Sage Tutorial in Russian

Выпуск 6.8

The Sage Development Team

29 July 2015

Оглавление

Sage — это бесплатное и свободно распространяемое математическое программное обеспечение с открытыми исходными кодами для исследовательской работы и обучения в самых различных областях включая алгебру, геометрию, теорию чисел, криптографию, численные вычисления и другие. Как модель разработки Sage, так и условия его распространения и использования выбраны в соответствии с принципами открытой и совместной работы: мы собираем машину, а не переизобретаем колесо. Одной из основных целей Sage является создание доступной, бесплатной и открытой альтернативы Maple, Matematica, Magma и MATLAB.

Глава 1

Введение

Данное учебное пособие — лучший способ познакомиться с Sage за несколько часов. Вы можете использовать его в HTML или PDF формате, а также открыть интерактивную версию для непосредственной работы в Sage notebook: нажмите `Help`, потом `Tutorial`. (Интерактивная версия может быть недоступна на русском языке, но может быть более полной и точнее соответствовать текущей версии Sage.)

Существенная часть Sage написана на языке программирования Python, однако его знание не требуется для чтения данного пособия. Если Вы пожелаете узнать больше о Python (очень элегантный язык!), существует много прекрасных (и бесплатных) источников, таких как [PyT] и [Dive]. Для первого же знакомства с Sage данное пособие является отличной отправной точкой. Итак:

```
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()
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)      # Точность 30 бит
0.16549568 - 0.052149208*I
sage: latex(k)
\frac{36}{20 \, \sqrt{73} + 36 i \, \sqrt{3} + 27}
```

1.1 Установка

Если на вашем компьютере не установлен Sage, и вы хотите попробовать некоторые команды, воспользуйтесь этой ссылкой: http://www.sagenb.org.

Руководство по установке Sage можно просмотреть на главной странице Sage в разделе документации: [SA] Здесь мы приведем лишь несколько комментариев:

1. Загруженный файл установки Sage является самодостаточным. То есть, хотя Sage использует Python, IPython, PARI, GAP, Singular, Maxima, NTL, GMP и т.д., отдельной установки вышеперечисленных пакетов не требуется, так как они уже включены. Однако, для использования некоторых функций Sage таких, как Macaulay или KASH, вы должны установить требующиеся файлы или иметь соответствующие программы на вашем компьютере. Macaulay и KASH являются пакетами Sage (для просмотра списка доступных пакетов введите `sage -optional` или изучите раздел "Download" на веб-сайте Sage).

2. Предварительно скомпилированную бинарную версию Sage, которую также можно найти на вебсайте, будет легче установить, чем версию в исходном коде. Просто распакуйте и выполните `sage`.

3. Если вы желаете использовать пакет SageTeX, который позволяет вставлять результаты вычислений Sage в LaTeX файл, требуется сделать SageTeX известным вашей системе TeX. Для этого изучите секцию "Make SageTeX known to TeX" в Руководстве по установке Sage (данная ссылка ведет к локальному размещению копии руководства по установке). Это довольно просто; вам понадобится всего лишь скопировать один файл в директорию поиска TeX.

   Документация по использованию SageTeX находится в `$SAGE_ROOT/local/share/texmf/tex/generic/sagetex/`, где "`$SAGE_ROOT`" соответствует директории, где установлен сам Sage, например, `/opt/sage-4.2.1`.

1.2 Работа в Sage

Работа в Sage может быть осуществлена несколькими путями:

- **Notebook (графический интерфейс):** см. раздел о Notebook в справочном руководстве, а также *Интерфейс Notebook* ниже;
- **Интерактивная командная строка:** см. *Интерактивная оболочка*;
- **Программы:** создание интерпретируемых и компилируемых программ в Sage (см. *Загрузка и прикрепление файлов Sage* и *Создание компилированного кода*);
- **Скрипты:** создание самостоятельных скриптов на Python, использующих библиотеки Sage (см. *Самостоятельные скрипты Python/Sage*).

1.3 Цели Sage

- **Полезный:** предполагаемая аудитория пользователей Sage — это школьники старших классов, студенты, учителя, профессора и математики-исследователи. Цель: предоставить программное обеспечение, которое было бы полезно для изучения и исследований с помощью математических конструкций в алгебре, геометрии, теории чисел, численных вычислениях и т.д. Sage упрощает интерактивное экспериментирование с помощью математических объектов.
- **Эффективный:** Будьте быстрыми в вычиселниях. Sage использует высокооптимизированное программное обеспечение, как GMP, PARI, GAP, and NTL, и поэтому является очень быстрым в операциях.
- Свободный и открытый: Исходный код должен быть свободно доступным, тем самым предоставляя пользователям возможность понять, что именно выполняется системой, и легко дополнять ее. Так же, как и математики приобретают более глубокое понимание теоремы, углубляясь в ее доказательство, люди, выполняющие вычисления, в силах понять, как эти вычисления производятся, почитав документированный исходный код. Если вы используете вычисления Sage в своих публикациях, вы можете быть уверены, что ваши читатели будут всегда иметь доступ к Sage и всему исходному коду. Вы также можете архивировать и перераспределять используемую версию Sage.
- **Легко компилируемый:** Sage должно быть легко скомпилировать из исходных кодов под GNU/Linux, OS X и Windows. Это предоставит пользователям возможность модифицировать и оптимизировать систему под свои предпочтения.
- **Взаимодействие:** Обеспечить простые и надежные интерфейсы для многих других систем компьютерной алгебры, включая PARI, GAP, Singular, Maxima, KASH, Magma, Maple, and Mathematica. Sage создан для объединения и расширения возможностей существующего математического программного обеспечения.
- **Хорошо документированный:** Вы имеете доступ к учебному пособию, руководству по программированию, справочному руководству и how-to, включающие в себя многочисленные примеры и обсуждение математической подоплеки.
- **Расширяемый:** Объявляйте новые типы данных или расширяйте встроенные, используйте код, написанный во множестве языков.
- **Дружественный:** Вам будет легко понимать функциональность любого объекта, а также просматривать документацию и исходный код. Также имейте в виду высокий уровень поддержки пользователей.

Глава 2

Тур по Sage

Данный раздел покажет, что доступно в Sage. Для других примеров см. “Sage Constructions”, где находятся ответы на часто задаваемые вопросы. Также см. “Справочное руководство Sage”, в котором можно найти тысячи других примеров. С данной экскурсией можно работать интерактивно, если запустить Sage Notebook и нажать на ссылку `Help`.

(Если уроки просматриваются в Sage Notebook, нажмите `shift-enter` для того, чтобы вычислить любую ячейку ввода. До нажатия сочетания клавиш `shift-enter` можно редактировать текст ввода.)

2.1 Присваивание, сравнение и арифметика

С некоторыми исключениями Sage использует язык программирования Python, поэтому многие книги, знакомящие с Python, помогут в изучении Sage.

Sage использует = для присваивания. ==, <=, >=, < и > используются для сравнения:

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

Sage поддерживает все базовые математические операции:

```
sage: 2**3    #  ** означает возведение в степень
8
sage: 2^3     #  в Sage ^ и ** синонимы (в отличие от Python)
8
sage: 10 % 3  #  для целых чисел % означает mod, т.е. взятие остатка
1
sage: 10/4
5/2
sage: 10//4   #  для целых чисел // означает целочисленное частное
2
sage: 4 * (10 // 4) + 10 % 4 == 10
True
```

```
sage: 3^2*4 + 2%5
38
```

Вычисление выражения, такого как `3^2*4 + 2%5`, производится в соответствии со старшинством операций, как описано в *Приоритет бинарных арифметических операторов*.

Sage также поддерживает многие математические функции:

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

Как показывает последний пример, некоторые математические выражения возвращают 'точные' величины, но не численные приближения. Для того, чтобы получить численное приближение, используйте функцию `n` или метод `n` (оба имеют более длинные названия - `numerical_approx`; функция `N` - это то же самое, что и `n`). Они принимают необязательные аргументы `prec`, который определяет количество битов точности, и `digits`, который определяет количество десятичных цифр точности. По умолчанию, применяется 53 бита точности.

```
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 имеет динамический контроль типов, так что значение, на которое ссылается переменная, имеет тип, связанный с ним. Однако, данная переменная может содержать значение любого типа из языка Python:

```
sage: a = 5   # a - целое число
sage: type(a)
<type 'sage.rings.integer.Integer'>
sage: a = 5/3  # теперь a - рациональное число
sage: type(a)
<type 'sage.rings.rational.Rational'>
sage: a = 'hello'  # теперь a - строка
sage: type(a)
<type 'str'>
```

Язык C, который имеет статический контроль типов, существенно отличается; переменная, объявленная как целое число, может содержать только целое число.

Потенциальным источником путаницы в Python является тот факт, что числовая константа, начинающаяся с 0, рассматривается как восьмеричное число, т.е. число по основанию 8:

```
sage: 011
9
sage: 8 + 1
```

```
9
sage: n = 011
sage: n.str(8)   # строка представляющая n по основанию 8
'11'
```

2.2 Получение помощи

В Sage есть исчерпывающая встроенная документация, к которой можно получить доступ, напечатав имя функции или константы с последующим вопросительным знаком:

```
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)
        -0.000092653590058191 3
        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
sage: sudoku?
File:        sage/local/lib/python2.5/site-packages/sage/games/sudoku.py
Type:        <type 'function'>
Definition:  sudoku(A)
Docstring:

    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 также предоставляет возможность 'Автозавершения': напечатайте несколько первых букв названия функции и нажмите `TAB`. Например, если напечатать `ta` и нажать `TAB`, Sage выведет `tachyon, tan, tanh, taylor`. Данная функция является хорошим способом поиска имен функций или других конструкций в Sage.

2.3 Функции, отступы и счетчики

Для того, чтобы определить функцию в Sage, используйте команду `def` и двоеточие после списка имен переменных:

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

Заметка: В зависимости от версии учебного пособия на второй строке этого примера можно увидеть `....:`. Не печатайте их, так как они служат лишь для того, чтобы показать отступы в коде.

Не определяйте типов аргументов. Можно определить несколько видов ввода, аргументы которых могут иметь значения по умолчанию. Например, функция в следующем примере использует `divisor=2`, если `divisor` не задан.

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

Также можно задавать вводные данные в явном виде при вызове функции. Если задавать параметры явно, то порядок не важен:

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

В Python блоки кода не отделяются фигурными скобками или другими обозначениями, как в других языках. Вместо этого используются отступы. Например, следующее выдаст синтаксическую ошибку, так как перед `return` нет такого же количества отступов, как в предыдущих строках.

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

Если добавить отступы, функция будет работать:

```
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]
```

Точки с запятой не нужны на концах строк. Можно расположить несколько утверждений на одной строке, отделенных точками с запятой:

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

Если требуется расположить строку кода на нескольких строках, используйте \:

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

В Sage счетчики производят итерации по интервалу целых чисел. Например, первая строчка в примере означает то же самое, что `for(i=0; i<3; i++)` в C++ или Java:

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

Первая строчка в следующем примере эквивалентна `for(i=2;i<5;i++)`.

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

Третий аргумент задает шаг. Следующее эквивалентно `for(i=1;i<6;i+=2)`.

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

Часто требуется создать таблицу для вывода чисел, посчитанных в Sage. Легкий способ — использовать форматирование строк. Ниже создается таблица с тремя столбцами шириной 6, содержащая таблицу квадратов и кубов:

```
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
```

Самым базовым типом данных в Sage является список — набор различных объектов. Например, команда `range` создаст список:

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

Далее показан пример более сложного списка:

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

Индексы в списке начинаются с нуля, как во многих языках программирования.

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

Используйте `len(v)` для того, чтобы получить длину `v`; `v.append(obj)` для того, чтобы добавить новый объект к концу `v`, и `del v[i]`, чтобы удалить i-й элемент из `v`:

```
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]
```

Другой очень важный тип данных — словарь (или ассоциативный массив). Он работает, как список, но может быть индексирован почти любым объектом (индексы должны быть неизменимыми):

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

Также можно определить новый тип данных с использованием классов. Инкапсулирование математических объектов в классах — это мощная техника, которая может помочь упростить и организовать программы в Sage. Ниже показан пример класса, который состоит из списка положительных чётных целых чисел до n; он получен из встроенного типа `list`.

```
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."
```

Метод `__init__` вызывается для инициализации объекта при его создании; метод `__repr__` выведет все объекты. Конструктор списка вызывается во второй строчке метода `__init__`. Объект класса `Evens` создается в следующем виде:

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

Заметьте, что `e` выводится с помощью метода `__repr__`, который был задан нами. Для просмотра списка чисел используйте функцию `list`:

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

Можно обратиться к атрибуту `n` или использовать `e` как список.

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

2.4 Базовая алгебра и вычисления

Sage может осуществлять вычисления такие, как поиск решений уравнений, дифференцирование, интегрирование и преобразования Лапласа. См. Sage Constructions , где содержатся примеры.

2.4.1 Решение уравнений

Точное решение уравнений

Функция `solve` решает уравнения. Для ее использования сначала нужно определить некоторые переменные; аргументами для `solve` будут уравнение (или система уравнений) и переменные, для которых нужно найти решение:

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

Можно решать уравнения для одной переменной через другие:

```
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)]
```

Также можно решать уравнения с несколькими переменными:

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

Следующий пример показывает, как Sage решает систему нелинейных уравнений. Для начала система решается символьно:

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

Для приближенных значений решения можно использовать:

```
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]]
```

(Функция `n` выведет приближенное значение. Аргументом для данной функции является количество битов точности)

Численное решение уравнений

Во многих случаях функция `solve` не способна найти точное решение уравнения. Вместо нее можно использовать функцию `find_root` для нахождения численного решения. Например, `solve` не возвращает ничего существенного для следующего уравнения:

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

С другой стороны функция `find_root` может использоваться для решения вышеуказанного примера в интервале $0 < \phi < \pi/2$:

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

2.4.2 Дифференцирование, интегрирование и т.д.

Sage умеет дифференцировать и интегрировать многие функции. Например, для того, чтобы продифференцировать $\sin(u)$ по переменной u, требуется:

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

Для подсчета четвертой производной функции $\sin(x^2)$ надо:

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

Для нахождения частных производных, как, например, для функции x^2+17y^2 по x и y соответственно:

```
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
```

Теперь найдём интегралы: и определенные, и неопределенные. Например, $\int x\sin(x^2)\,dx$ и $\int_0^1 \frac{x}{x^2+1}\,dx$

```
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)
```

Для нахождения разложения на простые дроби для $\frac{1}{x^2-1}$ нужно сделать следующее:

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

2.4.3 Решение дифференциальных уравнений

Sage может использоваться для решения дифференциальных уравнений. Для решения уравнения $x' + x - 1 = 0$ сделаем следующее:

```
sage: t = var('t')    # определение переменной t для символьных вычислений
sage: x = function('x',t)   # определение функции x зависящей от t
sage: DE = diff(x, t) + x - 1
sage: desolve(DE, [x,t])
(_C + e^t)*e^(-t)
```

Для этого используется интерфейс Maxima [Max], поэтому результат может быть выведен в виде, отличном от обычного вывода Sage. В данном случае общее решение для данного дифференциального уравнения - $x(t) = e^{-t}(e^t + C)$.

Преобразования Лапласа также могут быть вычислены. Преобразование Лапласа для $t^2e^t - \sin(t)$ вычисляется следующим образом:

```
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
```

Приведем более сложный пример. Отклонение от положения равновесия для пары пружин, прикрепленных к стене слева,

```
|------\/\/\/\/\---|масса1|----\/\/\/\/\/----|масса2|
        пружина1                 пружина2
```

может быть представлено в виде дифференциальных уравнений второго порядка

$$m_1x_1'' + (k_1 + k_2)x_1 - k_2x_2 = 0$$
$$m_2x_2'' + k_2(x_2 - x_1) = 0,$$

где m_i - это масса объекта i, x_i - это отклонение от положения равновесия массы i, а k_i - это константа для пружины i.

Пример: Используйте Sage для вышеуказанного примера с $m_1 = 2$, $m_2 = 1$, $k_1 = 4$, $k_2 = 2$, $x_1(0) = 3$, $x_1'(0) = 0$, $x_2(0) = 3$, $x_2'(0) = 0$.

Решение: Надо найти преобразование Лапласа первого уравнения (с условием $x = x_1$, $y = x_2$):

```
sage: de1 = maxima("2*diff(x(t),t, 2) + 6*x(t) - 2*y(t)")
sage: lde1 = de1.laplace("t","s"); lde1
2*(-%at('diff(x(t),t,1),t=0)+s^2*'laplace(x(t),t,s)-x(0)*s)-2*'laplace(y(t),t,s)+6*'laplace(x(t),t,s)
```

Данный результат тяжело читаем, однако должен быть понят как

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

Найдем преобразование Лапласа для второго уравнения:

```
sage: de2 = maxima("diff(y(t),t, 2) + 2*y(t) - 2*x(t)")
sage: lde2 = de2.laplace("t","s"); lde2
-%at('diff(y(t),t,1),t=0)+s^2*'laplace(y(t),t,s)+2*'laplace(y(t),t,s)-2*'laplace(x(t),t,s)-y(0)*s
```

Результат:

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

Вставим начальные условия для $x(0)$, $x'(0)$, $y(0)$ и $y'(0)$, и решим уравения:

```
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)]]
```

Теперь произведём обратное преобразование Лапласа для нахождения ответа:

```
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)
```

Итак, ответ:

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

График для ответа может быть построен параметрически, используя

```
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)
```

Графики могут быть построены и для отдельных компонентов:

```
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)
```

Для более исчерпывающей информации по графикам см. *Построение графиков*. Также см. секцию 5.5 из [NagleEtAl2004] для углубленной информации по дифференциальным уравнениям.

2.4.4 Метод Эйлера для решения систем дифференциальных уравнений

В следующем примере показан метод Эйлера для дифференциальных уравнений первого и второго порядков. Сначала вспомним, что делается для уравнений первого порядка. Дана задача с начальными условиями в виде

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

требуется найти приблизительное значение решения при $x = b$ и $b > a$.

Из определения производной следует, что

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

где $h > 0$ дано и является небольшим. Это и дифференциальное уравнение дают $f(x, y(x)) \approx \frac{y(x+h)-y(x)}{h}$. Теперь надо решить для $y(x + h)$:

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

Если назвать $h \cdot f(x, y(x))$ "поправочным элементом", $y(x)$ "прежним значением y" а $y(x + h)$ "новым значением y", тогда данное приближение может быть выражено в виде

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

Если разбить интервал между a и b на n частей, чтобы $h = \frac{b-a}{n}$, тогда можно записать информацию для данного метода в таблицу.

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

Целью является заполнить все пустоты в таблице по одному ряду за раз до момента достижения записи ???, которая и является приближенным значением метода Эйлера для $y(b)$.

Решение систем дифференциальных уравнений похоже на решение обычных дифференциальных уравнений.

Пример: Найдите численное приблизительное значение для $z(t)$ при $t = 1$, используя 4 шага метода Эйлера, где $z'' + tz' + z = 0$, $z(0) = 1$, $z'(0) = 0$.

Требуется привести дифференциальное уравнение 2го порядка к системе двух дифференцальных уравнений первого порядка (используя $x = z$, $y = z'$) и применить метод Эйлера:

```
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
```

Итак, $z(1) \approx 0.75$.

Можно построить график для точек (x, y), чтобы получить приблизительный вид кривой. Функция `eulers_method_2x2_plot` выполнит данную задачу; для этого надо определить функции f и g, аргумент которых имеет три координаты: (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)
```

В этот момент `P` содержит в себе два графика: `P[0]` - график x по t и `P[1]` - график y по t. Оба эти графика могут быть выведены следующим образом:

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

2.4.5 Специальные функции

Несколько ортогональных полиномов и специальных функций осуществлены с помощью PARI [GAP] и Maxima [Max].

```
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
```

На данный момент Sage рассматривает данные функции только для численного применения. Для символьного использования нужно напрямую использовать интерфейс Maxima, как описано ниже:

```
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'
```

2.5 Построение графиков

Sage может строить двумерные и трехмерные графики.

2.5.1 Двумерные графики

В двумерном пространстве Sage может отрисовывать круги, линии и многоугольники; графики функций в декартовых координатах; также графики в полярных координатах, контурные графики и изображения векторных полей. Некоторые примеры будут показаны ниже. Для более исчерпывающей информации по построению графиков см. *Решение дифференциальных уравнений* и *Maxima*, а также документацию Sage Constructions.

Данная команда построит желтую окружность радиуса 1 с центром в начале:

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

Также можно построить круг:

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

Можно создавать окружность и задавать ее какой-либо переменной. Данный пример не будет строить окружность:

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

Чтобы построить ее, используйте `c.show()` или `show(c)`:

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

`c.save('filename.png')` сохранит график в файл.

Теперь эти 'окружности' больше похожи на эллипсы, так как оси имеют разный масштаб. Это можно исправить следующим образом:

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

Команда `show(c, aspect_ratio=1)` выполнит то же самое. Сохранить картинку можно с помощью `c.save('filename.png', aspect_ratio=1)`.

Строить графики базовых функций легко:

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

Как только имя переменной определено, можно создать параметрический график:

```
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
```

Важно отметить, что оси графика будут пересекаться лишь в том случае, когда начало координат находится в поле зрения графика, и что к достаточно большим значениям можно применить научное обозначение:

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

Можно объединять построения, добавляя их друг другу:

```
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)
```

Хороший способ создания заполненных фигур — создание списка точек (`L` в следующем примере), а затем использование команды `polygon` для построения фигуры с границами, образованными заданными точками. К примеру, создадим зеленый дельтоид:

```
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
```

Напечатайте `show(p, axes=false)`, чтобы не показывать осей на графике.

Можно добавить текст на график:

```
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)
```

Учителя математики часто рисуют следующий график на доске: не одну ветвь arcsin, а несколько, т.е. график функции $y = \sin(x)$ для x между -2π и 2π, перевернутый по отношению к линии в 45 градусов. Следующая команда Sage построит вышеуказанное:

```
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
```

Так как функция тангенса имеет больший интервал, чем синус, при использовании той же техники для перевертывания тангенса требуется изменить минимальное и максимальное значения координат для оси x:

```
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 также может строить графики в полярных координатах, контурные построения и изображения векторных полей (для специальных видов функций). Далее следует пример контурного чертежа:

```
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 Трехмерные графики

Sage также может быть использован для создания трехмерных графиков. Эти графики строятся с помощью пакета [Jmol], который поддерживает поворот и приближение картинки с помощью мыши.

Используйте `plot3d`, чтобы построить график функции формы $f(x, y) = z$:

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

Еще можно использовать `parametric_plot3d` для построения графиков параметрических поверхностей, где каждый из x, y, z определяется функцией одной или двух переменных (параметры; обычно u и v). Предыдущий график может быть выражен параметрически в следующем виде:

```
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
```

Третий способ построить трехмерную поверхность в Sage - использование `implicit_plot3d`, который строит контуры графиков функций, как $f(x, y, z) = 0$. Чтобы построить сферу, возпользуемся классической формулой:

```
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
```

Ниже показаны несколько примеров:

Скрещенный колпак (близкий родственник широко известного листа Мёбиуса):

```
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
```

Крученый тороид:

```
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
```

Лемниската:

```
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 Распространённые проблемы с функциями

Некоторые аспекты определения функций (например, для дифференцирования или построения графика) могут быть не ясны. В этом разделе мы обращаем внимание на некоторые наиболее распространенные проблемы.

Далее показаны несколько способов определения того, что можно назвать "функцией":

1. Определите функцию Python, как описано в разделе *Функции, отступы и счетчики*. Для таких функций можно построить графики, но продифференцировать или проинтегрировать их нельзя.

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

Обратите внимание на синтаксис в последней строчке. `plot(f(z), 0, 2)` выдаст ошибку, так как `z` - это переменная-болванка в определении `f`, которая не определена внутри данной конструкции. Просто `f(z)` возвратит ошибку. Следующее будет работать в данном контексте, однако, в общем, возникнут некоторые затруднения, но они могут быть проигнорированы (см. пункт 4).

```
sage: var('z')   # определение переменной z для символьных вычислений
z
sage: f(z)
z^2
sage: plot(f(z), 0, 2)
Graphics object consisting of 1 graphics primitive
```

В этом случае `f(z)` - это символьное выражение.

2. Определим "вызываемое символьное выражение". Оно может быть продифференцировано, проинтегрировано, а также можно построить его график.

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

Если g — это вызываемое символьное выражение, g(x) — это связянный с ним объект, но другого вида, для которого можно построить график и который можно дифференциировать и т.д.

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

3. Можно использовать уже определенную функцию Sage — 'функцию исчисления'. Для нее может быть построен график, она может быть продифференцирована и проинтегрирована.

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

Сама по себе функция sin не может быть продифференцирована, по крайней мере, не может произвести cos.

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

Использование f = sin(x) вместо sin работает, но лучше использовать f(x) = sin(x) для того, чтобы определить вызываемое символьное выражение.

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

Далее следуют некоторые общие проблемы с объяснением:

4. Случайная оценка.

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

Проблема: plot(h(x), 0, 4) построит кривую $y = x - 2$. Причина: В команде plot(h(x), 0, 4) сначала оценивается h(x), что означает подставку x в функцию h и оценку x<2.

```
sage: type(x<2)
<type 'sage.symbolic.expression.Expression'>
```

Решение: Не используйте plot(h(x), 0, 4); используйте:

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

5. Ошибочное создание константы вместо функции.

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

Проблема: `g(3)`, например, возвратит ошибку с сообщением "ValueError: the number of arguments must be less than or equal to 0."

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

g не является функцией, это константа, поэтому она не имеет переменных, и вы можете вставлять что угодно в нее.

Решение: есть несколько возможных путей.

- Определить `f` изначально как символьное выражение.

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

- Либо вместе с `f`, определенной выше, определить `g` как символьное выражение.

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

- Либо с `f` и `g`, заданными, как показано выше, создать переменную, под которую подставляются значения.

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

Есть еще один способ, как определить различие между производными `f = x` и `f(x) = x`

```
sage: f(x) = x
sage: g = f.derivative()
sage: g.variables()  # переменные, которые присутствуют в g
()
sage: g.arguments()  # аргументы, которые могут быть подставлены в g
(x,)
sage: f = x
sage: h = f.derivative()
sage: h.variables()
()
sage: h.arguments()
()
```

Как показывает данный пример, `h` не принимает аргументов, поэтому `h(3)` вернет ошибку.

2.7 Основные кольца

При объявлении матриц, векторов или полиномов для них иногда полезно, а иногда и необходимо определять "кольца", на которых они определены. *Кольцо* - это математическая конструкция, в которой существуют определенные понятия суммы и произведения. Если вы никогда о них не слышали, то вам, вероятно, достаточно знать об этих четырех часто используемых кольцах:

- целые числа $\{..., -1, 0, 1, 2, ...\}$, называемые `ZZ` в Sage.
- рациональные числа – например, дроби или отношения целых чисел —, называемые `QQ` в Sage.
- вещественные числа, называемые `RR` в Sage.
- комплексные числа, называемые `CC` в Sage.

Знание различий между данными кольцами очень важно, так как один и тот же полином, определенный в разных кольцах, может вести себя по-разному. Например, полином $x^2 - 2$ имеет два корня: $\pm\sqrt{2}$. Эти корни не являются рациональными числами, поэтому если вы работаете с полиномами с рациональными коэффициентами, то полином не будет разлагаться на множители. С вещественными коэффициентами — будет. Поэтому стоит определить кольцо, чтобы быть уверенным, что полученный результат будет правильным. Следующие две команды задают множества полиномов с рациональными коэффициентами и вещественными коэффициентами соответственно. Множества названы "ratpoly" и "realpoly", но это не столь важно в данном контексте, однако символьные сочетания ".<t>" и ".<z>" являются названиями переменных, использованных в двух случаях.

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

Факторизируем $x^2 - 2$:

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

Символ `I` обозначает квадратный корень из -1; `i` — это то же самое, что `I`. Конечно, это не рациональное число:

```
sage: i  # квадратный корень из -1
I
sage: i in QQ
False
```

Заметка: Вышеописанный код может работать не так, как задумывалось, если переменной `i` было задано другое значение, например, если оно было использовано, как счетчик для цикла. В таком случае введите

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

для того, чтобы получить изначальное комплексное значение `i`.

Есть одна тонкость в задании комплексных чисел: как описано выше, символ `i` представляет квадратный корень из -1, но это *формальный* или *символичный* квадратный корень из -1. Вызов `CC(i)` или `CC.0` вернет *комплексный* квадратный корень из -1.

```
sage: i = CC(i)       # комплексное число с плавающей запятой
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()        # мнимая часть
0.666666666666667
sage: z.real() == a   # автоматическое приведение типов перед сравнением
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       # автоматическое приведение типов перед сложением
0.766666666666667
sage: 0.1 + 2/3       # приведение типов в Sage симметрично
0.766666666666667
```

Далее следуют примеры базовых колец в Sage. Как отмечено выше, кольцо рациональных чисел обозначается как `QQ`, а также как `RationalField()` (*поле* - это кольцо, в котором произведение является коммутативным и в котором каждый ненулевой элемент имеет обратную величину в этом кольце (рациональные числа являются полем, а целые - нет):

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

Десятичное число `1.2` рассматривается как `QQ`: десятичные числа, которые также являются рациональными, могут быть "приведены" к рациональным числам. Числа π и $\sqrt{2}$ не являются рациональными:

```
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
```

Для использования в высшей математике Sage также может выполнять операции с другими кольцами, как конечные поля, p-адические числа, кольцо алгебраических чисел, полиномиальные кольца и матричные кольца. Далее показаны некоторые из них:

```
sage: GF(3)
Finite Field of size 3
sage: GF(27, 'a')  # если поле не простое, нужно задать имя генератора
Finite Field in a of size 3^3
sage: Zp(5)
5-adic Ring with capped relative precision 20
sage: sqrt(3) in QQbar # алгебраическое замыкание QQ
True
```

2.8 Линейная алгебра

Sage поддерживает стандартные конструкции из линейной алгебры, как характеристические полиномы, ступенчатые формы, суммы элементов главной диагонали матрицы, разложения.

Создавать и перемножать матрицы легко:

```
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]
```

Решение матричных уравнений также выполняется без затруднений, используя метод `solve_right`. Вычисление `A.solve_right(Y)` возвратит матрицу (или вектор) X такой, что $AX = Y$:

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

`\` может быть использован вместо `solve_right`; используйте `A \ Y` вместо `A.solve_right(Y)`.

```
sage: A \ Y
(-2, 1, 0)
```

Если решения не существует, то Sage вернет ошибку:

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

Используйте `A.solve_left(Y)`, чтобы найти X в $XA = Y$. Sage может находить собственное число и собственный вектор:

```
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)]
```

(Результат `eigenvectors_left` - это список троек: (собственное число, собственный вектор, многообразие).) Собственные числа и вектора для `QQ` или `RR` также могут быть вычислены с помощью Maxima (см. *Maxima*).

Как указано в разделе *Основные кольца*, кольцо, в котором определена матрица, влияет на некоторые ее свойства. В следующем примере первый аргумент команды `matrix` сообщает Sage, чтобы матрица рассматривалась как матрица целых чисел (случай с `ZZ`), как матрица рациональных чисел (`QQ`) или как матрица вещественных чисел (`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]
sage: AQ.echelon_form()
[1 0]
[0 1]
sage: AR.echelon_form()
[ 1.00000000000000 0.000000000000000]
[0.000000000000000  1.00000000000000]
```

Для вычисления собственных значений и собственных векторов матриц действительных или комплексных чисел с плавающей точкой, матрица должна быть определена над `RDF` (Real Double Field) или `CDF` (Complex Double Field), соответственно. Если кольцо не указано, и в матрице используются действительные или комплексные константы с плавающей точкой, то (по умолчанию) она определена над не всегда поддерживающими такие вычисления полями `RR` или `CC`, соответственно:

```
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 Матричное пространство

Создадим пространство $\mathrm{Mat}_{3\times 3}(\mathbf{Q})$, состоящее из матриц 3×3 с элементами из рациональных чисел:

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

(Для того, чтобы создать пространство из матриц 3 на 4, используйте `MatrixSpace(QQ,3,4)`. Если число столбцов не указано, по умолчанию оно будет равно числу строк (`MatrixSpace(QQ,3)` эквивалентно `MatrixSpace(QQ,3,3)`.) Матричное пространство имеет базис, который содержится в Sage в виде списка:

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

Создадим матрицу как элемент `M`.

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

Далее покажем вычисление матриц, определенных в конечных полях:

```
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)]
```

Создадим подпространство в $\mathbf{F}_2$, охватывающее вышеперечисленные строки.

```
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]
```

2.8.2 Разреженная линейная алгебра

Sage поддерживает разреженную линейную алгебру.

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

Мультимодульный алгоритм в Sage работает хорошо для квадратных матриц (но не так хорошо для неквадратных матриц):

```
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()
```

Заметьте, что в Python использование заглавных букв играет роль:

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

2.9 Полиномы

Данный раздел содержит информацию о том, как создавать и использовать полиномы в Sage.

2.9.1 Полиномы одной переменной

Есть три способа создания полиномиальных колец.

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

Данный способ создаст полиномиальное кольцо и укажет Sage использовать строку 't' в качестве неизвестного при выводе на экран. Однако, это не определяет символ `t` для использования в Sage, так что нельзя при помощи него ввести полином (как $t^2 + 1$), принадлежащий `R`.

Другой способ:

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

Этот способ имеет ту же проблему по отношению к `t`.

Третий способ более удобный

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

или

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

или даже

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

Этот способ влечет за собой объявление переменной `t` как неизвестного в полиномиальном кольце так, что ее можно использовать при создании элементов `R`, как описано ниже. (Заметьте, что третий способ похож на обозначение конструктора в Magma, и, как в Magma, он может быть использован для широкого набора объектов.)

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

Какой бы способ ни использовался для задания полиномиального кольца, можно вычленить неизвестное в виде 0^{th} генератора:

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

Похожая конструкция используется для комплексных чисел: комплексные числа могут быть рассмотрены как генерированные из вещественных чисел с использованием символа `i`; из этого следует:

```
sage: CC
Complex Field with 53 bits of precision
sage: CC.0  # 0-ой генератор CC
1.00000000000000*I
```

Для полиномиальных колец можно получить и кольцо, и его генератор, или просто генератор во время создания кольца:

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

Наконец, можно совершить некоторые арифметические операции в $\mathbf{Q}[t]$.

```
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)]
```

Деление двух полиномов создаст элемент в дробном поле, что будет сделано Sage автоматически.

```
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
```

Используя ряды Лорана, можно посчитать разложение в ряд в дробном поле `QQ[x]`:

```
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)
```

Если назвать переменную по-другому, можно получить другое одномерное полиномиальное кольцо.

```
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
```

Кольцо определяется переменной. Обратите внимание, что создание ещё одного кольца с переменной `x` не вернет другого кольца.

```
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 поддерживает кольца степенных рядов и рядов Лорана для любого базисного кольца. В следующем примере создадим элемент из $\mathbf{F}_7[[T]]$ и поделим, чтобы создать элемент из $\mathbf{F}_7((T))$.

```
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
```

Также можно создавать кольца степенных рядов, используя двойные скобки:

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

2.9.2 Полиномы нескольких переменных

Для работы с полиномами с несколькими переменными, сначала надо объявить полиномиальное кольцо и переменные.

```
sage: R = PolynomialRing(GF(5),3,"z") # здесь 3 - это число переменных
sage: R
Multivariate Polynomial Ring in z0, z1, z2 over Finite Field of size 5
```

Так же, как и для одномерных полиномов, существует несколько путей:

```
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
```

Чтобы имена переменных состояли из букв, надо использовать следующее:

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

Немного арифметики:

```
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
```

Можно использовать более математическое обозначение, чтобы построить полиномиальное кольцо.

```
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,))
```

Многомерные полиномы внедрены в Sage с использованием словарей Python. Sage использует Singular [Si] для вычислений НОД и базиса Грёбнера идеалов.

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

```
sage: g = x^2*y^2
sage: f.gcd(g)
x^2
```

Создадим идеал (f, g), генерированный из f и f умножением `(f,g)` на `R`.

```
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
```

Кстати, базис Грёбнера является не списком, а неизменяемой последовательностью. Это означает, что у него есть универсум, родитель и что он не может быть изменен (что хорошо, поскольку изменение базиса нарушило бы другие операции, использующие базис Грёбнера).

```
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.
```

Некоторая коммутативная алгебра доступна в Sage и внедрена с помощью Singular. К примеру, можно посчитать примарное разложение и простые соответствующие для I:

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

Sage поддерживает вычисления с группами перестановок, конечными классическими группами (как, например, $SU(n, q)$), конечными матричными группами (с собственными генераторами) и группами Абеля (даже с бесконечными). Многое из этого осуществляется посредством интерфейса к GAP.

Например, чтобы построить группу перестановок, надо задать список генераторов:

```
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()
120
sage: G.is_abelian()
False
sage: G.derived_series()           # random 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 of (Permutation Group with generators [(3,4), (1,2,3)(4,5)]) generated by [()]
```

```
sage: G.random_element()           # random output
(1,5,3)(2,4)
sage: print latex(G)
\langle (3,4), (1,2,3)(4,5) \rangle
```

Также можно получить символьную таблицу в формате LaTeX:

```
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)
```

Sage также включает в себя классические и матричные группы для конечных полей:

```
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_class_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
```

Также можно производить вычисления с группами Абеля (конечными и бесконечными):

```
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
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
```

2.11 Теория чисел

Sage имеет обширную функциональность в плане теории чисел. Например, можно производить арифметические операции в $\mathbf{Z}/N\mathbf{Z}$:

```
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 содержит стандартные функции теории чисел. Например,

```
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
```

Отлично!

Функция `sigma(n,k)` в Sage суммирует k-е степени делителей `n`:

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

Далее покажем алгоритм Эвклида, ϕ-функцию Эйлера и китайскую теорему об остатках:

```
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
```

Уясним кое-что для $3n + 1$.

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

Китайская теорема об остатках:

```
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]]
```

2.11.1 p–адические числа

Поле p-адических чисел реализовано в Sage. Обратите внимание: как только поле p-адических чисел создано, его точность не может быть изменена.

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

Большое количество методов встроено для класса NumberField.

```
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(type="pari")
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field
in a with defining polynomial 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
```

2.12 Немного высшей математики

2.12.1 Алгебраическая геометрия

Sage позволяет создавать любые алгебраические многообразия, но иногда функциональность ограничивается кольцами $\mathbf{Q}$ или конечными полями. Например, найдем объединение двух плоских кривых, а затем вычленим кривые как несократимые составляющие объединения.

```
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 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
]
```

Также можно найти все точки пересечения двух кривых.

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

Таким образом точки $(1, 0)$ и $(0, 1)$ находятся на обеих кривых, а координаты по оси y удовлетворяют функции $2y^2 + 4y + 3 = 0$.

Sage может вычислить тороидальный идеал неплоской кривой третьего порядка:

```
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],
 [-c^2 + b*d, b^2 - a*c, -b*c + a*d],
 [-c^2 + b*d, b*c - a*d, b^2 - a*c, -c^3 + a*d^2],
 [c^3 - a*d^2, -c^2 + b*d, b*c - a*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^2 + a*c, -b^3 + a^2*d],
 [c^2 - b*d, b*c - a*d, b^3 - a^2*d, -b^2 + a*c],
 [c^2 - b*d, b*c - a*d, b^2 - a*c]]
sage: F.polyhedralfan()
Polyhedral fan in 4 dimensions of dimension 4
```

2.12.2 Эллиптические кривые

Функциональность эллиптических кривых включает в себя большую часть функциональности PARI, доступ к информации в онлайн таблицах Cremona (что требует дополнительный пакет баз данных), функциональность mwrank, алгоритм SEA, вычисление всех изогений, много нового кода для **Q** и некоторую функциональность программного обеспечения Denis Simon.

Команда `EllipticCurve` для создания эллиптических кривых имеет много форм:

- EllipticCurve([a_1, a_2, a_3, a_4, a_6]): Возвратит эллиптическую кривую

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

- EllipticCurve([a_4, a_6]): То же, что и выше, но $a_1 = a_2 = a_3 = 0$.
- EllipticCurve(label): Вернет эллиптическую кривую из базы данных Cremona с заданным ярлыком Cremona. Ярлык - это строка, например, `"11a"` or `"37b2"`.
- EllipticCurve(j): Вернет эллиптическую кривую с инвариантой j.
- EllipticCurve(R, [a_1, a_2, a_3, a_4, a_6]): Создаст эллиптическую кривую из кольца R с заданными a_i ‘ми, как было указано выше.

Использование каждого конструктора:

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

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
```

Пара $(0, 0)$ - это точка на эллиптической кривой E, заданной функцией $y^2 + y = x^3 - x$. Для создания этой точки в Sage напечатайте `E([0,0])`. Sage может добавить точки на такую эллиптическую кривую:

```
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
```

Эллиптические кривые для комплексных чисел задаются параметрами инварианты j. Sage вычислит инварианту j:

```
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
```

Если мы создадим кривую с той же инвариантой j, как для E, она не должна быть изоморфной E. В следующем примере кривые не изоморфны, так как их кондукторы различны.

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

Однако кручение F на 2 даст изоморфную кривую.

```
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
```

Можно посчитать коэффициенты a_n ряда L или модулярной формы $\sum_{n=0}^{\infty} a_n q^n$, прикрепленной к эллиптической кривой. Данное вычисление использует библиотеку PARI:

```
sage: E = EllipticCurve([0,0,1,-1,0])
sage: print 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)
```

Займет лишь секунду для подсчета всех a_n для $n \leq 10^5$:

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

Эллиптические кривые могут быть построены с помощью их ярлыков Cremona.

```
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
```

Также есть доступ к базе данных Cremona.

```
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]}
```

Объекты, возвращенные из базы данных, не принадлежат типу `EllipticCurve`. Это элементы базы данных, имеющие пару полей. Существует малая версия базы данных Cremona, которая есть по умолчанию в Sage и содержит ограниченную информацию о эллиптических кривых с кондуктором ≤ 10000. Также существует дополнительная большая версия, которая содержит исчерпывающую информацию о всех кривых с кондуктором до 120000 (Октябрь 2005). Еще один дополнительный пакет (2GB) для Sage содержит сотни миллионов эллиптических кривых в базе данных Stein-Watkins.

2.12.3 Символы Дирихле

Символ Дирихле - это расширение гомоморфизма $(\mathbf{Z}/N\mathbf{Z})^* \to R^*$ для кольца R к $\mathbf{Z} \to R$.

```
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()
```

```
(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
```

Создав группу, нужно создать элемент и с его помощью посчитать.

```
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
```

Также возможно посчитать действие группы Галуа $\mathrm{Gal}(\mathbf{Q}(\zeta_N)/\mathbf{Q})$ на эти символы.

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

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 of modulus 3 over Cyclotomic Field of order
6 and degree 2,
Group of Dirichlet characters of modulus 7 over Cyclotomic Field of order
6 and degree 2
]
```

Далее надо построить группу символов Дирихле по модулю 20, но со значениями с $\mathbf{Q}(i)$:

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

Теперь посчитаем несколько инвариант `G`:

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

В данном примере, символы Дирихле создаются со значениями в числовом поле, явно задается выбор корня объединения третьим аргументом `DirichletGroup`.

```
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 of modulus 5 over 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]
```

Здесь `NumberField(x^4 + 1, 'a')` говорит Sage использовать символ "a" для печати того, чем является `K` (числовое поле с определяющим полиномом $x^4 + 1$). Название "a" не объявлено на данный момент. Когда `a = K.0` (эквивалентно `a = K.gen()`) будет оценено, символ "a" представляет корень полинома $x^4 + 1$.

2.12.4 Модулярные формы

Sage может выполнять вычисления, связанные с модулярными формами, включая измерения, вычисление модулярных символов, операторов Гекке и разложения.

Существует несколько доступных функций для вычисления измерений пространств модулярных форм. Например,

```
sage: dimension_cusp_forms(Gamma0(11),2)
1
sage: dimension_cusp_forms(Gamma0(1),12)
1
sage: dimension_cusp_forms(Gamma1(389),2)
6112
```

Далее показаны вычисления операторов Гекке в пространстве модулярных символов уровня 1 и веса 12.

```
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
```

Также можно создавать пространство для $\Gamma_0(N)$ и $\Gamma_1(N)$.

```
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 and over Rational Field
```

Вычислим некоторые характеристические полиномы и q-разложения.

```
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)
]
```

Также возможны вычисления пространств модулярных символов с буквами.

```
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)
]
```

Пример того, как Sage может вычислять действия операторов Гекке на пространство модулярных форм.

```
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
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
```

Допустим, T_p — это обычный оператор Гекке (p простое). Как операторы Гекке T_2, T_3, T_5 ведут себя в пространстве модулярных символов?

```
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]
```

Глава 3

Интерактивная оболочка

Почти всегда в этом руководстве мы предполагаем, что интерпретатор Sage был запущен командой `sage`. Она запустит специальную версию консоли IPython и импортирует множество функций и классов, так что они готовы для использования в командной строке. Более тонкая настройка производится редактированием файла `$SAGE_ROOT/ipythonrc`. При запуске Sage вы увидите вывод, похожий на следующий:

```
----------------------------------------------------------------------
| SAGE Version 3.1.1, Release Date: 2008-05-24                       |
| Type notebook() for the GUI, and license() for information.        |
----------------------------------------------------------------------


sage:
```

Чтобы выйти из Sage, нажмите Ctrl-D или введите `quit` или `exit`.

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

Wall time — это прошедшее время. Это значение верно, потому как в "CPU time" не входит время, использованное субпроцессами вроде GAP или Singular.

(Постарайтесь не убивать процесс Sage командой `kill -9` из терминала, потому что Sage может не убить дочерние процессы, такие как Maple, или может не очистить временные файлы из директории `$HOME/.sage/tmp`.)

3.1 Ваша сессия Sage

Сессия — это последовательность вводов и выводов начиная с запуска программы и заканчивая выходом из нее. Sage заносит всю историю вводов в log-файл, используя IPython. Если вы используете интерактивную оболочку (не веб-интерфейс Notebook), то вы можете ввести `%hist`, чтобы вывести список всех введенных команд. Вы можете ввести `?` в командной строке Sage, чтобы получить больше информации о IPython, например, "IPython предоставляет пронумерованные командные строки... с кешированием ввода и вывода. Все введенные данные сохраняются и могут быть использованы как переменные (помимо обычного вызова с помощью стрелок). Следующие глобальные переменные присутствуют всегда (не перезаписывайте их!)":

```
_:  previous input (interactive shell and notebook)
__: next previous input (interactive shell only)
_oh : list of all inputs (interactive shell only)
```

Пример:

```
sage: factor(100)
 _1 = 2^2 * 5^2
sage: kronecker_symbol(3,5)
 _2 = -1
sage: %hist   # Работает только в интерактивной оболочке, но не в Sage 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
```

Мы не включаем номера строк в этом учебном пособии и в другой документации Sage.

Вы также можете хранить список введенных команд сессии в виде макроса для сессии.

```
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
```

При использовании интерактивной оболочки Sage, любая UNIX-команда может быть запущена с помощью префикса `!`. Например

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

возвращает содержание текущей директории.

Переменная `PATH` сожержит директорию bin (бинарные файлы) в самом начале, так что если вы запускаете `gp`, `gap`, `singular`, `maxima`, и т.д. вы получаете версии, включенные в Sage.

```
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 Журналирование ввода и вывода

Журналирование сессии Sage это не то же самое, что сохранение сессии (см. *Сохранение и загрузка полных сессий* для этого). Для журналирования ввода (и, опционально, вывода), используйте команду `logstart`. Введите `logstart?` для подробностей. Вы можете использовать эту команду для журналирования всего, что вы вводите, всего вывода, и даже можете воспроизвести введенные данные в будущей сессии (просто загрузив log-файл).

```
was@form:~$ sage
--------------------------------------------------------------------------
| SAGE Version 3.0.2, Release Date: 2008-05-24                           |
| Type notebook() for the GUI, and license() for information.            |
--------------------------------------------------------------------------

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
--------------------------------------------------------------------------
| SAGE Version 3.0.2, Release Date: 2008-05-24                           |
| Type notebook() for the GUI, and license() for information.            |
--------------------------------------------------------------------------

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)]
```

Если вы используете Sage в `konsole` — терминале среды KDE в GNU/Linux — тогда вы можете сохранить сессию следующим образом: после запуска Sage в `konsole`, выберите “settings”, потом “history...”, потом “set unlimited”. Когда вы готовы сохранить сессию, выберите “edit” и “save history as...” и введите имя файла для сохранения. После этого вы можете воспользоваться любым текстовым редактором, например xemacs, для чтения файла.

3.3 Вставка игнорирует приглашение

Допустим, вы читаете сессию Sage или вычисления Python, и хотите скопировать их в Sage. Но есть одна проблема: знаки `>>>` или `sage:`. На самом деле вы можете копировать и вставлять примеры, которые включают эти знаки. Дргуими словами, Sage игнорирует символы `>>>` или `sage:` перед отправкой команд в Python. Например,

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

3.4 Команды измерения времени

Если вы введете команду `%time` в начале строки ввода, то время, затраченное на выполнение операции, будет выведено на экран. Например, вы можете измерить время выполнения операции возведения в степень несколькими путями. Показания ниже будут отличаться от ваших; они могут отличаться даже в разных версиях Sage. Чистый 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
```

Это означает что 0.66 секунд было затрачено в сумме, а “Wall time”, (прошедшее время), тоже 0.66 секунд. Если ваш компьютер сильно загружен другими процессами, то “Wall time” может сильно отличаться от процессорного времени.

Далее мы посчитаем время возведения в степень с использованием встроенного в Sage целочисленного типа данных, реализованного (в Cython) с использованием библиотеки GMP:

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

Используя интерфейс PARI из библиотеки C:

```
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 ведет себя лучше, но только немного (как и ожидалось, ведь версия PARI, встроенная в Sage, использует GMP для работы с целыми числами).

Вы также можете замерить время выполнения блока команд с помощью `cputime`, как показано ниже:

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

Команда `walltime` ведет себя так же, как `cputime`, но она измеряет настоящее время.

Мы также можем возвести число в степень, используя системы компьютерной алгебры, включённые в Sage. В каждом случае мы запускаем простую команду в системе чтобы запустить сервер для этой программы. Самое точное - время это Wall time. Однако, если существует существенная разница между этим значением и процессорным временем (CPU time), то, возможно, есть смысл проверить систему на наличие проблем производительности.

```
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: gap(0)
0
sage: time g = gap.eval('1938^99484;;')
CPU times: user 0.00 s, sys: 0.00 s, total: 0.00 s
Wall time: 1.02
```

Заметьте, что GAP и Maxima являются самыми медленными в этом тесте (тест был проведен на машине `sage.math.washington.edu`). Так как они работают с другим интерфейсом, надстроенным над ними, судить об абсолютной производительности этих систем не стоит.

3.5 Ошибки и исключения

Когда что-то идет не так, обычно можно увидеть исключение Python (Python "exception"). Python даже попытается предположить, что вызвало ошибку. Часто вы можете видеть имя исключения, например, `NameError` или `ValueError` (см. Python Reference Manual [Py] для полного списка исключений). Например,

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

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

Интерактивный отладчик может быть полезным для понимая того, что пошло не так. Отладчик можно включать или выключать командой `%pdb` (по умолчанию он выключен). Приглашение командной строки `ipdb>` появляется на экране, если случилось исключение и отладчик был включен. Из отладчика вы можете вывести на экран состояние любой локальной переменной и двигаться вверх и вниз по стеку (execution stack). Например,

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

ipdb>
```

Для получения списка команд отладчика введите `?` в командной строке `ipdb>`:

```
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
```

Нажмите Ctrl-D или введите `quit` чтобы вернуться в Sage.

3.6 Обратный поиск и автодополнение

Сначала создадим трехмерное векторное пространство $V = \mathbf{Q}^3$ следующим образом:

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

Можно использовать сокращенное обозначение:

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

Введите начало команды, потом нажмите `Ctrl-p` (или просто нажмите стрелку вверх на клавиатуре) чтобы вернуться к любой из строк, которые вы вводили, начинающейся с таких же символов. Это работает даже если вы полность вышли из Sage и перезапустили его позже. Можно использовать и обратный поиск по истории команд с помощью `Ctrl-r`. Все эти возможности используют пакет `readline` который доступен почти на всех разновидностях GNU/Linux.

Можно с легкостью вывести список всех функций для V, используя автодополнение. Просто введите `V.`, потом нажмите `[TAB]` на своей клавиатуре:

```
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
```

Если вы введете первые несколько символов команды, а потом нажмёте `[TAB]`, вы получите функции, которые начинаются с этих символов.

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

Если вам интересно, что делает какая-нибудь функция, например coordinates, введите `V.coordinates?` для получения справки или `V.coordinates??` для получения исходного кода (объясняется в следующем разделе).

3.7 Встроенная справочная система

Sage обладает встроенной справочной системой. Введите название функции со знаком ? для доступа к документации по этой функции.

```
sage: V = QQ^3
sage: V.coordinates?
Type:           instancemethod
Base Class:     <type '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]
```

Как показано выше, вывод показывает тип объекта, файл, в котором он определен и полезное описание функции с примерами, которые можно вставить в вашу текущую сессию. Почти все примеры подвергаются регулярной автоматической проверке на предмет работоспособности и наличия требуемого поведения.

Другая возможность хорошо отражает дух открытого программного обеспечения: если `f` это функция Python'a, то `f??` выведет исходный код, который определяет `f`. Например,

```
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()
```

Отсюда мы знаем, что все, что делает функция `coordinates`, это вызов функции `coordinate_vector` и превращает результат в список. Что делает функция `coordinate_vector`?

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

Функция `coordinate_vector` удерживает введенные значения во внешнем пространстве, что позволяет добиться такого же эффекта, как при вычислении вектора коэффициентов переменной v с точки зрения V. Пространство V уже внешнее, так как оно является $\mathbf{Q}^3$. Существует также функция

coordinate_vector для подпространств, и она ведет себя по-иному. Мы создим подпространство и посмотрим:

```
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)
```

(Если вы считаете, что существующая реализация неэффективна, пожалуйста, зарегистрируйтесь и помогите оптимизировать линейную алгебру.)

Вы также можете ввести help(имя_команды) или help(класс) для получения справки о классах или функциях в стиле man-страниц.

```
sage: help(VectorSpace)
Help on class VectorSpace ...

class VectorSpace(__builtin__.object)
 |  Create a Vector Space.
 |
 |  To create an ambient space over a field with given dimension
 |  using the calling syntax ...
 :
 :
```

Когда вы вводите q для выхода из справочной системы, ваша сессия находится в том же состоянии, что и до этого. Справка не захламляет ваш экран, в отличие от формы function_name?, которая иногда может оставлять информацию в вашей сессии. Особенно полезно использовать help(module_name). Например, векторные пространства описаны в sage.modules.free_module, поэтому введите help(sage.modules.free_module) для документации обо всем модуле. Когда вы просматриваете документацию в справочной системе, вы можете осуществлять поиск с помощью / и в обратном порядке с помощью ?.

3.8 Сохранение и загрузка отдельных объектов

Допустим вы вычислили матрицу или хуже: сложное пространство модулярных символов, и хотите сохранить его для работы в будущем. Как это сделать? Есть несколько способов, которыми компьютерные алгебры пользуются для сохранения объектов.

1. **Сохранить игру:** Поддерживается сохранение и загрузка только полных сессий (например, GAP, Magma).
2. **Унифицированный ввод/вывод:** Вывод объектов на экран в таком виде, в котором они могут быть считаны позже. (GP/PARI).
3. **Eval**: Легкий способ запуска любого кода в интерпретаторе (например, Singular, PARI).

Так как Sage построен на Python'е, он использует иной подход: каждый объект может быть превращен в строку, из которой в последствии можно восстановить объект. Способ схож со способом унификации ввода и вывода, как в PARI, но в случае с Sage нет необходимости выводить объект на экран в самой неудобной форме. Также, поддержка сохранения и загрузки (в большинстве случаев) полностью автоматична, не требует дополнительного программирования; это просто возможность Python'а, которая была включена в язык с самого начала.

Почти любой объект x может быть сохранен в сжатой форме на диск при помощи команды ''save(x, filename)" (или во многих случаях ''x.save(filename)''). Для загрузки объекта введите ''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')
```

Теперь выйдите из Sage и перезапустите. Теперь вы можете получить ''A" обратно:

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

То же самое можно делать и с более сложными объектами, например эллиптическими кривыми. Вся информация об объекте (которая находится в кеше) сохраняется вместе с объектом. Например,

```
sage: E = EllipticCurve('11a')
sage: v = E.anlist(100000)              # требует некоторого времени...
sage: save(E, 'E')
sage: quit
```

Сохраненная версия `E` занимает 153 килобита, так как в нем содержатся первые 100000 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)              # моментально!
```

(В Python, сохранение и загрузка осуществляется модулем `cPickle`. Объект Sage `x` может быть сохранен с помощью `cPickle.dumps(x, 2)`. Обратите внимание на 2!)

Sage не может сохранять и загружать объекты, созданные в других системах компьютерной алгебры, таких как GAP, Singular, Maxima и пр. Они загружаются в состоянии, которое помечено как "invalid". Хотя, в GAP многие объекты выводятся в форме, из которой их потом можно восстановить, но многие не выводятся в такой форме, поэтому их восстановление из такого вида нарочно запрещено.

```
sage: a = gap(2)
sage: a.save('a')
sage: load('a')
Traceback (most recent call last):
...
ValueError: The session in which this object was defined is no longer
running.
```

Объекты GP/PARI могут быть сохранены и загружены, так как их вид при выводе на экран достаточен для восстановления объекта.

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

Сохраненные объекты могут быть загружены позже на компьютерах с другой архитектурой или операционной системой, например, вы можете сохранить огромную матрицу в 32-битной OS X и загрузить ее в 64-битную GNU/Linux, привести к ступенчатой форме и переместить обратно. Также во многих случаях вы можете загружать объекты в версии Sage, отличные от версии, на которой они были сохранены. Все атрибуты объекта сохраняются вместе с классом (но не включая исходный код), который описывает объект. Если класс более не существует в новой версии Sage, тогда объект не может быть загружен в эту новую версию. Но если вы загрузите ее на версию ниже, получите словарь объектов (с помощью `x.__dict__`) и сохраните словарь, то сможете загрузить его в новую версию.

3.8.1 Сохранение в виде текста

Вы также можете сохранять объекты в виде набора ASCII символов в простой текстовый файл простым открытием файла и сохранением строки, которая выражает (описывает) объект (вы можете записывать несколько объектов). Не забудьте закрыть файл после добавления данных.

```
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()
```

3.9 Сохранение и загрузка полных сессий

Sage обладает очень гибкими возможностями сохранения и загрузки полных сессий.

Команда `save_session(sessionname)` сохраняет все переменные, которые вы задали в текущей сессии в виде словаря в заданном `sessionname`. (В редком случае, когда объект не поддерживает сохранения, он просто не будет включен в словарь.) В результате будет создан файл с расширением `.sobj` и может быть загружен как любой другой объект. Когда вы загружаете сохраненные объекты в сессию, вы получаете словарь, ключами которого являются имена переменных, а значениями — объекты.

Вы можете использовать команду `load_session(sessionname)`, чтобы загрузить переменные, описанные в `sessionname`, в текущую сессию. Заметьте, что это не удаляет переменные, заданные в этой сессии. Вместо этого, две сессии объединяются.

Для начала запустим Sage и зададим несколько переменных.

```
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
```

Далее, сохраним нашу сессию, что включит в себя сохранение всех заданных выше переменных в файл. Потом мы проверим информацию о файле. Его размер — 3 килобайта.

```
sage: save_session('misc')
Saving a
Saving M
Saving t
Saving E
sage: quit
was@form:~/tmp$ ls -l misc.sobj
-rw-r--r--  1 was was 2979 2006-01-28 19:47 misc.sobj
```

Наконец, мы перезапустим Sage, зададим дополнительную переменную и загрузим сохраненную сессию.

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

Каждая сохраненная переменная снова является переменной. Кроме того, переменная `b` не была перезаписана.

```
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
```

3.10 Интерфейс Notebook

Sage notebook запускается с помощью следующей команды

```
sage: notebook()
```

введенной в командной строке. Она запустит Sage notebook и откроет его в браузере по умолчанию. Файлы состояния сервера хранятся в `$HOME/.sage/sage\_notebook`.

Другие параметры включают в себя:

```
sage: notebook("directory")
```

Этот параметр позволяет запустить сервер Notebook, используя файлы в заданной директории, вместо использования директории по умолчанию `$HOME/.sage/sage_notebook`. Это может оказаться полезным, если вы хотите иметь коллекцию рабочих листов, связанных с конкретным проектом, или если вы хотите запускать несколько отдельных серверов Notebook в одно время.

Когда вы запускаете Notebook, вначале он создает следующие файлы в директории `$HOME/.sage/sage_notebook`:

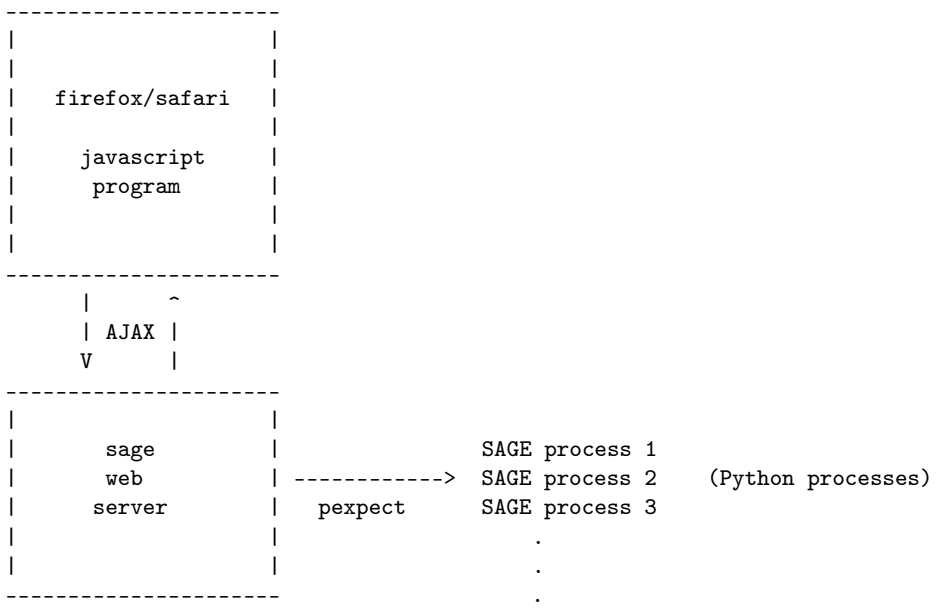
```
nb.sobj       (the notebook SAGE object file)
objects/      (a directory containing SAGE objects)
worksheets/   (a directory containing SAGE worksheets).
```

После создания этих файлов, Notebook запускает веб-сервер.

Notebook — это коллекция учетных записей пользователей (аккаунтов), каждый из которых может иметь любое количество рабочих листов. Когда вы создаете новый рабочий лист, информация, которая описывает его, сохраняется в директории `worksheets/username/number`. В каждой такой директории находится простой текстовый файл `worksheet.txt`; если что-то случится с вашими рабочими листами, или с Sage, или что-нибудь еще пойдет не так, то текстовый файл, который легко читается, поможет восстановить ваш лист полностью.

В Sage введите `notebook?` для получения подробной информации о том, как запустить сервер Notebook.

Следующая диаграмма иллюстрирует архитектуру Sage Notebook:



Для получения справки о команде Sage, `cmd`, в notebook введите, `cmd?` и нажмите `<esc>` (не `<shift-enter>`).

Для получения справки о горячих клавишах интерфейса notebook нажмите ссылку `Help`.

Глава 4

Интерфейсы

Краеугольным камнем Sage является поддержка вычислений с использованием объектов из разных систем компьютерной алгебры, которые находятся "под одной крышей" и используют общий интерфейс и чистый язык программирования.

Методы console и interact интерфейса делают разные вещи. Например, используя GAP как пример:

1. `gap.console()`: Открывает консоль GAP и передает управление GAP'у. Здесь Sage выступает в роли удобной командной строки, наподобие оболочки Bash в GNU/Linux.
2. `gap.interact()`: Это удобный способ взаимодействия с запущенным интерфейсом GAP, "заполненным" объектами Sage. Вы можете импортировать объекты Sage в сессию GAP (даже из интерактивного интерфейса), и пр.

4.1 GP/PARI

PARI это компактная, очень продуманная и хорошо оптимизированная программа на C, сосредоточенная на теории чисел. Существует два раздельных интерфейса, которые вы можете использовать в Sage:

- `gp` - gp - интерпретатор "**G** о **P** ARI" , и
- `pari` - pari - C-библиотека PARI.

Например, следующие две строчки выполняют одну и ту же операцию. Они выглядят идентично, но вывод на самом деле отличается, а за кулисами происходят совсем разные вещи.

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

В первом случае отдельная копия интерпретатора GP запускается как сервер, и строка `'znprimroot(10007)'` отправляется в него, вычисляется с помощью GP, и результат записывается в переменную в GP (которая занимает пространство в памяти процесса GP и не будет освобождена). После этого значение переменной выводится на экран. Во втором случае отдельная программа не запускается, и строка `'znprimroot(10007)'` вычисляется конкретной функцией C-библиотеки PARI. Результат сохраняется в heap-памяти Python'а, которая освобождается после того, как переменная перестает использоваться. У объектов разный тип:

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

```
sage: type(pari('znprimroot(10007)'))
<type 'sage.libs.pari.gen.gen'>
```

Так какой же способ использовать? Это зависит от того, что вы делаете. Интерфейс GP может делать все, что может делать программа GP/PARI, запускаемая из командной строки, потому как он запускает эту программу. Вы можете загрузить сложную программу PARI и запустить ее. С другой стороны, интерфейс PARI (через C-библиотеку) имеет намного больше ограничений. Во-первых, не все функции в ней реализованы. Во-вторых, много кода, например, численное интегрирование, не будет работать через интерфейс PARI. Интерйес PARI может быть намного быстрее и понятнее, чем сравнению с GP.

(Если у интерфейса GP закончится память при вычислении данной строки, он автоматически и без предупреждения удовит размер стека и попробует вычисление еще раз. Поэтому ваши вычисления всегда будут произведены корректно, если вы правильно расчитаете размер необходимой памяти. Этот удобный трюк не входит в арсенал простого интерпретатора GP. Относительно интерфейса C-библиотеки PARI: он сразу копирует каждый созданный объект из стека PARI, поэтому стек никогда не растет. Однако, каждый объект не должен превышать размера в 100 мегабайт, или стек будет переполнен при создании объекта. Дополнительное копирование немного влияет на общую производительность.)

Sage использует C-библиотеку PARI, чтобы поддерживать функциональность, схожую с интерпретатором GP/PARI, но включая различные сложные операции по работе с памятью и язык программирования Python.

Сначала, создадим список PARI из списка Python.

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

Каждый объект PARI является объектом типа `py_pari.gen`. Тип PARI может быть получен с помощью функции-члена `type`.

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

В PARI, чтобы создать эллиптическую кривую, нужно ввести `ellinit([1,2,3,4,5])`. В Sage способ схож, только `ellinit` — это метод, который может быть вызван для любого объекта PARI, например наша `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]
```

Теперь, когда у нас есть объект эллиптическая кривая, мы можем вычислить что-нибудь.

```
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 поставляется с GAP 4.4.10 для вычислений в области дискретной математики, в особенности, в теории групп.

Вот пример функции `IdGroup` из GAP, которая использует базу данных небольших групп, которая должна быть установлена отдельно, как показано ниже.

```
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()    # optional - database_gap
[ 120, 34 ]
sage: G.Order()
120
```

Мы можем провести те же вычисления в Sage без прямого вызова интерфейса GAP следующим образом:

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

(Для функционала GAP следует установить два дополнительных пакета Sage. Введите `sage -optional` для списка и выберите пакет вида `gap\_packages-x.y.z`, потом введите `sage -i gap\_packages-x.y.z`. Сделайте то же для `database\_gap-x.y.z`. Некоторые не-GPL пакеты GAP могут быть установлены скачиванием их с сайта GAP [GAPkg], и распаковкой их в директорию `$SAGE_ROOT/local/lib/gap-4.4.10/pkg`.)

4.3 Singular

Singular предоставляет массивную и продуманную библиотеку для базиса Грёбнера, нахождения наибольшего общего делителя полиномов, базиса пространств плоских кривых Римана-Роха и факторизации, наряду с другими вещами. Мы покажем пример факторизации полиномов с несколькими переменными, используя интерфейс Singular в Sage (не вводите `....:`):

```
sage: R1 = singular.ring(0, '(x,y)', 'dp')
sage: R1
//   characteristic : 0
//   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')
```

Теперь когда мы определили f, мы выводим на экран и факторизуем.

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

Как и на примере GAP в *GAP*, мы можем совершить данную факторизацию без прямого указания интерфейса Sage (однако за кулисами Sage все равно используется интерфейс Singular). Не вводите `....::`

```
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 включена в Sage, так же как реализация Лиспа. Пакет gnuplot (который Maxima использует по умолчанию для построения графиков) распространяется как дополнительный пакет Sage. Кроме остальных вещей, Maxima позволяет производить символические манипуляции. Maxima может интегрировать и дифференцировать функции символически, решать обыкновенные дифференциальные уравнения 1го порядка, большую часть линейных обыкновенных дифференциальных уравнений 2го порядка, использовать преобразования Лапласа как метод для решения линейных обыкновенных дифференциальных уравнений любого порядка. Maxima также "знает" о большом наборе специальных функций, имеет возможность строить графики при помощи gnuplot, имеет методы решения и манипуляции матрицами (к примеру, метод Гаусса, нахождение собственных значений и векторов), а также умеет решать полиномы.

Мы проиллюстрируем работу Sage/Maxima с помощью матрицы, значения i, j которой являются i/j, для $i, j = 1, \ldots, 4$.

```
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()
[[[0,4],[3,1]],[[[1,0,0,-4],[0,1,0,-2],[0,0,1,-4/3]],[[1,2,3,4]]]]
```

Вот другой пример:

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

Наконец, мы покажем, как строить графики средствами `openmath`. Многие примеры являются модифицированными примерами из руководства к Maxima.

2-мерные графики нескольких функций (не вводите `....:`):

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

“Живой” трехмерный график, который вы можете вращать мышкой (не вводите `....:`):

```
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]')
```

Следующий график — это знаменитая Лента Мёбиуса (не вводите `....:`):

```
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]')
```

Следующий график — это знаменитая Бутылка Клейна (не вводите `....:`):

```
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)")
-5*sin(x)*(sin(x/2)*sin(2*y)+cos(x/2)*cos(y)+3.0)
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]')
```

Глава 5

Программирование

5.1 Загрузка и прикрепление файлов Sage

Следующее показывает, как подгружать программы в Sage, записанные в отдельный файл. Создайте файл `example.sage` со следующим содержанием:

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

Вы можете прочитать и выполнить `example.sage` с помощью команды `load`.

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

Вы также можете прикрепить файл Sage к запущенной сессии в помощью команды `attach`:

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

Теперь если вы измените файл `example.sage` и введете пустую строку в Sage (т.е. нажмите `return`), то содержимое `example.sage` будет автоматически перегружено в Sage.

В частности, `attach` автоматически перегружает файл, как только он изменен, что очень удобно при поиске ошибок в коде, тогда как `load` загружает файл лишь единожды.

Когда `example.sage` загружается в Sage, он переводится в Python, а затем выполняется с помощью интерпретатора Python. Затраты на данную операцию минимальны; в основном, это включает в себя перевод целых констант в `Integer()`, дробных констант в `RealNumber()`, замену `^` на `**` и, например, `R.2` на `R.gen(2)`. Переведенная версия `example.sage` будет содержаться в той же директории, что `example.sage`, под названием `example.sage.py`. Данный файл будет содержать следующий код:

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

Целые контстанты переведены и `^` заменено на `**`. (В Python `^` означает “исключающее ИЛИ” и `**` означает “возведение в степень”.)

Данные операции выполняются в `sage/misc/interpreter.py`.)

Вы имеете возможность вставлять многострочный код с отступами в Sage до тех пор, пока есть новые строки для новых блоков (это необязательно для файлов). Однако, лучшим способом для вставки

такого кода является сохранение в файл и использование `attach`, как описано выше.

5.2 Создание компилированного кода

Скорость — важная составляющая в математических вычислениях. Хотя Python является высокоуровневым языком программирования, некоторые вычисления могут быть выполнены на несколько порядков быстрее в Python при использовании статических типов данных при компилировании. Некоторые компоненты Sage были бы слишком медленными, будь он написан целиком на Python. Для этого Sage поддерживает компилированную "версию" Python, которая называется Cython ([Cyt] и [Pyr]). Cython одновременно похож и на Python, и на C. Большинство конструкций Python, включая представление списков, условные выражения, код наподобие `+=`, разрешены; вы также можете импортировать код, написанный в других модулях Python. Кроме того, вы можете объявлять произвольные переменные C и напрямую обращаться к библиотекам C. Конечный код будет сконвертирован в C и обработан компилятором C.

Для того, чтобы создать компилируемый код в Sage, объявите файл с расширением `.spyx` (вместо `.sage`). Если вы работаете с интерфейсом коммандной строки, вы можете прикреплять и загружать компилируемый код точно так же, как и интерпретируемый (на данный момент, прикрепление и загрузка кода на Cython не поддерживается в интерфейсе Notebook). Само компилирование происходит "за кулисами", не требуя каких-либо действий с вашей стороны. Компилированная библиотека общих объектов содержится в `$HOME/.sage/temp/hostname/pid/spyx`. Эти файлы будут удалены при выходе из Sage.

Пре-парсировка не применяется к spyx файлам. Например `1/3` превратится в 0 в spyx файле вместо рационального числа 1/3. Допустим, `foo` - это функция в библиотеке Sage. Для того, чтобы использовать ее из spyx-файла, импортируйте `sage.all` и примените `sage.all.foo`.

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

5.2.1 Доступ к функциям C из внешних файлов

Доступ к функциям C из внешних `*.c` файлов осуществляется довольно просто. Создайте файлы `test.c` и `test.spyx` в одной директории со следующим содержанием:

Код на языке C: `test.c`

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

Код на языке Cython: `test.spyx`:

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

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

Выполните:

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

```
sage: test(10)
11
```

В том случае, если понадобится дополнительная библиотека `foo` для того, чтобы скомпилировать код на C, полученный из файла Cython, добавьте `clib foo` в источник Cython кода. Аналогично, дополнительный C файл `bar` может быть добавлен в компиляцию с объявлением `cfile bar`.

5.3 Самостоятельные скрипты Python/Sage

Данный самостоятельный скрипт Sage раскладывает на множители целые числа, полиномы и т.д.:

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

import sys
from sage.all import *

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]))
```

Для того, чтобы использовать этот скрипт, `SAGE_ROOT` должен быть в PATH. Если вышеописанный скрипт называется `factor`, следующее показывает, как его выполнить:

```
bash $ ./factor 2006
2 * 17 * 59
bash $ ./factor "32*x^5-1"
(2*x - 1) * (16*x^4 + 8*x^3 + 4*x^2 + 2*x + 1)
```

5.4 Типы данных

Каждый объект в Sage имеет определенный тип. Python включает в себя большой спектр встроенных типов тогда, как библиотеки Sage добавляют еще больше. Встроенные типы данных Python включают в себя символьные строки, списки, кортежи, целые и дробные числа:

```
sage: s = "sage"; type(s)
<type 'str'>
sage: s = 'sage'; type(s)      # Вы можете использовать двойные или одинарные кавычки
<type 'str'>
sage: s = [1,2,3,4]; type(s)
<type 'list'>
sage: s = (1,2,3,4); type(s)
<type 'tuple'>
sage: s = int(2006); type(s)
<type 'int'>
sage: s = float(2006); type(s)
<type 'float'>
```

В свою очередь Sage добавляет много других типов данных, например, векторное поле:

```
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'>
```

Только определенные функции могут быть применены к `V`. В других математических программах функции вызывались бы в "функциональном" виде: `foo(V,...)`. В Sage определенные функции прикреплены к типу (или классу) `V` и вызываются с помощью объектно-ориентированного синтаксиса, как в Java или C++, например, `V.foo(...)`. Это способствует тому, что именная область видимости не захламляется десятками тысяч функций, и означает, что многие функции с разным содержанием могут быть названы "foo" без проверки типов аргументов. Также, если Вы используете имя функции повторно, эта функция все равно доступна (например, если Вы вызываете что-то наподобие `zeta`, а затем хотите вычислить значение функции Riemann-Zeta при 0.5, Вы можете напечатать `s=.5; s.zeta()`).

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

В некоторых часто встречающихся случаях, обычное функциональное обозначение также способствует удобству из-за того, что математические выражения могут выглядеть запутанно при исаользовании объектно-ориентированного обозначения. Например:

```
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
```

Для того, чтобы перечислить все члены-функции для A, напечатайте `A.`, а затем нажмите кнопку `[tab]` на Вашей клавиатуре, как описано в разделе *Обратный поиск и автодополнение*

5.5 Списки, кортежи и последовательности

Тип данных список может хранить в себе элементы разных типов данных. Как в C, C++ и т.д., но в отличие от других алгебраических систем, элементы списка начинаются с индекса 0:

```
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)
<type 'list'>
sage: v[0]
2
sage: v[2]
5
```

При индексировании списка, применение индексов, не являющихся целым числом Python, сработает нормально.

```
sage: v = [1,2,3]
sage: v[2]
3
sage: n = 2      # целое число Sage
sage: v[n]       # работает правильно
3
sage: v[int(n)]  # тоже работает правильно
3
```

Функция `range` создает список целых чисел, используемых Python(не Sage):

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

Это удобно, когда для создания списков используется вид списка:

```
sage: L = [factor(n) for n in range(1, 15)]
sage: print 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]
```

Для большего понимания списков см. [PyT].

Расщепление списков - это очень удобный инструмент. Допустим `L` - это список, тогда `L[m:n]` вернет под-список L, полученный, начиная с элемента на позиции m и заканчивая элементом на позиции $(n-1)$, как показано ниже.

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

Кортежи имеют сходство со списками, однако они неизменяемы с момента создания.

```
sage: v = (1,2,3,4); v
(1, 2, 3, 4)
```

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

Последовательности - это тип данных, схожий по свойствам со списком. Последовательности как тип данных не встроены в Python в отличие от списков и кортежей. По умолчанию, последовательность является изменяемой, однако используя метод `set_immutable` из класса `Sequence`, она может быть сделана неизменяемой, как показано в следующем примере. Все элементы последовательности имеют общего родителя, именуемого универсумом последовательности.

```
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])
<type '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.
```

Последовательности могут быть использованы везде, где могут быть использованы списки:

```
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))
<type 'list'>
```

Базис для векторного поля является неизменяемой последовательностью, так как очень важно не изменять их. Это показано в следующем примере:

```
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
```

5.6 Словари

Словарь (также именуемый ассоциативным массивом) - это сопоставление 'хэшируемых' объектов (как строки, числа и кортежи из них; см. документацию Python: http://docs.python.org/tut/node7.html и http://docs.python.org/lib/typesmapping.html) произвольным объектам.

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

Третий ключ показывает, что индексы словаря могу быть сложными, как, например, кольцо целых чисел.

Можно превратить вышеописанный словарь в список с тем же содержимым:

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

Часто используемой практикой является произведение итераций по парам в словаре:

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

Как показывает последний пример, словарь не упорядочен.

5.7 Множества

В Python есть встроенный тип множество. Главным преимуществом этого типа является быстрый просмотр, проверка того, принадлежит ли элемент множеству, а также обычные операции из теории множеств.

```
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}
```

В Sage также имеется свой тип данных множество, который (в некоторых случаях) осуществлен с использованием встроенного типа множество Python, но включает в себя функциональность, связанную с Sage. Создайте множество Sage с помощью `Set(...)`. Например,

```
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
```

5.8 Итераторы

Итераторы - это сравнительно недавнее добавление в Python, которое является очень полезным в математических приложениях. Несколько примеров использования итераторов приведены ниже; подробнее см. [PyT]. Здесь создается итератор для квадратов неотрицательных чисел до 10000000.

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

Следующий пример - создание итераторов из простых чисел вида $4p + 1$ с простым p и просмотр нескольких первых значений:

```
sage: w = (4*p + 1 for p in Primes() if is_prime(4*p+1))
sage: w         # random output на следующей строке 0xb0853d6c может быть другим шестнадцатиричным числом
<generator object at 0xb0853d6c>
sage: next(w)
13
sage: next(w)
29
sage: next(w)
53
```

Определенные кольца, как и конечные поля и целые числа, имеют итераторы:

```
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)
```

5.9 Циклы, функции, управляющие конструкции и сравнения

Мы уже видели несколько примеров с использованием циклов `for`. В Python цикл `for` имеет табулированную структуру:

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

Заметьте двоеточие на конце выражения("do" или "od", как GAP или Maple, не используются), а отступы перед "телом" цикла, в частности, перед `print(i)`. Эти отступы важны. В Sage отступы ставятся автоматически при нажатии `enter` после ":", как показано ниже.

```
sage: for i in range(5):
....:     print(i)  # нажмите Enter дважды
....:
0
1
2
3
4
```

Символ = используется для присваивания. Символ == используется для проверки равенства:

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

Имейте в виду, как табуляция определяет структуру блоков для операторов `if`, `for` и `while`:

```
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
```

Конечно, это не эффективная реализация символа Лежандра! Данный пример служит лишь иллюстрацией разных аспектов программирования в Python/Sage. Функция {kronecker}, встроенная в Sage,

подсчитывает символ Лежандра эффективно с использованием библиотек C, в частности, с использованием PARI.

Сравнения ==, !=, <=, >=, >, < между числами автоматически переводят оба члена в одинаковый тип:

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

Практически любые два объекта могут быть сравнены.

```
sage: 2 < CC(3.1,1)
True
sage: 5 < VectorSpace(QQ,3)    # random output
True
```

Используйте переменные bool для символьных неравенств:

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

При сравнении объектов разного типа в большинстве случаев Sage попытается найти каноническое приведение обоих к общему родителю. При успехе, сравнение выполняется между приведёнными объектами; если нет, то объекты будут расценены как неравные. Для проверки равенства двух переменных используйте `is`. Например:

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

В следующих двух строках первое неравенство дает `False`, так как нет канонического морфизма $\mathbf{Q} \to \mathbf{F}_5$, поэтому не существует канонического сравнения между 1 в $\mathbf{F}_5$ и $1 \in \mathbf{Q}$. Однако, существует каноническое приведение $\mathbf{Z} \to \mathbf{F}_5$, поэтому второе выражение дает `True`. Заметьте, порядок не имеет значения.

```
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
```

ВНИМАНИЕ: Сравнение в Sage проводится более жёстко, чем в Magma, которая объявляет $1 \in \mathbf{F}_5$ равным $1 \in \mathbf{Q}$.

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

5.10 Профилирование

Автор раздела: Martin Albrecht (malb@informatik.uni-bremen.de)

> “Преждевременная оптимизация - это корень всего зла.” - Дональд Кнут

Часто очень полезно проверять код на слабые места, понимать, какие части отнимают наибольшее время на вычисления; таким образом можно узнать, какие части кода надо оптимизировать. Python и Sage предоставляет несколько возможностей для профилирования (так называется этот процесс).

Самый легкий путь - это использование команды `prun`. Она возвращает краткую информацию о том, какое время отнимает каждая функция. Далее следует пример умножения матриц из конечных полей:

```
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)
  ...
```

В данном примере `ncalls` - это количество вызовов, `tottime` - это общее время, затраченное на определенную функцию (за исключением времени вызовов суб-функций), `percall` - это отношение `tottime` к `ncalls`. `cumtime` - это общее время, потраченное в этой и всех суб-функциях, `percall` - это отношение `cumtime` к числу примитивных вызовов, `filename:lineno(function)` предоставляет информацию о каждой функции. Чем выше функция находится в этом списке, тем больше времени она отнимает.

`prun?` покажет детали о том, как использовать команду профилирования и понимать результат ее использования.

Профилирующая информация может быть вписана в объект для более подробного изучения:

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

Заметка: ввод `stats = prun -r A\*A` отобразит синтаксическую ошибку, так как `prun` - это команда оболочки IPython, а не обычная функция.

Для графического отображения профилирующей информации, Вы можете использовать hotshot - небольшой скрипт, названный hotshot2cachetree и программу `kcachegrind` (только в Unix). Тот же пример с использованием hotshot:

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

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

Результат будет помещен в файл `pythongrind.prof` в текущей рабочей директории. Для визуализации эта информация может быть переведена в формат cachegrind.

В системной оболочке введите

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

Выходной файл `cachegrind.out.42` теперь может быть проанализирован с помощью `kcachegrind`. Заметьте, что обозначение `cachegrind.out.XX` должно быть соблюдено.

Использование SageTeX

Встроенный в Sage пакет SageTeX позволяет внедрять результаты вычислений в документ типа LaTeX. Для того, чтобы использовать данный пакет, понадобится "установить" его в локальную систему TeX (под "установкой" подразумевается копирование одного файла). См. *Установка*, а также раздел "Make SageTeX known to TeX" Руководства по установке Sage (данная ссылка ведет к локальному размещению копии руководства по установке).

В этом уроке показан небольшой пример использования SageTeX. Полная документация находится в `SAGE_ROOT/local/share/texmf/tex/generic/sagetex`, где `SAGE_ROOT` - это директория, в которой установлен Sage. Эта папка содержит документацию, файл с примером и полезные скрипты Python.

Для начала работы с SageTeX следуйте указаниям по установке (в *Установка*) и вставьте следующия текст в файл названный, скажем, `st_example.tex`:

Предупреждение: Нижеследующий текст может содержать несколько сообщений об ошибках, связанных с неизвестными управляющими последовательностями, если Вы используете интерактивную помощь. Откройте статическую версию, чтобы увидеть правильный текст.

```
\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}
    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}
```

Запустите LaTeX для `st_example.tex`. Заметьте, что LaTeX будет жаловаться на некоторые вещи, как-то:

> 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.sage, and then run LaTeX on st_example.tex again.

Среди файлов, сгенерированных после запуска LaTeX, есть файл `st_example.sage`, являющийся скриптом Sage. Сообщение, показанное выше, предлагало запустить `st_example.sage`, поэтому стоит так и сделать. Затем будет предложено запустить LaTeX для `st_example.tex` еще раз; перед этим будет создан файл `st_example.sout`. Этот файл содержит результаты вычислений в Sage в формате, удобном для LaTeX. Новая папка, содержащая EPS-файл с графиком, также будет создана автоматически. Запустите LaTeX еще раз: все, что было вычислено в Sage, теперь включено в Ваш документ.

Перечислим шаги:

- запустите LaTeX для .tex файла;
- запустите Sage для сгенерированного .sage файла;
- запустите LaTeX еще раз.

Пункт с запуском Sage можно пропустить, если никакие изменения не были применены к командам Sage в документе.

SageTeX предлагает много возможностей, и так как Sage и LaTeX являются мощными инструментами, то стоит изучить `SAGE_ROOT/local/share/texmf/tex/generic/sagetex`.

Глава 7

Послесловие

7.1 Почему Python?

7.1.1 Преимущества Python

Основной язык реализации Sage — это Python (см. [Py]), однако код, который должен обрабатываться быстро, написан на компилируемом языке. У Python есть ряд преимуществ:

- **Сохранение объектов** широко используется в Python. В Python присутствует поддержка сохранения (почти) любых объектов на диск или в базу данных.
- Замечательная поддержка **документации** функций и пакетов в исходном коде, включая автоматический доступ к документации и автоматическое тестирование всех примеров. Примеры проверяются автоматически на регулярной основе и их правильная работоспособность гарантирована.
- **Управление памятью**: Python имеет продуманный и стабильный менеджер памяти и сборщик мусора, которые исправно работают с циклическими ссылками и позволяют использовать локальные переменные в файлах.
- Python имеет **множество пакетов**, доступных уже сейчас, которые могут быть интересны пользователям Sage: численный анализ и линейная алгебра, 2D и 3D визуализация, сеть (для распределенных вычислений и серверов, например с помощью twisted), поддержка баз данных и т.д.
- **Портируемость:** Python с легкостью компилируется на большинстве платформ в считанные минуты.
- **Работа с исключениями:** Python содержит сложный и продуманный механизм работы с исключениями, благодаря чему программы продолжают работать даже при возникновении ошибок в вызываемом ими коде.
- **Отладчик:** Python включает в себя отладчик, так что когда программа не работает по какой-то причине, пользователь может получить доступ к истории стека, проверить состояние необходимых переменных, перемещаться по стеку.
- **Профилировщик:** существует профилировщик Python, который запускает код и создает отчет по количеству вызовов и времени работы каждой функции.
- **Язык:** Вместо того, чтобы писать **новый язык** для математики, как это было сделано для Magma, Maple, Mathematica, Matlab, GP/PARI, GAP, Macaulay 2, Simath, и т.д., мы используем язык Python, который является популярным языком программирования; он активно развивается и оптимизируется сотнями опытных специалистов по программному обеспечению (см. [PyDev]).

7.1.2 Пре-парсер: Различия между Sage и Python

В некоторых математических аспектах Python может ввести в заблуждение, поэтому Sage ведет себя немного другим образом.

- **Обозначение возведения в степень:** `**` вместо `^`. В Python, `^` означает "исключительно или (xor)", а не возведение в степень, так в Python:

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

  Использование `^` может показаться странным; это не так важно для математических исследований, потому как "исключительно или" используется довольно редко. Для удобства, Sage использует пре-парсер для проверки кода перед тем, как он передается в Python, и символ `^` (если он не находится в строке) заменяет на `**`:

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

- **Деление целых чисел:** Выражение `2/3` в Python означает не то, чего ожидает математик. В Python, если `m` и `n` - целые числа, то `m/n` также целое число, если быть точнее, то целая часть от деления `m` на `n`. Следовательно `2/3=0`. В сообществе Python обсуждается вариант изменения оператора так, чтобы `2/3` возвращало число с плавающей точкой `0.6666...`, а `2//3` возвращало `0`.

  В интерпретаторе Sage мы используем обозначение `Integer( )` и деление используем как конструктор для рациональных чисел. Например:

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

- **Большие целые числа:** Python имеет встроенную поддержку целых чисел произвольной точности в дополнение к C-int'ам. Они намного медленнее, чем то, что предоставляет GMP, а также имеют свойство: символ `L` в конце, чтобы отличать их от переменных типа int (и это не изменится в ближайшем будущем). Sage использует целые числа произвольной точности с помощью GMP C-библиотеки, и они выводятся на экран без L.

Вместо изменения интерпретатора Python (как поступили некоторые люди для внутренних проектов), мы используем Python как есть, и применяем пре-парсер для IPython так чтобы поведение командной строки IPython соответствовало ожиданиям математиков. Это означает, что любой существующий код на Python может быть использован в Sage. Однако, нужно придерживаться стандарта Python при написании пакетов, которые будут импортированы в Sage.

(Чтобы установить пакет Python, который, скажем, вы нашли в интернете, следуйте инструкции, но запускайте `sage -python` вместо `python`. Очень часто это означает, что нужно ввести `sage -python setup.py install`.)

7.2 Как принять участие в разработке Sage?

Если вы хотите помочь в разработке Sage, это будет оценено по достоинству! Помощь может варьироваться от внесения изменений в код до дополнения справочной информации и нахождения багов.

Поищите информацию для разработчиков на главной странице Sage; кроме всего прочего, вы можете найти список проектов, связанных с Sage, отсортированных по приоритету и категории. Руководство разроботчика Sage содержит полезную информацию; вы также можете узнать больше в Google-группе `sage-devel`.

7.3 Как правильно ссылаться на Sage?

Если вы используете Sage для написания работы, пожалуйста, укажите, что вычисления были произведены с помощью Sage. Включите

```
[Sage] William A. Stein et al., Sage Mathematics Software (Version 4.3).
       The Sage Development Team, 2009, http://www.sagemath.org.
```

в раздел библиографии (заменяя 4.3 версией Sage, которую вы используете). Кроме того, пожалуйста, постарайтесь отследить, какие компоненты Sage были использованы для вычислений, например PARI?, GAP?, Singular? Maxima?, и укажите эти системы. Если вы сомневаетесь о том, какое программное обеспечение используется для вычислений, задайте вопрос в Google-группе `sage-devel`. См. *Полиномы одной переменной* для дальнейшего обсуждения этой темы.

Если вы прочитали это руководство от начала до конца, и у вас есть соображения по поводу времени, затраченного на него, пожалуйста, выскажите свое мнение в Google-группе `sage-devel`.

Наслаждайтесь Sage!

Глава 8

Дополнение

8.1 Приоритет бинарных арифметических операторов

Что такое `3^2*4 + 2%5`? Значение (38) определено по этой "таблице приоритета операторов". Таблица ниже основана на таблице из § 5.14 книги *Python Language Reference Manual*, G. Rossum and F. Drake. Операторы расположены в порядке возрастания старшинства.

Operators	Description
or	boolean or
and	boolean and
not	boolean not
in, not in	membership
is, is not	identity test
>, <=, >, >=, ==, !=	comparison
+, -	addition, subtraction
*, /, %	multiplication, division, remainder
**, ^	exponentiation

Следовательно, чтобы посчитать `3^2*4 + 2%5`, Sage расставляет скобки так: `((3^2)*4) + (2%5)`. Сначала считается `3^2`, то есть `9`, затем считаются `(3^2)*4` и `2%5`, и наконец они складываются.

Глава 9

Библиография

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

[Dive] Dive into Python, Freely available online at http://www.diveintopython.net/.

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

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

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

[Ip] The IPython shell http://ipython.scipy.org.

[Jmol] Jmol: an open-source Java viewer for chemical structures in 3D http://www.jmol.org/.

[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] The Python language http://www.python.org/ Reference Manual http://docs.python.org/ref/ref.html.

[PyDev] Guido, Some Guys, and a Mailing List: How Python is Developed, http://www.python.org/dev/dev_intro.html.

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

[PyT] The Python Tutorial http://www.python.org/.

[SA] Sage web site http://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). http://www.singular.uni-kl.de.

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