# Three Lectures about Explicit Methods in Number Theory Using Sage

William Stein

October 2008

**Abstract**

This article is about using the mathematical software Sage to do computations with number fields and modular forms. It was written for the October 2008 Bordeaux meeting on explicit methods in number theory (`http://www.math.u-bordeaux.fr/gtem2008/`). It assumes no prior knowledge about Sage, but assumes a graduate level background in algebraic number theory.

## Contents

## Introduction

Sage (see `http://sagemath.org`) is a comprehensive mathematical software system for computations in many areas of pure and applied mathematics. We program Sage using the mainstream programming language Python (see `http://python.org`), or its compiled variant Cython. It is also very easy to efficiently use code written in C/C++ from Sage.

The author of this article started the Sage project in 2005. Sage is free and open source, meaning you can change any part of Sage and redistribute the result without having to pay any license fees, and Sage can also leverage the power of commercial mathematical software such as Magma and Mathematica, if you happen to have access to those closed source commercial systems.

This paper assumes no prior knowledge of either Python or Sage. Our goal is to help number theorists do computations involving number fields and modular forms using Sage.

As you read this article, please try every example in Sage, and make sure things works as I claim, and do all of the exercises. Moreover, you should experiment by typing in similar examples and checking that the output you get agrees with what you expect.

To use Sage, install it on your computer, and use either the command line or start the Sage notebook by typing `notebook()` at the command line.

We show Sage sessions as follows:

```
sage: factor(123456)
2^6 * 3 * 643
```

This means that if you type `factor(123456)` as input to Sage, then you'll get `2^6 * 3 * 643` as output. If you're using the Sage command line, you type `factor(123456)` and press enter; if you're using the Sage notebook via your web browser, you type `factor(123456)` into an input cell and press shift-enter; in the output cell you'll see `2^6 * 3 * 643`.

After trying the `factor` command in the previous paragraph (do this now!), you should try factoring some other numbers.

**Exercise 0.1.** What happens if you factor a negative number? a rational number?

You can also draw both 2d and 3d pictures using Sage. For example, the following input plots the number of prime divisors of each positive integer up to 500.

```
sage: line([(n, len(factor(n))) for n in [1..500]])
```

And, this example draws a similar 3d plot:

```
sage: v = [[len(factor(n*m)) for n in [1..15]] for m in [1..15]]
sage: list_plot3d(v, interpolation_type='nn')
```

*The main difference between Sage and Pari is that Sage is vastly larger than Pari with a much wider range of functionality, and has many more datatypes and much more structured objects.* Sage in fact includes Pari, and a typical Sage install takes nearly a gigabyte of disk space, whereas a typical Pari install is much more nimble, using only a few megabytes. There are many number-theoretic algorithms that are included in Sage, which have never been implemented in Pari, and Sage has 2d and 3d graphics which can be helpful for visualizing number theoretic ideas, and a graphical user interface. Both Pari and Sage are free and open source, which means anybody can read or change anything in either program, and the software is free.

*The biggest difference between Sage and Magma is that Magma is closed source, not free, and difficult for users to extend.* This means that most of Magma cannot be changed except by the core Magma developers, since Magma itself is well over two million lines of compiled C code, combined with about a half million lines of interpreted Magma code (that anybody can read and modify). In designing Sage, we carried over some of the excellent design ideas from Magma, such as the parent, element, category hierarchy.

*Any mathematician who is serious about doing extensive computational work in algebraic number theory and arithmetic geometry is strongly urged to become familiar with all three systems*, since they all have their pros and cons. Pari is sleek and small, Magma has much unique functionality for computations in arithmetic geometry, and Sage has a wide range of functionality in most areas of mathematics, a large developer community, and much unique new code.

## 1 Number Fields

In Sage, we can create the number field $\mathbb{Q}(\sqrt[3]{2})$ as follows.

```
sage: K.<alpha> = NumberField(x^3 - 2)
```

The above creates *two* Sage objects, $K$ and $\alpha$. Here $K$ "is" (isomorphic to) the number field $\mathbb{Q}(\sqrt[3]{2})$, as we confirm below:

```
sage: K
Number Field in alpha with defining polynomial x^3 - 2
```

and $\alpha$ is a root of $x^3 - 2$, so $\alpha$ is an abstract choice of $\sqrt[3]{2}$ (no specific embedding of the number field $K$ into $\mathbb{C}$ is chosen by default in Sage-3.1.2):

```
sage: alpha^3
2
sage: (alpha+1)^3
3*alpha^2 + 3*alpha + 3
```

Note that we did *not* define $x$ above before using it. You could "break" the above example by redefining $x$ to be something funny:

```
sage: x = 1
sage: K.<alpha> = NumberField(x^3 - 2)
Traceback (most recent call last):
...
TypeError: polynomial (=-1) must be a polynomial.
```

The *Traceback* above indicates that there was an error. Potentially lots of detailed information about the error (a "traceback") may be given after the word `Traceback` and before the last line, which contains the actual error messages.

**Important:** *whenever you use Sage and get a big error, look at the last line for the actual error, and only look at the rest if you are feeling adventurous.* In the notebook, the part indicated by ... above is not displayed; to see it, click just to the left of the word *Traceback* and the traceback will appear.

If you redefine $x$ as above, but need to define a number field using the indeterminate $x$, you have several options. You can reset $x$ to its default value at the start of Sage, you can redefine $x$ to be a symbolic variable, or you can define $x$ to be a polynomial indeterminant (a polygen):

```
sage: reset('x')
sage: x
x
sage: x = 1
sage: x = var('x')
sage: x
x
sage: x = 1
sage: x = polygen(QQ, 'x')
sage: x
x
sage: x = 1
sage: R.<x> = PolynomialRing(QQ)
sage: x
x
```

One you have created a number field $K$, type `K.[tab key]` to see a list of functions. Type, e.g., `K.Minkowski_embedding?[tab key]` to see help on the `Minkowski_embedding` command. To see source code, type `K.Minkowski_embedding??[tab key]`.

```
sage: K.<alpha> = NumberField(x^3 - 2)
sage: K.[tab key]
```

## 1.1 Symbolic Expressions

Another natural way for us to create certain number fields is to create a symbolic expression and adjoin it to the rational numbers. Unlike Pari and Magma (and like Mathematica and Maple), Sage also supports manipulation of symbolic expressions and solving equations, without defining abstract structures such as a number fields. For example, we can define a variable $a = \sqrt{2}$ as an abstract symbolic object by simply typing `a = sqrt(2)`. When we type `parent(a)` below, Sage tells us the mathematical object that it views $a$ as being an element of; in this case, it's the ring of all symbolic expressions.

```
sage: a = sqrt(2)
sage: parent(a)
Symbolic Ring
```

In particular, typing `sqrt(2)` does *not* numerically extract an approximation to $\sqrt{2}$, like it would in Pari or Magma. We illustrate this below by calling Pari (via the gp interpreter) and Magma directly from within Sage. After we evaluate the following two input lines, copies of GP/Pari and Magma are running, and there is a persistent connection between Sage and those sessions.

```
sage: gp('sqrt(2)')
1.414213562373095048801688724
sage: magma('Sqrt(2)')                    # optional
1.41421356237309504880168872421
```

You probably noticed a pause when evaluated the second line as Magma started up. Also, note the `# optional` comment, which indicates that the line won't work if you don't have Magma installed.

Incidentally, if you want to numerically evaluate $\sqrt{2}$ in Sage, just give the optional `prec` argument to the `sqrt` function, which takes the required number of *bits* (binary digits) of precision.

```
sage: sqrt(2, prec=100)
1.4142135623730950488016887242
```

It's important to note in computations like this that there is not an *a priori* guarantee that `prec` bits of the *answer* are all correct. Instead, what happens is that Sage creates the number 2 as a floating point number with 100 bits of accuracy, then asks Paul Zimmerman's MPFR C library to compute the square root of that approximate number.

We return now to our symbolic expression $a = \sqrt{2}$. If you ask to square $a + 1$ you simply get the formal square. To expand out this formal square, we use the expand command.

```
sage: a = sqrt(2)
sage: (a+1)^2
(sqrt(2) + 1)^2
sage: expand((a+1)^2)
2*sqrt(2) + 3
```

Given any symbolic expression for which Sage can computes its minimal polynomial, you can construct the number field obtained by adjoining that expression to $\mathbb{Q}$. The notation is quite simple – just type `QQ[a]` where `a` is the symbolic expression.

```
sage: a = sqrt(2)
sage: K.<b> = QQ[a]
sage: K
Number Field in sqrt2 with defining polynomial x^2 - 2
sage: b
sqrt2
sage: (b+1)^2
2*sqrt2 + 3
sage: QQ[a/3 + 5]
Number Field in a with defining polynomial x^2 - 10*x + 223/9
```

You can't create the number field $\mathbb{Q}(a)$ in Sage by typing `QQ(a)`, which has a *very different* meaning in Sage. It means "try to create a rational number from $a$." Thus `QQ(a)` in Sage is the analogue of `QQ!a` in Magma (Pari has no notion of rings such as `QQ`).

```
sage: a = sqrt(2)
sage: QQ(a)
Traceback (most recent call last):
...
TypeError: unable to convert sqrt(2) to a rational
```

In general, if $X$ is a ring, or vector space or other "parent structure" in Sage, and $a$ is an element, type `X(a)` to make an element of $X$ from $a$. For example, if $X$ is the finite field of order 7, and $a = 2/5$ is a rational number, then `X(a)` is the finite field element 6 (as a quick exercise, check that this is mathematically the correct interpretation).

```
sage: X = GF(7); a = 2/5
sage: X(a)
6
```

As a slightly less trivial illustration of symbolic manipulation, consider the cubic equation

$$x^3 + \sqrt{2}x + 5 = 0. \qquad (1.1)$$

In Sage, we can create this equation, and find an exact symbolic solution.

```
sage: x = var('x')
sage: eqn =  x^3 + sqrt(2)*x + 5 == 0
sage: a = solve(eqn, x)[0].rhs()
```

The first line above makes sure that the symbolic variable $x$ is defined, the second creates the equation `eqn`, and the third line solves `eqn` for $x$, extracts the first solution (there are three), and takes the right hand side of that solution and assigns it to the variable `a`.

To see the solution nicely typeset, use the `show` command:

```
sage: show(a)
{{\left(...
```

$$\left(\frac{\sqrt{8\sqrt{2}+675}}{6\sqrt{3}}-\frac{5}{2}\right)^{\frac{1}{3}}\left(\frac{-\sqrt{3}i}{2}-\frac{1}{2}\right)-\frac{\sqrt{2}\left(\frac{\sqrt{3}i}{2}-\frac{1}{2}\right)}{3\left(\frac{\sqrt{8\sqrt{2}+675}}{6\sqrt{3}}-\frac{5}{2}\right)^{\frac{1}{3}}}$$

You can also see the latex needed to paste $a$ into a paper by typing `latex(a)`. The `latex` command works on most Sage objects.

```
sage: latex(a)
{{\left( \frac{\sqrt{ {8 \sqrt{ 2 }} ...
```

Next, we construct the number field obtained by adjoining the solution `a` to $\mathbb{Q}$. Notice that the minimal polynomial of the root is $x^6 + 10x^3 - 2x^2 + 25$.

```
sage: K.<b> = QQ[a]
sage: K
Number Field in a with defining
polynomial x^6 + 10*x^3 - 2*x^2 + 25
sage: a.minpoly()
x^6 + 10*x^3 - 2*x^2 + 25
sage: b.minpoly()
x^6 + 10*x^3 - 2*x^2 + 25
```

We can now compute interesting invariants of the number field $K$:

```
sage: K.class_number()
5
sage: K.galois_group().order()
72
```

## 1.2 Galois Groups

We can compute the Galois group of the Galois closure as an abstract "Pari group" using the `galois_group` function, which by default calls Pari (`http://pari.math.u-bordeaux.fr/`). You do not have to worry about installing Pari, since *Pari is part of Sage.* In fact, despite appearances much of the difficult algebraic number theory in Sage is actually done by the Pari C library (be sure to also cite Pari in papers that use Sage).

```
sage: K.<alpha> = NumberField(x^3 - 2)
sage: G = K.galois_group()
sage: G
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the
Number Field in alpha with defining polynomial x^3 - 2
```

We can find out more about $G$, too:

```
sage: G.order()
6
```

We compute two more Galois groups of degree 5 extensions, and see that one has Galois group $S_5$, so is not solvable by radicals:

```
sage: NumberField(x^5 - 2, 'a').galois_group()
Galois group PARI group [20, -1, 3, "F(5) = 5:4"] of
degree 5 of the Number Field in a with defining
polynomial x^5 - 2
sage: NumberField(x^5 - x + 2, 'a').galois_group()
Galois group PARI group [120, -1, 5, "S5"] of degree 5 of
the Number Field in a with defining polynomial x^5 - x + 2
```

Recent versions of Magma have an algorithm for computing Galois groups that in theory applies when the input polynomial has any degree. There are no open source implementation of this algorithm (as far as I know). If you have Magma, you can use this algorithm from Sage by calling the `galois_group` function and giving the `algorithm='magma'` option.

```
sage: K.<a> = NumberField(x^3 - 2)
sage: K.galois_group(algorithm='magma')    # optional
verbose...
Galois group Transitive group number 2 of degree 3 of
the Number Field in a with defining polynomial x^3 - 2
```

We emphasize that the above example should not work if you don't have Magma.

It is also possible to work explicitly with the group of automorphisms of a field (though the link in Sage between abstract groups and automorphisms of fields is currently poor[1]). For example, here we first define $\mathbb{Q}(\sqrt[3]{2})$, then compute its Galois closure, which we represent as $\mathbb{Q}(b)$, where $b^6 + 40b^3 + 1372 = 0$. Then we compute the automorphism group of the field $L$, and explicitly list its elements.

```
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b> = K.galois_closure()
sage: L
Number Field in b with defining polynomial x^6 + 40*x^3 + 1372
sage: G = Hom(L, L)
sage: G
Automorphism group of Number Field in b ...
sage: G.list()
[
Ring endomorphism of Number Field in b ...
  Defn: b |--> b,
Ring endomorphism of Number Field in b ...
```

```
  Defn: b |--> 1/36*b^4 + 1/18*b,
...
Ring endomorphism of Number Field in b ...
  Defn: b |--> -2/63*b^4 - 31/63*b
]
```

You can explicitly apply any of the automorphisms above to any elements of $L$.

```
sage: phi = G.list()[1]
sage: phi
Ring endomorphism of Number Field in b ...
  Defn: b |--> 1/36*b^4 + 1/18*b
sage: phi(b^2 + 2/3*b)
-1/36*b^5 + 1/54*b^4 - 19/18*b^2 + 1/27*b
```

You can also enumerate all complex embeddings of a number field:

```
sage: K.complex_embeddings()
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Double Field
  Defn: a |--> -0.629960524947 - 1.09112363597*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Double Field
  Defn: a |--> -0.629960524947 + 1.09112363597*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Double Field
  Defn: a |--> 1.25992104989
]
```

## 1.3 Class Numbers and Class Groups

The class group $C_K$ of a number field $K$ is the group of fractional ideals of the maximal order $R$ of $K$ modulo the subgroup of principal fractional ideals. One of the main theorems of algebraic number theory asserts that $C_K$ is a finite group. For example, the quadratic number field $\mathbb{Q}(\sqrt{-23})$ has class number 3, as we see using the Sage `class_number` command.

```
sage: L.<a> = NumberField(x^2 + 23)
sage: L.class_number()
3
```

There are only 9 quadratic imaginary field $\mathbb{Q}(\sqrt{D})$ that have class number 1:

$$D = -3, -4, -7, -8, -11, -19, -43, -67, -163.$$

To find this list using Sage, we first experiment with making lists in Sage. For example, typing `[1..10]` makes the list of integers between 1 and 10.

```
sage: [1..10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

We can also make the list of odd integers between 1 and 11, by typing `[1,3,..,11]`, i.e., by giving the second term in the arithmetic progression.

```
sage: [1,3,..,11]
[1, 3, 5, 7, 9, 11]
```

Applying this idea, we make the list of negative numbers from $-1$ down to $-10$.

```
sage: [-1,-2,..,-10]
[-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
```

The first two lines below makes a list $v$ of every $D$ from $-1$ down to $-200$ such that $D$ is a fundamental discriminant (the discriminant of a quadratic imaginary field). Note that you will not see the ... in the output below; this ... notation just means that part of the output is omitted below.

```
sage: w = [-1,-2,..,-200]
sage: v = [D for D in w if is_fundamental_discriminant(D)]
sage: v
[-3, -4, -7, -8, -11, -15, -19, -20, ..., -195, -199]
```

Finally, we make the list of $D$ in our list $v$ such that the quadratic number field $\mathbb{Q}(\sqrt{D})$ has class number 1. Notice that `QuadraticField(D)` is a shorthand for `NumberField(x^2 - D)`.

```
sage: [D for D in v if QuadraticField(D,'a').class_number()==1]
[-3, -4, -7, -8, -11, -19, -43, -67, -163]
```

Of course, we have *not* proved that this is the list of all negative $D$ so that $\mathbb{Q}(\sqrt{D})$ has class number 1.

A frustrating open problem is to prove that there are infinitely many number fields with class number 1. It is quite easy to be convinced that this is probably true by computing a bunch of class numbers of real quadratic fields. For example, over 58 percent of the real quadratic number fields with discriminant $D < 1000$ have class number 1!

```
sage: w = [1..1000]
sage: v = [D for D in w if is_fundamental_discriminant(D)]
sage: len(v)
302
sage: len([D for D in v if QuadraticField(D,'a').class_number() == 1])
176
sage: 176.0/302
0.582781456953642
```

For more intuition about what is going on, read about the Cohen-Lenstra heuristics.

Sage can also compute class numbers of extensions of higher degree, within reason. Here we use the shorthand `CyclotomicField(n)` to create the number field $\mathbb{Q}(\zeta_n)$.

```
sage: CyclotomicField(7)
Cyclotomic Field of order 7 and degree 6
sage: for n in [2..15]: print n, CyclotomicField(n).class_number()
2 1
3 1
...
15 1
```

In the code above, the notation `for n in [2..15]: ...` means "do ... for $n$ equal to each of the integers $2, 3, 4, \dots, 15$."

**Exercise 1.1.** Compute what is omitted (replaced by ...) in the output of the previous example.

Computations of class numbers and class groups in Sage is done by the Pari C library, and *unlike in Pari*, by default Sage tells Pari *not to assume* any conjectures. This can make some commands vastly slower than they might be directly in Pari, which *does assume unproved conjectures* by default. Fortunately, it is easy to tell Sage to be more permissive and allow Pari to assume conjectures, either just for this one call or henceforth for all number field functions. For example, with `proof=False` it takes only a few seconds to verify, modulo the conjectures assumed by Pari, that the class number of $\mathbb{Q}(\zeta_{23})$ is 3.

```
sage: CyclotomicField(23).class_number(proof=False)
3
```

**Exercise 1.2.** What is the smallest $n$ such that $\mathbb{Q}(\zeta_n)$ has class number bigger than 1?

In addition to computing class numbers, Sage can also compute the group structure and generators for class groups. For example, the quadratic field $\mathbb{Q}(\sqrt{-30})$ has class group $C = (\mathbb{Z}/2\mathbb{Z})^{\oplus 2}$, with generators the ideal classes containing $(5, \sqrt{-30})$ and $(3, \sqrt{-30})$.

```
sage: K.<a> = QuadraticField(-30)
sage: C = K.class_group()
sage: C
Class group of order 4 with structure C2 x C2 of Number Field
in a with defining polynomial x^2 + 30
sage: category(C)
Category of groups
sage: C.gens()
[Fractional ideal class (5, a), Fractional ideal class (3, a)]
```

In Sage, the notation `C.i` means "the $i$th generator of the object $C$," where the generators are indexed by numbers $0, 1, 2, \ldots$. Below, when we write `C.0 * C.1`, this means "the product of the 0th and 1st generators of the class group $C$."

```
sage: K.<a> = QuadraticField(-30)
sage: C = K.class_group()
sage: C.0
Fractional ideal class (5, a)
sage: C.0.ideal()
Fractional ideal (5, a)
sage: I = C.0 * C.1
sage: I
Fractional ideal class (2, a)
```

Next we find that the class of the fractional ideal $(2, \sqrt{-30} + 4/3)$ is equal to the ideal class $I$.

```
sage: A = K.ideal([2, a+4/3])
sage: J = C(A)
sage: J
Fractional ideal class (2/3, 1/3*a)
sage: J == I
True
```

Unfortunately, there is currently no Sage function that writes a fractional ideal class in terms of the generators for the class group.

## 1.4 Orders in Number Fields

An *order* in a number field $K$ is a subring of $K$ whose rank over $\mathbb{Z}$ equals the degree of $K$. For example, if $K = \mathbb{Q}(\sqrt{-1})$, then $\mathbb{Z}[7i]$ is an order in $K$. A good first exercise is to prove that every element of an order is an algebraic integer.

```
sage: K.<I> = NumberField(x^2 + 1)
sage: R = K.order(7*I)
sage: R
Order in Number Field in I with defining polynomial x^2 + 1
sage: R.basis()
[1, 7*I]
```

Using the `discriminant` command, we compute the discriminant of this order:

```
sage: factor(R.discriminant())
-1 * 2^2 * 7^2
```

You can give any list of elements of the number field, and it will generate the smallest ring $R$ that contains them.

```
sage: K.<a> = NumberField(x^4 + 2)
sage: K.order([12*a^2, 4*a + 12]).basis()
[1, 4*a, 4*a^2, 16*a^3]
```

If $R$ isn't of rank equal to the degree of the number field (i.e., $R$ isn't an order), then you'll get an error message.

```
sage: K.order([a^2])
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```

We can also compute the maximal order, using the `maxima_order` command, which behind the scenes finds an integral basis using Pari's `nfbasis` command. For example, $\mathbb{Q}(\sqrt[4]{2})$ has maximal order $\mathbb{Z}[\sqrt[4]{2}]$, and if $\alpha$ is a root of $x^3 + x^2 - 2x + 8$, then $\mathbb{Q}(\alpha)$ has maximal order with $\mathbb{Z}$-basis

$$1, \frac{1}{2}a^2 + \frac{1}{2}a, a^2.$$

```
sage: K.<a> = NumberField(x^4 + 2)
sage: K.maximal_order().basis()
[1, a, a^2, a^3]
sage: L.<a> = NumberField(x^3 + x^2 - 2*x+8)
sage: L.maximal_order().basis()
[1, 1/2*a^2 + 1/2*a, a^2]
sage: L.maximal_order().basis()[1].minpoly()
x^3 - 2*x^2 + 3*x - 10
```

There is still much important functionality for computing with non-maximal orders that is missing in Sage. For example, there is no support at all in Sage for computing with modules over orders or with ideals in non-maximal orders.

```
sage: K.<a> = NumberField(x^3 + 2)
sage: R = K.order(3*a)
sage: R.ideal(5)
Traceback (most recent call last):
...
NotImplementedError: ideals of non-maximal orders not
yet supported.
```

## 1.5 Relative Extensions

A *relative number field* $L$ is a number field of the form $K(\alpha)$, where $K$ is a number field, and an *absolute number field* is a number field presented in the form $\mathbb{Q}(\alpha)$. By the primitive element theorem, any relative number field $K(\alpha)$ can be written as $\mathbb{Q}(\beta)$ for some $\beta \in L$. However, in practice it is often convenient to view $L$ as $K(\alpha)$. In Section 1.1 we constructed the number field $\mathbb{Q}(\sqrt{2})(\alpha)$, where $\alpha$ is a root of $x^3 + \sqrt{2}x + 5$, but *not* as a relative field—we obtained just the number field defined by a root of $x^6 + 10x^3 - 2x^2 + 25$.

To construct this number field as a relative number field, first we let $K$ be $\mathbb{Q}(\sqrt{2})$.

```
sage: K.<sqrt2> = QuadraticField(2)
```

Next we create the univariate polynomial ring $R = K[X]$. In Sage, we do this by typing `R.<X> = K[]`. Here `R.<X>` means "create the object $R$ with generator $X$" and `K[]` means a "polynomial ring over $K$", where the generator is named based on the afformentioned $X$ (to create a polynomial ring in two variables $X, Y$ simply replace `R.<X>` by `R.<X,Y>`).

```
sage: R.<X> = K[]
sage: R
Univariate Polynomial Ring in X over Number Field in sqrt2
with defining polynomial x^2 - 2
```

Now we can make a polynomial over the number field $K = \mathbb{Q}(\sqrt{2})$, and construct the extension of $K$ obtained by adjoining a root of that polynomial to $K$.

```
sage: L.<a> = K.extension(X^3 + sqrt2*X + 5)
sage: L
Number Field in a with defining polynomial X^3 + sqrt2*X + 5...
```

Finally, $L$ is the number field $\mathbb{Q}(\sqrt{2})(\alpha)$, where $\alpha$ is a root of $X^3 + \sqrt{2}\alpha + 5$. We can do now do arithmetic in this number field, and of course include $\sqrt{2}$ in expressions.

```
sage: a^3
(-sqrt2)*a - 5
sage: a^3 + sqrt2*a
-5
```

The relative number field $L$ also has numerous functions, many of which are by default relative. For example the `degree` function on $L$ returns the relative degree of $L$ over $K$; for the degree of $L$ over $\mathbb{Q}$ use the `absolute_degree` function.

```
sage: L.degree()
3
sage: L.absolute_degree()
6
```

Given any relative number field you can also an absolute number field that is isomorphic to it. Below we create $M = \mathbb{Q}(b)$, which is isomorphic to $L$, but is an absolute field over $\mathbb{Q}$.

```
sage: M.<b> = L.absolute_field()
sage: M
Number Field in b with defining
polynomial x^6 + 10*x^3 - 2*x^2 + 25
```

The `structure` function returns isomorphisms in both directions between $M$ and $L$.

```
sage: M.structure()
(Isomorphism from Number Field in b ...,
 Isomorphism from Number Field in a ...)
```

In Sage one can create arbitrary towers of relative number fields (unlike in Pari, where a relative extension must be a single extension of an absolute field).

```
sage: R.<X> = L[]
sage: Z.<b> = L.extension(X^3 - a)
sage: Z
Number Field in b with defining polynomial
X^3 + (-1)*a over its base field
sage: Z.absolute_degree()
18
```

**Exercise 1.3.** Construct the relative number field $L = K(\sqrt[3]{\sqrt{2}+\sqrt{3}})$, where $K = \mathbb{Q}(\sqrt{2}, \sqrt{3})$.

One shortcoming with relative extensions in Sage is that behind the scenes all arithmetic is done in terms of a single absolute defining polynomial, and in some cases this can be very slow (much slower than Magma). Perhaps this could be fixed by using Singular's multivariate polynomials modulo an appropriate ideal, since Singular polynomial arithmetic is extremely flast. Also, Sage has very little direct support for constructive class field theory, which is a major motivation for explicit computation with relative orders; it would be good to expose more of Pari's functionality in this regard.

## 2 A Birds Eye View

We now take a whirlwind tour of some of the number theoretical functionality of Sage. There is much that we won't cover here, but this should help give you a flavor for some of the number theoretic capabilities of Sage, much of which is unique to Sage.

### 2.1 Integer Factorization

Bill Hart's quadratic sieve is included with Sage. The quadratic sieve is the best algorithm for factoring numbers of the form $pq$ up to around 100 digits. It involves searching for relations, solving a linear algebra problem modulo 2, then factoring $n$ using a relation $x^2 \equiv y^2 \mod n$.

```
sage: qsieve(next_prime(2^90)*next_prime(2^91), time=True)   # not tested
([1237940039285380274899124357, 2475880078570760549798248507],
 '14.94user 0.53system 0:15.72elapsed 98%CPU (0avgtext+0avgdata 0maxresident)k')
```

Using *qsieve* is twice as fast as Sage's *general* factor command in this example. Note that Sage's general factor command does nothing but call Pari's factor C library function.

```
sage: time factor(next_prime(2^90)*next_prime(2^91))     # not tested
CPU times: user 28.71 s, sys: 0.28 s, total: 28.98 s
Wall time: 29.38 s
1237940039285380274899124357 * 2475880078570760549798248507
```

Obviously, Sage's `factor` command should not just call Pari, but nobody has gotten around to rewriting it yet.

Paul Zimmerman's GMP-ECM is included in Sage. The elliptic curve factorization (ECM) algorithm is the best algorithm for factoring numbers of the form $n = pm$, where $p$ is not "too big". ECM is an algorithm due to Hendrik Lenstra, which works by "pretending" that $n$ is prime, chosing a random elliptic curve over $\mathbb{Z}/n\mathbb{Z}$, and doing arithmetic on that curve—if something goes wrong when doing arithmetic, we factor $n$.

In the following example, GMP-ECM is over 10 times faster than Sage's generic factor function. Again, this emphasizes that Sage's generic `factor` command would benefit from a rewrite that uses GMP-ECM and qsieve.

```
sage: time ecm.factor(next_prime(2^40) * next_prime(2^300))    # not tested
CPU times: user 0.85 s, sys: 0.01 s, total: 0.86 s
Wall time: 1.73 s
[1099511627791,
 2037035976334486086268445688409378161051468393665936250636140449354381299763336706183397533
sage: time factor(next_prime(2^40) * next_prime(2^300))        # not tested
CPU times: user 23.82 s, sys: 0.04 s, total: 23.86 s
Wall time: 24.35 s
1099511627791 * 2037035976334486086268445688409378161051468393665936250636140449354381299763
```

## 2.2 Elliptic Curves

Cremona's databases of elliptic curves is part of Sage. The curves up to conductor 10,000 come standard with Sage, and an optional 75MB download gives all his tables up to conductor 130,000. Type `sage -i database_cremona_ellcurve-20071019` to automatically download and install this extended table.

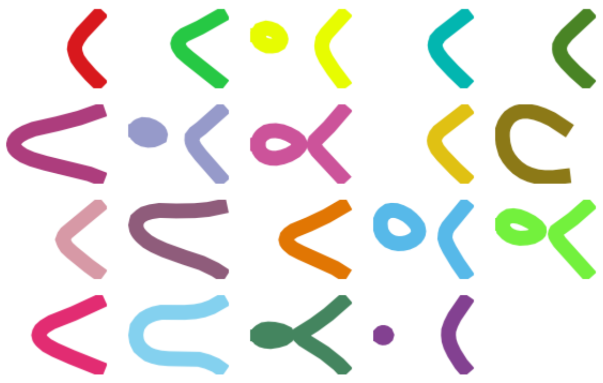
To use the database, just create a curve by giving

```
sage: EllipticCurve('5077a1')
Elliptic Curve defined by y^2 + y = x^3 - 7*x + 6 over Rational Field
sage: C = CremonaDatabase()
sage: C.number_of_curves()
847550
sage: C[37]
{'a': {'a1': [[0, 0, 1, -1, 0], 1, 1],
       'b1': [[0, 1, 1, -23, -50], 0, 3], ...
sage: C.isogeny_class('37b')
[Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50
over Rational Field, ...]
```

There is also a Stein-Watkins database that contains hundreds of millions of elliptic curves. It's over a 2GB download though!

Bryan Birch's recently had a birthday conference, and I used Sage to draw the cover of his birthday card by enumerating all optimal elliptic curves of conductor up to 37, then plotting them with thick randomly colored lines. As you can see below, plotting an elliptic curve is as simple as calling the plot method on it. Also, the `graphics_array` command allows us to easily combine numerous plots into a single graphics object.

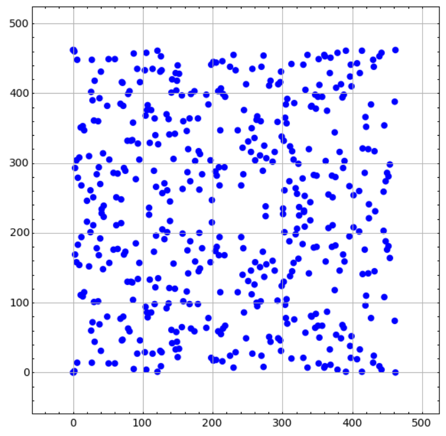
```
sage: v = cremona_optimal_curves([11..37])
sage: w = [E.plot(thickness=10,
   rgbcolor=(random(),random(),random())) for E in v]
sage: graphics_array(w, 4, 5).show(axes=False)
```



We can use Sage's interact feature to draw a plot of an elliptic curve modulo $p$, with a slider that one drags to change the prime $p$. The interact feature of Sage is very helpful for interactively changing parameters and viewing the results. Type `interact?` for more help and examples and visit the webpage `http://wiki.sagemath.org/interact`.

In the code below we first define the elliptic curve $E$ using the Cremona label `37a`. Then we define an interactive function $f$, which is made interactive using the `@interact` Python *decorator*. Because the default for $p$ is `primes(2,500)`, the Sage notebook constructs a slider that varies over the primes up to 500. When you drag the slider and let go, a plot is drawn of the affine $\mathbb{F}_p$ points on the curve $E_{\mathbb{F}_p}$. Of course, one *should* never plot curves over finite fields, which makes this even more fun.

```
E = EllipticCurve('37a')
@interact
def f(p=primes(2,500)):
    show(plot(E.change_ring(GF(p)),pointsize=30),
    axes=False, frame=True, gridlines="automatic",
    aspect_ratio=1, gridlinesstyle={'rgbcolor':(0.7,0.7,0.7)})
```



Sage includes `sea.gp`, which is a fast implementation of the SEA (Schoff-Elkies-Atkin) algorithm for counting the number of points on an elliptic curve over $\mathbb{F}_p$.

We create the finite field $k = \mathbb{F}_p$, where $p$ is the next prime after $10^{20}$. The `next_prime` command uses Pari's `nextprime` function, but proves primality of the result (unlike Pari which gives only the next probable prime after a number). Sage also has a `next_probable_prime` function.

```
sage: k = GF(next_prime(10^20))
```

compute its cardinality, which behind the scenes uses SEA.

```
sage: E = EllipticCurve(k.random_element())
sage: E.cardinality()                    # less than a second
100000000005466254167
```

To see how Sage chooses when to use SEA versus other methods, type `E.cardinality??` and read the source code. As of this writing, it simply uses SEA whenever $p > 10^{18}$.

Sage has the world's best code for computing $p$-adic regulators of elliptic curves, thanks to work of David Harvey and Robert Bradshaw. The $p$-adic regulator of an elliptic curve $E$ at a good ordinary prime $p$ is the determinant of the global $p$-adic height pairing matrix on the Mordell-Weil group $E(\mathbb{Q})$. (This has nothing to do with local or archimedean heights.) This is the analogue of the regulator in the Mazur-Tate-Teitelbaum $p$-adic analogue of the Birch and Swinnerton-Dyer conjecture.

In particular, Sage implements Harvey's improvement on an algorithm of Mazur-Stein-Tate, which builds on Kiran Kedlaya's Monsky-Washnitzer approach to computing $p$-adic cohomology groups.

We create the elliptic curve with Cremona label 389a, which is the curve of smallest conductor and rank 2. We then compute both the 5-adic and 997-adic regulators of this curve.

```
sage: E = EllipticCurve('389a')
sage: E.padic_regulator(5, 10)
5^2 + 2*5^3 + 2*5^4 + 4*5^5 + 3*5^6 + 4*5^7 + 3*5^8 + 5^9 + O(5^11)
sage: E.padic_regulator(997, 10)
740*997^2 + 916*997^3 + 472*997^4 + 325*997^5 + 697*997^6
          + 642*997^7 + 68*997^8 + 860*997^9 + 884*997^10 + O(997^11)
```

Before the new algorithm mentioned above, even computing a 7-adic regulator to 3 digits of precision was a nontrivial computational challenge. Now in Sage computing the 100003-adic regulator is routine:

```
sage: E.padic_regulator(100003,5)  # a couple of seconds
42582*100003^2 + 35250*100003^3 + 12790*100003^4 + 64078*100003^5 + O(100003^6)
```

$p$-adic $L$-functions play a central role in the arithmetic study of elliptic curves. They are $p$-adic analogues of complex analytic $L$-function, and their leading coefficient (at 0) is the analogue of $L^{(r)}(E,1)/\Omega_E$ in the $p$-adic analogue of the Birch and Swinnerton-Dyer conjecture. They also appear in theorems of Kato, Schneider, and others that prove partial results toward $p$-adic BSD using Iwasawa theory.

The implementation in Sage is mainly due to work of myself, Christian Wuthrich, and Robert Pollack. We use Sage to compute the 5-adic $L$-series of the elliptic curve 389a of rank 2.

```
sage: E = EllipticCurve('389a')
sage: L = E.padic_lseries(5)
sage: L
```

```
5-adic L-series of Elliptic Curve defined
by y^2 + y = x^3 + x^2 - 2*x over Rational Field
sage: L.series(3)
O(5^5) + O(5^2)*T + (4 + 4*5 + O(5^2))*T^2 +
(2 + 4*5 + O(5^2))*T^3 + (3 + O(5^2))*T^4 + O(T^5)
```

Sage implements code to compute numerous explicit bounds on Shafarevich-Tate Groups of elliptic curves. This functionality is *only* available in Sage, and uses results Kolyvagin, Kato, Perrin-Riou, etc., and unpublished papers of Wuthrich and me.

```
sage: E = EllipticCurve('11a1')
sage: E.sha().bound()           # so only 2,3,5 could divide sha
[2, 3, 5]
sage: E = EllipticCurve('37a1')  # so only 2 could divide sha
sage: E.sha().bound()
([2], 1)
sage: E = EllipticCurve('389a1')
sage: E.sha().bound()
(0, 0)
```

The $(0, 0)$ in the last output above indicates that the Euler systems results of Kolyvagin and Kato give no information about finiteness of the Shafarevich-Tate group of the curve $E$. In fact, it is an open problem to prove this finiteness, since $E$ has rank 2, and finiteness is only known for elliptic curves for which $L(E, 1) \neq 0$ or $L'(E, 1) \neq 0$.

Partial results of Kato, Schneider and others on the $p$-adic analogue of the BSD conjecture yield algorithms for bounding the $p$-part of the Shafarevich-Tate group. These algorithms require as input explicit computation of $p$-adic $L$-functions, $p$-adic regulators, etc., as explained in Stein-Wuthrich. For example, below we use Sage to prove that 5 and 7 do not divide the Shafarevich-Tate group of our rank 2 curve 389a.

```
sage: E = EllipticCurve('389a1')
sage: sha = E.sha()
sage: sha.p_primary_bound(5)  # iwasawa theory ==> 5 doesn't divide sha
0
sage: sha.p_primary_bound(7)  # iwasawa theory ==> 7 doesn't divide sha
0
```

This is consistent with the Birch and Swinnerton-Dyer conjecture, which predicts that the Shafarevich-Tate group is trivial. Below we compute this predicted order, which is the floating point number 1.000000 to some precision. That the result is a floating point number helps emphasize that it is an open problem to show that the *conjectural order* of the Shafarevich-Tate group is even a rational number in general!

```
sage: E.sha().an()
1.00000000000000
```

## 2.3 Mordell-Weil Groups and Integral Points

Sage includes both Cremona's mwrank library and Simon's 2-descent GP scripts for computing Mordell-Weil groups of elliptic curves.

```
sage: E = EllipticCurve([1,2,5,7,17])
sage: E.conductor()       # not in the Tables
154907
sage: E.gens()            # a few seconds
[(1 : 3 : 1), (67/4 : 507/8 : 1)]
```

Sage can also compute the torsion subgroup, isogeny class, determine images of Galois representations, determine reduction types, and includes a full implementation of Tate's algorithm over number fields.

Sage has the world's fastest implementation of computation of all integral points on an elliptic curve over $\mathbb{Q}$, due to work of Cremona, Michael Mardaus, and Tobias Nagel. This is also the only free open source implementation available.

```
sage: E = EllipticCurve([1,2,5,7,17])
sage: E.integral_points(both_signs=True)
[(1 : -9 : 1), (1 : 3 : 1)]
```

A very impressive example is the lowest conductor elliptic curve of rank 3, which has 36 integral points.

```
sage: E = elliptic_curves.rank(3)[0]
sage: E.integral_points(both_signs=True)   # less than 3 seconds
[(-3 : -1 : 1), (-3 : 0 : 1), (-2 : -4 : 1), (-2 : 3 : 1),
 ...(816 : -23310 : 1), (816 : 23309 : 1)]
```

The algorithm to compute all integral points involves first computing the Mordell-Weil group, then bounding the integral points, and listing all integral points satisfying those bounds. See Cohen's new GTM 239 for complete details.

The complexity grows exponentially in the rank of the curve. We can do the above calculation, but with the first known curve of rank 4, and it finishes in about a minute (and outputs 64 points).

```
sage: E = elliptic_curves.rank(4)[0]
sage: E.integral_points(both_signs=True)   # about a minute
[(-10 : 3 : 1), (-10 : 7 : 1), ...
 (19405 : -2712802 : 1), (19405 : 2693397 : 1)]
```

## 2.4 Elliptic Curve $L$-functions

We next compute with the complex $L$-function

$$L(E,s) = \prod_{p\nmid\Delta=389} \frac{1}{1-a_p p^{-s} + p p^{-2s}} \cdot \prod_{p\mid\Delta=389} \frac{1}{1-a_p p^{-s}}$$

of $E$. Though the above Euler product only defines an analytic function on the right half plane where $\mathrm{Re}(s) > 3/2$, a deep theorem of Wiles et al. (the **Modularity Theorem**) implies that it has an analytic continuation to the whole complex plane and functional equation. We can evaluate the function $L$ anywhere on the complex plane using Sage (via code of Tim Dokchitser).

```
sage: E = EllipticCurve('389a1')
sage: L = E.lseries()
sage: L
Complex L-series of the Elliptic Curve defined by
       y^2 + y = x^3 + x^2 - 2*x over Rational Field
sage: L(1)
-1.04124792770327e-19
sage: L(1+I)
-0.638409938588039 + 0.715495239204667*I
sage: L(100)
1.00000000000000
```

We can also compute the Taylor series of $L$ about *any* point, thanks to Tim Dokchitser's code.

```
sage: E = EllipticCurve('389a1')
sage: L = E.lseries()
sage: Ld = L.dokchitser()
sage: Ld.taylor_series(1,4)
-1.28158145691931e-23 + (7.26268290635587e-24)*z + 0.759316500288427*z^2
                      - 0.430302337583362*z^3 + O(z^4)
```

The Generalized Riemann Hypothesis asserts that all nontrivial zeros of $L(E, s)$ are of the form $1 + iy$. Mike Rubinstein has written a C++ program that is part of Sage that can for any $n$ compute the first $n$ values of $y$ such that $1 + iy$ is a zero of $L(E, s)$. It also verifies the Riemann Hypothesis for these zeros (I think). Rubinstein's program can also do similar computations for a wide class of $L$-functions, though not all of this functionality is as easy to use from Sage as for elliptic curves. Below we compute the first 10 zeros of $L(E, s)$, where $E$ is still the rank 2 curve 389a.

```
sage: L.zeros(10)
[0.000000000, 0.000000000, 2.87609907, 4.41689608, 5.79340263,
 6.98596665, 7.47490750, 8.63320525, 9.63307880, 10.3514333]
```

## 2.5 The Matrix of Frobenius on Hyperelliptic Curves

Sage has a highly optimized implementation of the Harvey-Kedlaya algorithm for computing the matrix of Frobenius associated to a curve over a finite field. This is an implementation by David Harvey, which is GPL'd and depends only on NTL and `zn_poly` (a C library in Sage for fast arithmetic $(\mathbb{Z}/n\mathbb{Z})[x]$).

We import the `hypellfrob` function and call it on a polynomial over $\mathbb{Z}$.

```
sage: from sage.schemes.hyperelliptic_curves.hypellfrob import hypellfrob
sage: R.<x> = PolynomialRing(ZZ)
sage: f = x^5 + 2*x^2 + x + 1; p = 101
sage: M = hypellfrob(p, 1, f); M
[ 0 + O(101)  0 + O(101) 93 + O(101) 62 + O(101)]
[ 0 + O(101)  0 + O(101) 55 + O(101) 19 + O(101)]
[ 0 + O(101)  0 + O(101) 65 + O(101) 42 + O(101)]
[ 0 + O(101)  0 + O(101) 89 + O(101) 29 + O(101)]
```

We do the same calculation but in $\mathbb{Z}/101^4\mathbb{Z}$, which gives enough precision to recognize the exact characteristic polynomial in $\mathbb{Z}[x]$ of Frobenius as an element of the endomorphism ring. This computation is still very fast, taking only a fraction of a second.

```
sage: M = hypellfrob(p, 4, f)   # about 0.25 seconds
sage: M[0,0]
91844754 + O(101^4)
```

The characteristic polynomial of Frobenius is $x^4 + 7x^3 + 167x^2 + 707x + 10201$, which determines the $\zeta$ function of the curve $y^2 = f(x)$.

```
sage: M.charpoly()
(1 + O(101^4))*x^4 + (7 + O(101^3))*x^3 + (167 + O(101^3))*x^2
   + (707 + O(101^3))*x + (10201 + O(101^4))
```

## 2.6 Modular Symbols

Modular symbols play a key role in algorithms for computing with modular forms, special values of $L$-functions, elliptic curves, and modular abelian varieties. Sage has the most general implementation of modular symbols available, thanks to work of myself, Jordi Quer (of Barcelona) and Craig Citro (a student of Hida). Moreover, computation with modular symbols is by far my *most favorite* part of computational mathematics. There is still a lot of tuning and optimization work to be done for modular symbols in Sage, in order for it to be across the board the fastest implementation in the world, since my Magma implementation is still better in some important cases.

We create the space $M$ of weight 4 modular symbols for a certain congruence subgroup $\Gamma_H(13)$ of level 13. Then we compute a basis for this space, expressed in terms of *Manin symbols*. Finally, we compute the Hecke operator $T_2$ acting on $M$, find its characteristic polynomial and factor it. We also compute the dimension of the cuspidal subspace.

```
sage: M = ModularSymbols(GammaH(13,[3]), weight=4)
sage: M
Modular Symbols space of dimension 14 for Congruence Subgroup
Gamma_H(13) with H generated by [3] of weight 4 with sign 0
and over Rational Field
sage: M.basis()
```

```
([X^2,(0,1)], [X^2,(0,7)], [X^2,(2,5)], [X^2,(2,8)], [X^2,(2,9)],
 [X^2,(2,10)], [X^2,(2,11)], [X^2,(2,12)], [X^2,(4,0)], [X^2,(4,3)],
 [X^2,(4,6)], [X^2,(4,8)], [X^2,(4,12)], [X^2,(7,1)])
sage: factor(charpoly(M.T(2)))
(x - 7) * (x + 7) * (x - 9)^2 * (x + 5)^2
        * (x^2 - x - 4)^2 * (x^2 + 9)^2
sage: dimension(M.cuspidal_subspace())
10
```

Sage includes John Cremona's specialized and *insanely fast* implementation of modular symbols for weight 2 and trivial character. We illustrate below computing the space of modular symbols of level 20014, which has dimension 5005, along with a Hecke operator on this space. The whole computation below takes only a few seconds; a similar computation takes a few minutes using Sage's generic modular symbols code. Moreover, Cremona has done computations at levels over 200,000 using his library, so the code is known to scale well to large problems. The new code in Sage for modular symbols is much more general, but doesn't scale nearly so well (yet).

```
sage: M = CremonaModularSymbols(20014)       # few seconds
sage: M
Cremona Modular Symbols space of dimension 5005 for
Gamma_0(20014) of weight 2 with sign 0
sage: t = M.hecke_matrix(3)            # few seconds
```

## 2.7 Enumerating Totally Real Number Fields

As part of his project to enumerate Shimura curves, John Voight has contributed code to Sage for enumerating totally real number fields. The algorithm isn't extremely complicated, but it involves some "inner loops" that have to be coded to run very quickly. Using *Cython*, Voight was able to implement exactly the variant of Newton iteration that he needed for his problem.

The function `enumerate_totallyreal_fields_prim(n, B, ...)` enumerates without using a database (!) primitive (no proper subfield) totally real fields of degree $n > 1$ with discriminant $d \leq B$.

We compute the totally real quadratic fields of discriminant $\leq 50$. The calculation below, which is almost instant, is done in real time and is not a table lookup.

```
sage: enumerate_totallyreal_fields_prim(2,50)
[[5, x^2 - x - 1], [8, x^2 - 2], [12, x^2 - 3], [13, x^2 - x - 3],
 [17, x^2 - x - 4], [21, x^2 - x - 5], [24, x^2 - 6], [28, x^2 - 7],
 [29, x^2 - x - 7], [33, x^2 - x - 8], [37, x^2 - x - 9],
 [40, x^2 - 10], [41, x^2 - x - 10], [44, x^2 - 11]]
```

We compute all totally real quintic fields of discriminant $\leq 10^5$. Again, this is done in real time – it's not a table lookup!

```
sage: enumerate_totallyreal_fields_prim(5,10^5)
[[14641, x^5 - x^4 - 4*x^3 + 3*x^2 + 3*x - 1],
 [24217, x^5 - 5*x^3 - x^2 + 3*x + 1],
 [36497, x^5 - 2*x^4 - 3*x^3 + 5*x^2 + x - 1],
 [38569, x^5 - 5*x^3 + 4*x - 1],
 [65657, x^5 - x^4 - 5*x^3 + 2*x^2 + 5*x + 1],
 [70601, x^5 - x^4 - 5*x^3 + 2*x^2 + 3*x - 1],
 [81509, x^5 - x^4 - 5*x^3 + 3*x^2 + 5*x - 2],
 [81589, x^5 - 6*x^3 + 8*x - 1],
 [89417, x^5 - 6*x^3 - x^2 + 8*x + 3]]
```

## 2.8 Bernoulli Numbers

From the mathematica website:

> "**Today We Broke the Bernoulli Record: From the Analytical Engine to Mathematica**
> April 29, 2008
> Oleksandr Pavlyk, Kernel Technology
> A week ago, I took our latest development version of Mathematica, and I typed `BernoulliB[10^7]`.
> And then I waited.
> Yesterday—5 days, 23 hours, 51 minutes, and 37 seconds later—I got the result!"

Tom Boothby did that same computation in Sage, which uses Pari's `bernfrac` command that uses evaluation of $\zeta$ and factorial to high precision, and it took 2 days, 12 hours.

Then David Harvey came up with an entirely new algorithm that parallelizes well. He gives these timings for computing $B_{10^7}$ on his machine (it takes 59 minutes, 57 seconds on my 16-core 1.8ghz Opteron box):

```
   PARI: 75 h, Mathematica: 142 h
   bernmm (1 core) = 11.1 h, bernmm (10 cores) = 1.3 h
```

> "Running on 10 cores for 5.5 days, I [David Harvey] computed [the Bernoulli number] $B_k$ for $k = 10^8$, which I believe is a new record. Essentially it's the multimodular algorithm I suggested earlier on this thread, but I figured out some tricks to optimise the crap out of the computation of $B_k \mathrm{mod} p$."

So now Sage is the fastest in the world for large Bernoulli numbers. The timings below are on a 16-core 1.8Ghz Opteron box.

```
sage: w = bernoulli(100000, num_threads=16)     # 1.87 seconds
sage: w = bernoulli(100000, algorithm='pari')   # 28 seconds
```

## 2.9 Polynomial Arithmetic

Sage uses Bill Hart and David Harvey's GPL'd Flint C library for arithmetic in $\mathbb{Z}[x]$. Its main claim to fame is that it is the world's fastest for polynomial multiplication, e.g., in the benchmark below it is 3 times faster than NTL and twice as fast as Magma. Behind the scenes it contains some carefully tuned discrete Fourier transform code (which I know nearly nothing about).

```
sage: Rflint = PolynomialRing(ZZ, 'x')
sage: f = Rflint([ZZ.random_element(2^64) for _ in [1..32]])
sage: g = Rflint([ZZ.random_element(2^64) for _ in [1..32]])
sage: timeit('f*g')               # random output
625 loops, best of 3: 105 microseconds per loop
sage: Rntl = PolynomialRing(ZZ, 'x', implementation='NTL')
sage: f = Rntl([ZZ.random_element(2^64) for _ in [1..32]])
sage: g = Rntl([ZZ.random_element(2^64) for _ in [1..32]])
sage: timeit('f*g')               # random output
625 loops, best of 3: 310 microseconds per loop
sage: ff = magma(f); gg = magma(g)
sage: s = 'time v := [%s * %s for _ in [1..10^5]];'%(ff.name(), gg.name())
sage: magma.eval(s)     # random output
'Time: 17.120'
sage: (17.120/10^5)*10^(6)    # convert to microseconds
171.200000000000
```

Multivariate polynomial arithmetic in many cases uses Singular in library mode (Martin Albrecht), which is quite fast. For example, below we do the Fateman benchmark over the finite field of order 32003.

```
sage: P.<x,y,z> = GF(32003)[]
sage: p = (x+y+z+1)^20
sage: q = p+1
sage: timeit('p*q')   # random output
5 loops, best of 3: 384 ms per loop
sage: pp = magma(p); qq = magma(q)
sage: s = 'time w := %s*%s;'%(pp.name(),qq.name())
sage: magma.eval(s)
'Time: 1.480'
```

Notice that the multiplication takes about four times as long in Magma.

# 3 Modular Forms

This section is about computing with modular forms, modular symbols, and modular abelian varieties. Most of the Sage functionality we describe below is new code written for Sage by myself, Craig Citro, Robert Bradshaw, and Jordi Quer in consultation with John Cremona. It has much overlap in functionality with the modular forms code in Magma, which I developed during 1998–2004.

## 3.1 Modular Forms and Hecke Operators

A *congruence subgroup* is a subgroup of the group $\mathrm{SL}_2(\mathbb{Z})$ of determinant $\pm 1$ integer matrices that contains

$$\Gamma(N) = \mathrm{Ker}(\mathrm{SL}_2(\mathbb{Z}) \to \mathrm{SL}_2(\mathbb{Z}/N\mathbb{Z}))$$

for some positive integer $N$. Since $\Gamma(N)$ has finite index in $\mathrm{SL}_2(\mathbb{Z})$, all congruence subgroups have finite index. The converse is *not* true, though in many other settings it is true (see [paper of Serre]).

The inverse image $\Gamma_0(N)$ of the subgroup of upper triangular matrices in $\mathrm{SL}_2(\mathbb{Z}/N\mathbb{Z})$ is a congruence subgroup, as is the inverse image $\Gamma_1(N)$ of the subgroup of matrices of the form $\left(\begin{smallmatrix} 1 & * \\ 0 & 1 \end{smallmatrix}\right)$. Also, for any subgroup $H \subset (\mathbb{Z}/N\mathbb{Z})^*$, the inverse image $\Gamma_H(N)$ of the subgroup of $\mathrm{SL}_2(\mathbb{Z}/N\mathbb{Z})$ of all elements of the form $\left(\begin{smallmatrix} a & * \\ 0 & d \end{smallmatrix}\right)$ with $d \in H$ is a congruence subgroup.

We can create each of the above congruence subgroups in Sage, using the `Gamma0`, `Gamma1`, and `GammaH` commands.

```
sage: Gamma0(8)
Congruence Subgroup Gamma0(8)
sage: Gamma1(13)
Congruence Subgroup Gamma1(13)
sage: GammaH(11,[2])
Congruence Subgroup Gamma_H(11) with H generated by [2]
```

The second argument to the `GammaH` command is a list of generators of the subgroup $H$ of $(\mathbb{Z}/N\mathbb{Z})^*$.
Sage can compute a list of generators for these subgroups. The algorithm Sage uses is a straightforward generic procedure that uses coset representatives for the congruence subgroup (which are easy to enumerate) to obtain a list of generators [[ref my modular forms book]].

```
sage: Gamma0(2).gens()
([1 1]
 [0 1],
 [-1  0]
 [ 0 -1],
 [ 1 -1]
 [ 0  1],
 [ 1 -1]
 [ 2 -1],
 [-1  1]
 [-2  1])
sage: len(Gamma1(13).gens())
284
```

As you can see above, the list of generators Sage computes is unfortunately large. Improving this would be an excellent Sage development project, which would involve much beautiful mathematics.

A *modular form* on a congruence subgroup $\Gamma$ of integer weight $k$ is a holomorphic function $f(z)$ on the upper half plane

$$\mathfrak{h}^* = \{z \in \mathbb{C} : \mathrm{Im}(z) > 0\} \cup \mathbb{Q} \cup \{i\infty\}$$

such that for every matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \Gamma$, we have

$$f\left(\frac{az+b}{cz+d}\right) = (cz+d)^k f(z). \tag{3.1}$$

A *cusp form* is a modular form that vanishes at all of the *cusps* $\mathbb{Q} \cup \{i\infty\}$.

If $\Gamma$ contains $\Gamma_1(N)$ for some $N$, then $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right) \in \Gamma$, so (3.1) implies that $f(z) = f(z+1)$. This, coupled with the holomorphicity condition, implies that $f(z)$ has a Fourier expansion

$$f(z) = \sum_{n=0}^{\infty} a_n e^{2\pi i n z},$$

with $a_n \in \mathbb{C}$. We let $q = e^{2\pi i z}$, and call $f = \sum_{n=0}^{\infty} a_n q^n$ the *q-expansion* of $f$.

Henceforth we assume that $\Gamma$ is either $\Gamma_1(N)$, $\Gamma_0(N)$, or $\Gamma_H(N)$ for some $H$ and $N$. The complex vector space $M_k(\Gamma)$ of all modular forms of weight $k$ on $\Gamma$ is a finite dimensional vector space.

We create the space $M_k(\Gamma)$ in Sage by typing `ModularForms(G, k)` where $G$ is the congruence subgroup and $k$ is the weight.

```
sage: ModularForms(Gamma0(25), 4)
Modular Forms space of dimension 11 for ...
sage: S = CuspForms(Gamma0(25),4, prec=15); S
Cuspidal subspace of dimension 5 of Modular Forms space ...
sage: S.basis()
[
q + q^9 - 8*q^11 - 8*q^14 + O(q^15),
q^2 - q^7 - q^8 - 7*q^12 + 7*q^13 + O(q^15),
q^3 + q^7 - 2*q^8 - 6*q^12 - 5*q^13 + O(q^15),
q^4 - q^6 - 3*q^9 + 5*q^11 - 2*q^14 + O(q^15),
q^5 - 4*q^10 + O(q^15)
]
```

Sage computes the dimensions of all these spaces using simple arithmetic formulas instead of actually computing bases for the spaces in question. In fact, Sage has the most general collection of modular forms dimension formulas of any software; type `help(sage.modular.dims)` to see a list of arithmetic functions that are used to implement these dimension formulas.

```
sage: ModularForms(Gamma1(949284), 456).dimension()
11156973844800
sage: a = [dimension_cusp_forms(Gamma0(N),2) for N in [1..25]]; a
[0, 0, ..., 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 2, 2, 1, 0]
```

```
sage: sloane_find(a)
Searching Sloane's online database...
[[1617,
  'Genus of modular group GAMMA_0 (n). Or, genus of
   modular curve X_0(n).',...
```

Sage doesn't have simple formulas for dimensions of spaces of modular forms of weight 1, since such formulas perhaps do not exist.

The space $M_k(\Gamma_1(N))$ is equipped with an action of $(\mathbb{Z}/N\mathbb{Z})^*$ by *diamond bracket operators* $\langle d \rangle$, and this induces a decomposition

$$M_k(\Gamma_1(N)) = \bigoplus_{\varepsilon:(\mathbb{Z}/N\mathbb{Z})^* \to \mathbb{C}^*} M_k(N, \varepsilon),$$

where the sum is over all complex characters of the finite abelian group $(\mathbb{Z}/N\mathbb{Z})^*$. These characters are called *Dirichlet characters*, which are central in number theory.

**Theorem 3.1.** *The space $M_k(\Gamma_1(N))$ has a basis of elements whose $q$-expansions $f(q)$ are all elements of $\mathbb{Z}[[q]]$.*

The factors $M_k(N, \varepsilon)$ then have bases whose $q$-expansions are elements of $R[[q]]$, where $R = \mathbb{Z}[\varepsilon]$ is the ring generated over $\mathbb{Z}$ by the image of $\varepsilon$. We illustrate this with $N = k = 5$ below, where `DirichletGroup` will be described later.

```
sage: CuspForms(DirichletGroup(5).0, 5).basis()
[q + (-zeta4 - 1)*q^2 + (6*zeta4 - 6)*q^3 - ... + O(q^6)]
```

Use the command `DirichletGroup(N,R)` to create the group of all Dirichlet characters of modulus $N$ taking values in the ring $R$. If $R$ is omited, it defaults to a cyclotomic field.

```
sage: G = DirichletGroup(8); G
Group of Dirichlet characters of modulus 8 over Cyclotomic
Field of order 2 and degree 1
sage: v = G.list(); v
[[1, 1], [-1, 1], [1, -1], [-1, -1]]
sage: eps = G.0; eps
[-1, 1]
sage: [eps(3), eps(5)]
[-1, 1]
```

Sage both represents Dirichlet characters by giving a "matrix", i.e., the list of images of canonical generators of $(\mathbb{Z}/N\mathbb{Z})^*$, and as vectors modulo and integer $n$. For years, I was torn between these two representations, until J. Quer and I realized that the best approach is to use both and make it easy to convert between them.

```
sage: parent(eps.element())
Vector space of dimension 2 over Ring of integers modulo 2
```

Given a Dirichlet character, Sage also lets you compute the associated Jacobi and Gauss sums, generalized Bernoulli numbers, the conductor, Galois orbit, etc.

Recall that Dirichlet characters give a decomposition

$$M_k(\Gamma_1(N)) = \bigoplus_{\varepsilon:(\mathbb{Z}/N\mathbb{Z})^*\to\mathbb{C}^*} M_k(N,\varepsilon).$$

Given a Dirichlet character $\varepsilon$ we type `ModularForms(eps, weight)` to create the space of modular forms with that character and a given integer weight. For example, we create the space of forms of weight 5 with the character modulo 8 above that is $-1$ on 3 and 1 on 5 as follows.

```
sage: ModularForms(eps,5)
Modular Forms space of dimension 6, character [-1, 1] and
weight 5 over Rational Field
sage: sum([ModularForms(eps,5).dimension() for eps in v])
11
sage: ModularForms(Gamma1(8),5)
Modular Forms space of dimension 11 ...
```

**Exercise 3.2.** Compute the dimensions of all spaces $M_2(37,\varepsilon)$ for all Dirichlet characters $\varepsilon$.

The space $M_k(\Gamma)$ is equipped with an action of a commuting ring $\mathbb{T}$ of Hecke operators $T_n$ for $n \geq 1$. A standard computational problem in the theory of modular forms is to compute an explicit basis of $q$-expansion for $M_k(\Gamma)$ along with matrices for the action of any Hecke operator $T_n$, and to compute the subspace $S_k(\Gamma)$ of cusp forms.

```
sage: M = ModularForms(Gamma0(11),4)
sage: M.basis()
[
q + 3*q^3 - 6*q^4 - 7*q^5 + O(q^6),
q^2 - 4*q^3 + 2*q^4 + 8*q^5 + O(q^6),
1 + O(q^6),
q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6)
]
sage: M.hecke_matrix(2)
[0 2 0 0]
[1 2 0 0]
[0 0 9 0]
[0 0 0 9]
```

We can also compute Hecke operators on the cuspidal subspace.

```
sage: S = M.cuspidal_subspace()
sage: S.hecke_matrix(2)
[0 2]
[1 2]
sage: S.hecke_matrix(3)
[ 3 -8]
[-4 -5]
```

Unfortunately, Sage doesn't yet implement computation of the Hecke operators on $M_k(\Gamma_1(N))$.

```
sage: M = ModularForms(Gamma1(5),2)
sage: M
Modular Forms space of dimension 3 for Congruence Subgroup
Gamma1(5) of weight 2 over Rational Field
sage: M.hecke_matrix(2)
Traceback (most recent call last):
...
NotImplementedError
```

However, we can compute Hecke operators on *modular symbols* for $\Gamma_1(N)$, which is a $\mathbb{T}$-module that is isomorphic to $M_k(\Gamma_1(N))$ (see Section 3.2).

```
sage: ModularSymbols(Gamma1(5),2,sign=1).hecke_matrix(2)
[ 2  1  1]
[ 1  2 -1]
[ 0  0 -1]
```

## 3.2 Modular Symbols

Modular symbols are a beautiful piece of mathematics that was developed since the 1960s by Birch, Manin, Shokorov, Mazur, Merel, Cremona, and others. Not only are modular symbols a powerful computational tool as we will see, they have also been used to prove rationality results for special values of $L$-series, to construct $p$-adic $L$-series, and they play a key role in Merel's proof of the uniform boundedness theorem for torsion points on elliptic curves over number fields.

We view modular symbols as a remarkably flexible computational tool that provides a single uniform algorithm for computing $M_k(N, \varepsilon)$ for any $N, \varepsilon$ and $k \geq 2$. There are ways to use computation of those spaces to obtain explicit basis for spaces of weight 1 and half-integral weight, so in a sense modular symbols yield everything. There are also generalizations of modular symbols to higher rank groups, though Sage currently has no code for modular symbols on higher rank groups.

A *modular symbol* of weight $k$, and level $N$, with character $\varepsilon$ is a sum of terms $X^iY^{k-2-i}\{\alpha, \beta\}$, where $0 \leq i \leq k-2$ and $\alpha, \beta \in \mathbb{P}^1(\mathbb{Q}) = \mathbb{Q} \cup \{\infty\}$. Modular symbols satisfy the relations

$$X^iY^{k-2-i}\{\alpha, \beta\} + X^iY^{k-2-i}\{\beta, \gamma\} + X^iY^{k-2-i}\{\gamma, \alpha\} = 0,$$

$$X^iY^{k-2-i}\{\alpha,\beta\} = -X^iY^{k-2-i}\{\beta,\alpha\},$$

and for every $\gamma = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in \Gamma_0(N)$, we have

$$(dX - bY)^i(-cX + aY)^{k-2-i}\{\gamma(\alpha), \gamma(\beta)\} = \varepsilon(d)X^iY^{k-2-i}\{\alpha, \beta\}.$$

The modular symbols space $\mathcal{M}_k(N,\varepsilon)$ is the torsion free $\mathbb{Q}[\varepsilon]$-module generated by all sums of modular symbols, modulo the relations listed above. Here $\mathbb{Q}[\varepsilon]$ is the ring generated by the values of the character $\varepsilon$, so it is of the form $\mathbb{Q}[\zeta_m]$ for some integer $m$.

The amazing theorem that makes modular symbols useful is that there is an explicit description of an action of a Hecke algebra $\mathbb{T}$ on $\mathcal{M}_k(N,\varepsilon)$, and there is an isomorphism

$$\mathcal{M}_k(N,\varepsilon;\mathbb{C}) \xrightarrow{\approx} M_k(N,\varepsilon) \oplus S_k(N,\varepsilon).$$

This means that if modular symbols are computable (they are!), then they can be used to compute a lot about the $\mathbb{T}$-module $M_k(N,\varepsilon)$.

Though $\mathcal{M}_k(N,\varepsilon)$ as described above is not explicitly generated by finitely many elements, it is finitely generated. Manin, Shokoruv, and Merel give an explicit description of finitely many generators (Manin symbols) for this space, along with all explicit relations that these generators satisfy (see my book). In particular, if we let

$$(i,c,d) = [X^iY^{2-k-i},(c,d)] = (dX - bY)^i(-cX + aY)^{k-2-i}\{\gamma(0),\gamma(\infty)\},$$

where $\gamma = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$, then the Manin symbols $(i,c,d)$ with $0 \le i \le k-2$ and $(c,d) \in \mathbb{P}^1(N)$ generate $\mathcal{M}_k(N,\varepsilon)$.

We compute a basis for the space of weight 4 modular symbols for $\Gamma_0(11)$, then coerce in $(2,0,1)$ and $(1,1,3)$.

```
sage: M = ModularSymbols(11,4)
sage: M.basis()
([X^2,(0,1)], [X^2,(1,6)], [X^2,(1,7)], [X^2,(1,8)],
 [X^2,(1,9)], [X^2,(1,10)])
sage: M( (2,0,1) )
[X^2,(0,1)]
sage: M( (1,1,3) )
2/7*[X^2,(1,6)] + 1/14*[X^2,(1,7)] - 4/7*[X^2,(1,8)]
                 + 3/14*[X^2,(1,10)]
```

We compute a modular symbols representation for the Manin symbol $(2,1,6)$, and verify this by converting back.

```
sage: a = M.1; a
[X^2,(1,6)]
sage: a.modular_symbol_rep()
36*X^2*{5/6,1} - 60*X*Y*{5/6,1} + 25*Y^2*{5/6,1}
sage: 36*M([2,5/6,1]) - 60*M([1,5/6,1]) + 25*M([0,5/6,1])
[X^2,(1,6)]
```

## 3.3 Method of Graphs

The Mestre Method of Graphs is an intriguing algorithm for computing the action of Hecke operators on yet another module $X$ that is isomorphic to $M_2(\Gamma_0(N))$. The implementation in Sage unfortunately only works when $N$ is prime; in contrast, my implementation in Magma works when $N = pM$ and $S_2(\Gamma_0(M)) = 0$.

The matrices of Hecke operators on $X$ are *vastly* sparser than on any basis of $M_2(\Gamma_0(N))$ that you are likely to use.

```
sage: X = SupersingularModule(389); X
Module of supersingular points on X_0(1)/F_389 over Integer Ring
sage: t2 = X.T(2).matrix(); t2[0]
(1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
sage: factor(charpoly(t2))
(x - 3) * (x + 2) * (x^2 - 2) * (x^3 - 4*x - 2) * ...
sage: t2 = ModularSymbols(389,sign=1).hecke_matrix(2); t2[0]
(3, 0, -1, 0, 0, -1, 1, 0, 0, 0, -1, 1, 0, 1, -1, 0, 1, 1,
 0, 1, -1, 1, -1, 1, 0, 0, 0, 0, 0, 0, 1, -1, -1)
sage: factor(charpoly(t2))
(x - 3) * (x + 2) * (x^2 - 2) * (x^3 - 4*x - 2) * ...
```

The method of graphs is also used in computer science to construct *expander graphs* with good properties. And it is important in my algorithm for computing Tamagawa numbers of purely toric modular abelian varieties. This algorithm is not implemented in Sage yet, since it is only interesting in the case of non-prime level, as it turns out.

## 3.4 Level One Modular Forms

The modular form

$$\Delta = q \prod (1 - q^n)^{24} = \sum \tau(n) q^n$$

is perhaps the world's most famous modular form. We compute some terms from the definition.

```
sage: R.<q> = QQ[[]]
sage: q * prod( 1-q^n+O(q^6) for n in (1..5) )^24
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 + O(q^7)
```

There are much better ways to compute $\Delta$, which amount to just a few polynomial multiplactions over $\mathbb{Z}$.

```
sage: D = delta_qexp(10^5)       # less than 10 seconds
sage: D[:10]
q - 24*q^2 + 252*q^3 - 1472*q^4 + ...
sage: [p for p in primes(10^5) if D[p] % p == 0]
```

```
[2, 3, 5, 7, 2411]
sage: D[2411]
4542041100095889012
sage: f = eisenstein_series_qexp(12,6) - D[:6]; f
691/65520 + 2073*q^2 + 176896*q^3 + 4197825*q^4 + 48823296*q^5 + O(q^6)
sage: f % 691
O(q^6)
```

The Victor Miller basis for $M_k(\mathrm{SL}_2(\mathbb{Z}))$ is the reduced row echelon basis. It's a lemma that it has all integer coefficients, and a rather nice diagonal shape.

```
sage: victor_miller_basis(24, 6)
[
1 + 52416000*q^3 + 39007332000*q^4 + 6609020221440*q^5 + O(q^6),
q + 195660*q^3 + 12080128*q^4 + 44656110*q^5 + O(q^6),
q^2 - 48*q^3 + 1080*q^4 - 15040*q^5 + O(q^6)
]
sage: dimension_modular_forms(1,200)
17
sage: time B = victor_miller_basis(200, 18)
CPU time: 4.43 s,  Wall time: 5.07 s
sage: B
[
1 + 79288314420681734048660707200000*q^17 + O(q^18),
q + 2687602718106772837928968846869*q^17 + O(q^18),
...
q^16 + 96*q^17 + O(q^18)
]
```

Note: Craig Citro has made the above computation an order of magnitude faster in code he hasn't quite got into Sage yet. "I'll clean those up and submit them soon, since I need them for something I'm working on ... I'm currently in the process of making spaces of modular forms of level one subclass the existing code, and actually take advantage of all our fast $E_k$ and $\Delta$ computation code, as well as cleaning things up a bit."

## 3.5 Half Integral Weight Forms

ALGORITHM: Basmaji (page 55 of his Essen thesis, "Ein Algorithmus zur Berechnung von Hecke-Operatoren und Anwendungen auf modulare Kurven", `http://wstein.org/scans/papers/basmaji/`).

Let $S = S_{k+1}(\varepsilon)$ be the space of cusp forms of even integer weight $k+1$ and character $\varepsilon = \chi\psi^{(k+1)/2}$, where $\psi$ is the nontrivial mod-4 Dirichlet character. Let $U$ be the subspace of $S \times S$ of elements $(a, b)$ such that $\Theta_2 a = \Theta_3 b$. Then $U$ is isomorphic to $S_{k/2}(\chi)$ via the map $(a, b) \mapsto a/\Theta_3$.

This algorithm is implemented in Sage. I'm sure it could be implemented in a way that is much faster than the current implementation...

```
sage: half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 3, 10)
[]
sage: half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 5, 10)
[q - 2*q^3 - 2*q^5 + 4*q^7 - q^9 + O(q^10)]
sage: half_integral_weight_modform_basis(DirichletGroup(16*7).0^2,3,30)
[q - 2*q^2 - q^9 + 2*q^14 + 6*q^18 - 2*q^21 - 4*q^22 - q^25 + O(q^30),
 q^2 - q^14 - 3*q^18 + 2*q^22 + O(q^30),
 q^4 - q^8 - q^16 + q^28 + O(q^30), q^7 - 2*q^15 + O(q^30)]
```

## 3.6 Generators for Rings of Modular Forms

For any congruence subgroup $\Gamma$, the direct sum

$$M(\Gamma) = \bigoplus_{k\geq 0} M_k(\Gamma)$$

is a ring, since the product of modular forms $f \in M_k(\Gamma)$ and $g \in M_{k'}(\Gamma)$ is an element $fg \in M_{k+k'}(\Gamma)$. Sage can compute likely generators for rings of modular forms, but currently doesn't prove any of these results.

We verify the statement proved in Serre's "A Course in Arithmetic" that $E_4$ and $E_6$ generate the space of level one modular forms.

```
sage: from sage.modular.modform.find_generators import modform_generators
sage: modform_generators(1)
[(4, 1 + 240*q + 2160*q^2 + 6720*q^3 + O(q^4)),
 (6, 1 - 504*q - 16632*q^2 - 122976*q^3 + O(q^4))]
```

Have you ever wondered which forms generate the ring $M(\Gamma_0(2))$? it turns out a form of weight 2 and two forms of weight 4 together generate.

```
sage: modform_generators(2)
[(2, 1 + 24*q + 24*q^2 + ... + 288*q^11 + O(q^12)),
 (4, 1 + 240*q^2 + .. + 30240*q^10 + O(q^12)),
 (4, q + 8*q^2 + .. + 1332*q^11 + O(q^12))]
```

Here's generators for $M(\Gamma_0(3))$. Notice that elements of weight 6 are now required, in addition to weights 2 and 4.

```
sage: modform_generators(3)
[(2, 1 + 12*q + 36*q^2 + .. + 168*q^13 + O(q^14)),
 (4, 1 + 240*q^3 + 2160*q^6 + 6720*q^9 + 17520*q^12 + O(q^14)),
 (4, q + 9*q^2 + 27*q^3 + 73*q^4 + .. + O(q^14)),
 (6, q - 6*q^2 + 9*q^3 + 4*q^4 + .. + O(q^14)),
 (6, 1 - 504*q^3 - 16632*q^6 .. + O(q^14)),
 (6, q + 33*q^2 + 243*q^3 + .. + O(q^14))]
```

## 3.7 $L$-series

Thanks to wrapping work of Jennifer Balakrishnan of M.I.T., we can compute explicitly with the $L$-series of the modular form $\Delta$. Like for elliptic curves, behind these scenes this uses Dokchitsers $L$-functions calculation Pari program.

```
sage: L = delta_lseries(); L
L-series associated to the modular form Delta
sage: L(1)
0.0374412812685155
```

In some cases we can also compute with $L$-series attached to a cusp form.

```
 sage: f = CuspForms(2,8).0
 sage: L = f.cuspform_lseries()
 sage: L(1)
 0.0884317737041015
 sage: L(0.5)
 0.0296568512531983
```

Unfortunately, computing with the $L$-series of a general newform is not yet implemented.

```
sage: S = CuspForms(23,2); S
Cuspidal subspace of dimension 2 of Modular Forms space of
dimension 3 for Congruence Subgroup Gamma0(23) of weight
2 over Rational Field
sage: f = S.newforms('a')[0]; f
q + a0*q^2 + (-2*a0 - 1)*q^3 + (-a0 - 1)*q^4 + 2*a0*q^5 + O(q^6)
```

Computing with $L(f, s)$ totally not implemented yet, though should be easy via Dokchitser.

## 3.8 Modular Abelian Varieties

The quotient of the extended upper half plane $\mathfrak{h}^*$ by the congruence subgroup $\Gamma_1(N)$ is the modular curve $X_1(N)$. Its Jacobian $J_1(N)$ is an abelian variety that is canonically defined over $\mathbb{Q}$. Likewise, one defines a modular abelian variety $J_0(N)$ associated to $\Gamma_0(N)$.

**Definition 3.3.** A *modular abelian variety* is an abelian variety over $\mathbb{Q}$ that is a quotient of $J_1(N)$ for some $N$.

The biggest recent theorem in number theory is the proof of Serre's conjecture by Khare and Wintenberger. According to an argument of Ribet and Serre, this implies the following modularity theorem, which generalizes the modularity theorem that Taylor-Wiles proved in the course of proving Fermat's Last Theorem.

**Theorem 3.4** (Modularity Theorem). *Let $A$ be a simple abelian variety defined over $\mathbb{Q}$. Then* $\mathrm{End}(A) \otimes \mathbb{Q}$ *is a number field of degree* $\dim(A)$ *if and only if $A$ is modular.*

One of my longterm research goals is to develop a systematic theory for computing with modular abelian varieties. A good start is the observation using the Abel-Jacobi theorem that every modular abelian variety (up to isomorphism) can be specified by giving a lattice in a space of modular symbols.

We define some modular abelian varieties of level 39, and compute some basic invariants.

```
sage: D = J0(39).decomposition(); D
[
Simple abelian subvariety 39a(1,39) of dimension 1 of J0(39),
Simple abelian subvariety 39b(1,39) of dimension 2 of J0(39)
]
sage: D[1].lattice()
Free module of degree 6 and rank 4 over Integer Ring
Echelon basis matrix:
[ 1  0  0  1 -1  0]
[ 0  1  1  0 -1  0]
[ 0  0  2  0 -1  0]
[ 0  0  0  0  0  1]
sage: G = D[1].rational_torsion_subgroup(); G
Torsion subgroup of Simple abelian subvariety 39b(1,39)
of dimension 2 of J0(39)
sage: G.order()
28
sage: G.gens()
[[(1/14, 2/7, 0, 1/14, -3/14, 1/7)], [(0, 1, 0, 0, -1/2, 0)],
 [(0, 0, 1, 0, -1/2, 0)]]
sage: B, phi = D[1]/G
sage: B
Abelian variety factor of dimension 2 of J0(39)
sage: phi.kernel()
(Finite subgroup with invariants [2, 14] ...
```

There is an algorithm in Sage for computing the exact endomorphism ring of any modular abelian variety.

```
sage: A = J0(91)[2]; A
Simple abelian subvariety 91c(1,91) of dimension 2 of J0(91)
sage: R = End(A); R
Endomorphism ring of Simple abelian subvariety 91c(1,91)
of dimension 2 of J0(91)
sage: for x in R.gens(): print x.matrix(),'\n'
[1 0 0 0]
```

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

[ 0  4 -2  0]
[-1  5 -2  1]
[-1  2  0  2]
[-1  1  0  3]
```

It is also possible to test isomorphism of two modular abelian varieties. But much exciting theoretical and computational work remains to be done.