LaTeX。请注意 LaTeX 会有一些警告，其中包括：

```
Package sagetex Warning: Graphics file
sage-plots-for-st_example.tex/plot-0.eps on page 1 does not exist. Plot
command is on input line 25.

Package sagetex Warning: There were undefined Sage formulas and/or
plots. Run Sage on st_example.sagetex.sage, and then run LaTeX on
st_example.tex again.
```

请注意，除了 LaTeX 产生的常规文件集合外，还有一个名为 `st_example.sagetex.sage` 的文件。这是在 `st_example.tex` 上运行 LaTeX 时生成的 Sage 脚本。警告信息告诉你在 `st_example.sagetex.sage` 上运行 Sage，请听从建议并进行操作。它会告诉你再次在 `st_example.tex` 上运行 LaTeX，但在此之前，请注意新文件 `st_example.sagetex.sout` 已被创建。该文件包含 Sage 计算结果，可供 LaTeX 插入到你的文本中。还创建了一个包含 EPS 文件的新目录。再次运行 LaTeX，你会看到 Sage 计算和绘图的所有内容已包含在你的文档中。

上面使用的各种宏应该很容易理解。`sageblock` 环境按原样排版你的代码，并在运行 Sage 时执行代码。当你执行 `\sage{foo}` 时，插入到文档中的结果就是在 Sage 内部运行 `latex(foo)` 得到的结果。绘图命令稍微复杂一些，但在最简单形式下，`\sageplot{foo}` 插入的是由 `foo.save('filename.eps')` 得到的图像。

一般来说，操作步骤是：

- 在.tex 文件上运行 LaTeX；
- 在生成的.sage 文件上运行 Sage；
- 再次运行 LaTeX。

如果文档中没有更改任何 Sage 命令，则可以省略运行 Sage。

SageTeX 还有很多内容，由于 Sage 和 LaTeX 都是复杂且强大的工具，建议阅读 SageTeX 的文档 `SAGE_ROOT/venv/share/doc/sagetex`。

7.2 让 TeX 识别 SageTeX

Sage 基本上是自包含的，但某些部分需要进行一些干预才能正常工作。SageTeX 就是其中之一。

SageTeX 包允许在 LaTeX 文档中嵌入来自 Sage 的计算和绘图。Sage 中默认安装了 SageTeX，但要在 LaTeX 文档中使用 SageTeX，你需要先让 TeX 识别它。

关键在于 TeX 需要能够找到 sagetex.sty，该文件位于 SAGE_ROOT/venv/share/texmf/tex/latex/sagetex/，其中 SAGE_ROOT 是你构建或安装 Sage 的目录。如果 TeX 能找到 sagetex.sty，那么 SageTeX 就可以工作。有几种方法可以实现这一点。

- 第一种方法，也是最简单的方法是将 sagetex.sty 复制到与 LaTeX 文档相同的目录中。在排版文档时，总会搜索当前目录，因此这种方法始终有效。

  但这种方法有两个小问题：首先，会在计算机上产生很多不必要的 sagetex.sty 拷贝。其次，更严重的问题是，如果升级 Sage 并获得新版本的 SageTeX，Python 代码和 SageTeX 的 LaTeX 代码可能不再匹配，从而导致错误。

- 第二种方法是使用 TEXMFLOCAL 环境变量。如果你使用的是 bash shell，可以这样做：

  ```
  $ export TEXMFLOCAL=SAGE_ROOT/venv/share/texmf
  $ mktexlsr       # update kpathsea ls-R databases
  ```

  其中 SAGE_ROOT 是 Sage 安装位置。之后，TeX 和相关程序将找到 SageTeX 样式文件。如果你想使这个更改持续生效，可以将上述第一行添加到 .bashrc 文件中。如果你使用的是不同的 shell，可能需要调整以上命令从而让环境变量可被识别；请查阅所用 shell 的文档以了解如何操作。

  如果你移动了 Sage 的安装目录或在新目录中安装了新版本，需要用新的 SAGE_ROOT 更新上述命令。

- 让 TeX 识别 sagetex.sty 的第三种（也是最佳的）方法，是将该文件复制到主目录中的一个方便的位置。大多数 TeX 发行版会自动搜索主目录中的 texmf 目录以寻找包。要确切了解这个目录的位置，请在命令行种执行以下操作：

  ```
  $ kpsewhich -var-value=TEXMFHOME
  ```

  这将打印出一个目录，例如 /home/drake/texmf 或 /Users/drake/Library/texmf。使用如下命令将 SAGE_ROOT/venv/share/texmf/ 中的 tex/ 目录复制到主目录的 texmf 目录：

  ```
  $ cp -R SAGE_ROOT/venv/share/texmf/tex TEXMFHOME
  ```

  其中 SAGE_ROOT 仍然是 Sage 的安装位置，TEXMFHOME 是 kpsewhich 命令的结果。

  如果你升级了 Sage 并发现 SageTeX 无法工作，可以简单地重复上述步骤以确保 SageTeX 的 Sage 部分和 TeX 部分再次同步。

- 对于多用户系统上的安装，只需适当修改上述指令，将 sagetex.sty 复制到系统范围的 TeX 目录中。最好的选择可能是使用以下结果，而不是 TEXMFHOME 目录：

  ```
  $ kpsewhich -var-value=TEXMFLOCAL
  ```

  这很可能会产生类似于 /usr/local/share/texmf 的结果。按照上述方式将 tex 目录复制到 TEXMFLOCAL 目录中。现在需要通过运行以下命令更新 TeX 的包数据库：

  ```
  $ texhash TEXMFLOCAL
  ```

  以 root 身份，适当替换 TEXMFLOCAL。现在系统中所有用户都可以访问 LaTeX 包，如果他们也能运行 Sage，他们就可以使用 SageTeX。

警告

确保 LaTeX 在排版文档时使用的 sagetex.sty 文件与 SageTeX 使用的版本匹配，这一点至关重要。如果你升级了 Sage，应该删除所有旧版本的 sagetex.sty。

由于此问题，我们建议将 SageTeX 文件复制到主目录的 texmf 目录中（上述第 3 种方法）。这样，升级 Sage 时，仅需做一件事（复制目录）即可确保 SageTeX 正常工作。

7.3 SageTeX 文档

虽然这不严格属于安装的一部分，但值得在此提及的是，SageTeX 的文档维护在 `SAGE_ROOT/venv/share/doc/sagetex/sagetex.pdf`。同一目录中还有一个示例文件 -- 请参见 `example.tex` 和 `example.pdf`，这是使用 LaTeX 和 Sage 对该文件进行排版的预生成结果。你也可以从 SageTeX 页面 获取这些文件。

7.4 SageTeX 与 TeXLive

一个潜在的令人困惑的问题是流行的 TeX 发行版 TeXLive 包含 SageTeX。虽然看起来很方便，但对于 SageTeX 而言，确保 Sage 部分和 LaTeX 部分同步是非常重要的 -- 在这种情况下，这就成为了一个问题，因为由操作系统发行版或软件包管理器提供的 TeXLive 可能与官方 TeXLive 分发版本不同步，而后者也可能与当前的 SageTeX 版本不同步。

因此，强烈建议你始终按照上面的说明，从 Sage 安装 SageTeX 的 LaTeX 部分。上述说明将确保 SageTeX 的两个部分兼容并正常工作。

CHAPTER 8

后记

8.1 为什么选择 Python？

8.1.1 Python 的优势

尽管必须快速执行的代码是用编译型语言实现的，但 Sage 的主要实现语言是 Python（见 [Py] ）。Python 具有以下几点优势：

- **对象保存**在 Python 中得到了很好的支持。Python 广泛支持将（几乎）任意对象保存到磁盘文件或数据库中。
- 源代码中对函数和包的 **文档**支持非常好，包括自动提取文档和自动测试所有示例。这些示例会定期自动测试，并保证如预期工作。
- **内存管理**：Python 现有的内存管理器和垃圾收集器设计精巧且强大，可以正确处理循环引用，并允许文件中的局部变量。
- Python 拥有许多对 Sage 用户非常有用的 **包**：数值分析和线性代数，2D 和 3D 可视化，网络（用于分布式计算和服务，例如通过 twisted），数据库支持等。
- **可移植性**：Python 在大多数平台上，只需几分钟即可轻松从源代码编译 Python。
- **异常处理**：Python 拥有复杂且精巧的异常处理系统，即使代码出现错误，程序也能优雅地恢复。
- **调试器**：Python 包含调试器，因此当代码由于某种原因失败时，用户可以访问详尽的堆栈跟踪，检查所有相关变量的状态，并上下移动堆栈。
- **性能分析器**：Python 拥有性能分析器，它会运行代码并创建一份详细报告，说明每个函数被调用的次数和时间。
- **一门语言**：不同于 Magma、Maple、Mathematica、Matlab、GP/PARI、GAP、Macaulay 2、Simath 等那样为数学编写一门 **新语言**，我们使用 Python 语言，它是一种流行的计算机语言，由数百名经验丰富的软件工程师积极开发和优化。Python 是一个重要的开源成功案例，拥有成熟的开发流程（见 [PyDev] ）。

8.1.2 预解析器：Sage 与 Python 之间的区别

Python 的一些数学方面可能会令人困惑，因此 Sage 在多个方面的行为与 Python 不同。

- **指数运算符的表示法**：`**` vs `^`。在 Python 中，`^` 表示“异或”，而不是指数运算。因此在 Python 中我们有：

```
>>> 2^8
10
>>> 3^2
1
>>> 3**2
9
```

`^` 的这种用法可能略显奇怪，并且对于纯数学研究来说效率不高，因为“异或”函数很少使用。为了方便起见，Sage 在将所有命令行传递给 Python 之前都会进行预解析，将不在字符串中的 `^` 替换为 `**`：

```
sage: 2^8
256
sage: 3^2
9
sage: "3^2"
'3^2'
```

Sage 中的按位异或运算符是 `^^`。这也适用于就地运算符对于就地运算符 `^^=`：

```
sage: 3^^2
1
sage: a = 2
sage: a ^^= 8
sage: a
10
```

- **整数除法**：Python 表达式 `2/3` 并不像数学家们所预期的那样：`2/3` 返回浮点数 `0.6666...`。请注意 `//` 是欧几里得除法，`2//3` 返回 `0`。

  我们在 Sage 解释器中通过将整型字面量包装在 `Integer( )` 中，并使除法作为有理数的构造函数来处理这个问题。例如：

```
sage: 2/3
2/3
sage: (2/3).parent()
Rational Field
sage: 2//3
0
```

- **长整数**：Python 原生支持除 C int 类型外的任意精度整数。这些原生整数的性能显著低于 GMP 所提供的。Sage 使用 GMP C 库来实现任意精度整数。

与某些人为了内部项目修改 Python 解释器不同，我们完全按照原样使用 Python 语言，并为 IPython 编写预解析器，使 IPython 的命令行行为符合数学家的预期。这意味着任何现有的 Python 代码都可以在 Sage 中使用。然而，仍需遵守标准的 Python 规则，以便编写能够导入 Sage 的包。

（例如，要安装在互联网上找到的 Python 库，请按照说明进行操作，但使用 `sage -python` 而不是 `python`。通常这意味着输入 `sage -python setup.py install`。）

8.2 我想做出一些贡献。我应该怎么做？

如果你想为 Sage 做出贡献，我们会非常感谢你的帮助！贡献可以从实质性代码贡献到向 Sage 添加文档或报告错误。

浏览 Sage 网页以获取开发者信息。你还可以找到一份按优先级和类别排序的 Sage 相关项目列表。Sage 开发者指南 也有一些有用的信息，你还可以查看 `sage-devel` Google 讨论组。

8.3 如何引用 Sage？

如果你在论文中使用了 Sage，当引用使用 Sage 的计算时，包含以下内容：如果你使用 Sage 撰写论文，请将以下内容添加到参考文献中来引用使用 Sage 完成的计算

```
[Sage] SageMath, the Sage Mathematics Software System (Version 8.7),
       The Sage Developers, 2019, https://www.sagemath.org.
```

（将 8.7 替换为你使用的 Sage 版本）。此外，请尝试追踪在计算中使用了哪些 Sage 组件，例如 PARI?, GAP?, Singular?, Maxima? 并引用这些系统。如果你不确定计算使用了哪个软件，可以随时在 `sage-devel` Google 讨论组上提问。有关这一点的进一步讨论，请参阅单变量多项式。

如果你恰好刚刚读完这篇教程，并且知道花了多长时间，请在 `sage-devel` Google 讨论组上告诉我们。

祝使用 Sage 愉快！

CHAPTER 9

附录

9.1 算术二元运算符的优先级

`3^2*4 + 2%5` 的结果是什么？这里的结果 (38) 取决于下面的 “运算符优先级表”。下面的表格基于 G. Rossum 和 F. Drake 编写的 *Python 语言参考手册* §5.14 中的表格。这里列出的操作按优先级从低到高排列。

运算符	描述
or	布尔或
and	布尔与
not	布尔非
in, not in	成员判断
is, is not	同一性测试
>, <=, >, >=, ==, !=	比较
+, -	加法，减法
*, /, %	乘法，除法，取余
**, ^	幂

因此，为了计算 `3^2*4 + 2%5`，Sage 将计算过程括号化为: `((3^2)*4) + (2%5)`。从而，首先计算 `3^2`，结果为 `9`，然后分别计算 `(3^2)*4` 和 `2%5`，最后将结果相加。

CHAPTER 10

参考文献

CHAPTER 11

索引与表格

- genindex
- search

Bibliography

[Cyt] Cython, http://www.cython.org

[GAP] The GAP Group, GAP - Groups, Algorithms, and Programming, Version 4.4; 2005, https://www.gap-system.org

[GAPkg] GAP Packages, https://www.gap-system.org/Packages/packages.html

[GP] PARI/GP, https://pari.math.u-bordeaux.fr/

[Mag] Magma, http://magma.maths.usyd.edu.au/magma/

[Max] Maxima, http://maxima.sf.net/

[NagleEtAl2004] Nagle, Saff, and Snider. *Fundamentals of Differential Equations*. 6th edition, Addison-Wesley, 2004.

[Py] Python 编程语言, http://www.python.org/

[PyB] Python 初学者手册, https://wiki.python.org/moin/BeginnersGuide

[PyDev] Python 开发者手册, https://docs.python.org/devguide/

[PyLR] Python 标准库, https://docs.python.org/3/library/index.html

[Pyr] Pyrex, http://www.cosc.canterbury.ac.nz/~greg/python/Pyrex/

[PyT] Python 教程, https://docs.python.org/3/tutorial/

[SA] Sage 官网, https://www.sagemath.org/

[Si] G.-M. Greuel, G. Pfister, and H. Schönemann. Singular 3.0. A Computer Algebra System for Polynomial Computations. Center for Computer Algebra, University of Kaiserslautern (2005). https://www.singular.uni-kl.de

[SJ] William Stein, David Joyner, Sage: System for Algebra and Geometry Experimentation, Comm. Computer Algebra {39} (2005) 61-64.

[ThreeJS] three.js, http://threejs.org

索引

非字母

E