Handbook of Magma Functions

Volume 11

Modular Forms

John Cannon Wieb Bosma
Claus Fieker Allan Steel
Editors

Version 2.22
Sydney
June 9, 2016

MAGMA
COMPUTER ALGEBRA

Handbook of Magma Functions

Editors:

John Cannon *Wieb Bosma* *Claus Fieker* *Allan Steel*

Handbook Contributors:

Geoff Bailey, Wieb Bosma, Gavin Brown, Nils Bruin, John Cannon, Jon Carlson, Scott Contini, Bruce Cox, Brendan Creutz, Steve Donnelly, Tim Dokchitser, Willem de Graaf, Andreas-Stephan Elsenhans, Claus Fieker, Damien Fisher, Volker Gebhardt, Sergei Haller, Michael Harrison, Florian Hess, Derek Holt, David Howden, Al Kasprzyk, Markus Kirschmer, David Kohel, Axel Kohnert, Dimitri Leemans, Paulette Lieby, Graham Matthews, Scott Murray, Eamonn O'Brien, Dan Roozemond, Ben Smith, Bernd Souvignier, William Stein, Allan Steel, Damien Stehlé, Nicole Sutherland, Don Taylor, Bill Unger, Alexa van der Waall, Paul van Wamelen, Helena Verrill, John Voight, Mark Watkins, Greg White

Production Editors:

Wieb Bosma *Claus Fieker* *Allan Steel* *Nicole Sutherland*

HTML Production:

Claus Fieker *Allan Steel*

VOLUME 11: OVERVIEW

VOLUME 11: CONTENTS

PART XVII
MODULAR ARITHMETIC GEOMETRY

134 MODULAR CURVES

Chapter 134
MODULAR CURVES

134.1 Introduction

Modular curves in MAGMA are a special type `CrvMod` of plane curve. A modular curve X is defined in terms of standard affine modular polynomials which are stored in precomputed databases. Those modular curves presently available are defined by bivariate polynomials relating the j-invariant and one of several standard functions on $X_0(N)$. These give singular models for $X_0(N)$ designed for computing isogenies of elliptic curves.

134.2 Creation Functions

Several different models for modular curves are available. The possible model types are "Atkin", "Canonical", and "Classical", each giving affine models defined by the modular polynomial databases of the same names. For more details on these polynomials see Section 134.4 on modular polynomials and databases.

134.2.1 Creation of a Modular Curve

`ModularCurve(X,t,N)`

Returns a model of the modular curve $X_0(N)$, in an affine plane specified by X. The string t must be one of "Atkin", "Canonical", or "Classical", with N a level in the corresponding modular curve database.

`ModularCurve(D, N)`

Returns an affine model of the modular curve $X_0(N)$ of level N from a database D of modular curves.

134.2.2 Creation of Points

Points on modular curves can be created in the same way as points on curves or schemes in general. In addition, there exist several specific constructors, defined in terms of the moduli structure, which take a parameterized elliptic curve as an argument.

`ModuliPoints(X,E)`

Given a modular curve $X = X_0(N)$ and an elliptic curve E, with compatible base rings, returns the sequence of points over the base field of E, corresponding to E with additional level structure.

Example H134E1

Below we give an example of the use of the moduli interpretation of modular curves in order to construct the corresponding subgroup scheme structures defined over a finite field.

```
> FF := FiniteField(NextPrime(10^6));
> A2 := AffineSpace(FF,2);
> X0 := ModularCurve(A2,"Canonical",17);
> E := EllipticCurve([FF|1,23]);
> mp := ModuliPoints(X0,E); mp;
[ (259805, 350390), (380571, 350390) ]
```

We will see later that it is possible to construct the structure of an elliptic curve parameterized by the corresponding moduli points.

```
> P, Q := Explode(mp);
> SubgroupScheme(E,P);
Subgroup of E defined by x^8 + 377217*x^7 + 190510*x^6 + 872850*x^5
  + 816054*x^4 + 457629*x^3 + 64955*x^2 + 361795*x + 460146
> SubgroupScheme(E,Q);
Subgroup of E defined by x^8 + 796070*x^7 + 308587*x^6 + 62023*x^5
  + 976430*x^4 + 380273*x^3 + 200328*x^2 + 892738*x + 536749
```

In this example we see that the prime 17 splits in the endomorphism ring of E, so we have exactly two parameterized isogenies defined over the base field of E.

134.3 Invariants

The defining data and invariants of the curves are accessed through standard functions.

`Level(X)`

Returns the level of the modular curve X as an integer.

`Genus(X)`

Returns the genus of the modular curve X .

`ModelType(X)`

Returns the type of the model for the modular curve X, presently limited to "Atkin", "Canonical", or "Classical". This is used to determine the algorithm by which parameterized isogenies are determined.

`Indices(X)`

Returns a sequence of integers, of the form $[N, M, P]$, classifying the class of the modular curve X. The integer sequence defines a congruence subgroup of $\mathrm{PSL}_2(\mathbf{Z})$ defined by

$$\Gamma(N, M, P) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \;\middle|\; c \equiv 0 \bmod N,\ b \equiv 0 \bmod P,\ a \equiv 1 \bmod M \right\},$$

where M divides $\mathrm{LCM}(N, P)$. For the models presently available, this will be the sequence $[N, 1, 1]$, where is the level of $X = X_0(N)$.

134.4 Modular Polynomial Databases

MAGMA contains several databases of standard defining polynomials for modular curves which are used throughout the system for the construction of isogenies of elliptic curves, and which are made available to the user. These define singular models for modular curves $X_0(N)$, in terms for standard functions on the curves.

The classical model for $X_0(N)$ is in terms of the polynomial $\Phi_N(X, Y)$ such that $\Phi_N(j(\tau), j(N\tau)) = 0$, where $j(\tau)$ is the j-function. The bivariate polynomial $\Phi_N(X, Y)$ is defined to be the *classical modular polynomial*. The classical modular polynomial has the property of being symmetric in X and Y, and moreover, the canonical involution is defined by $(X, Y) \mapsto (Y, X)$.

Suppose that N is a prime and set $s = 12/\text{GCD}(N-1, 12)$. Using the property that the Dedekind η-function is holomorphic with no zeros on the upper half plane $\mathbf{H}$, the function

$$f(\tau) = N^s \left(\frac{\eta(N\tau)}{\eta(\tau)} \right)^{2s},$$

is a holomorphic function whose reciprocal is also holomorphic on $\mathbf{H}$. The transformation properties of the η-function imply that $f(\tau)$ is invariant under $\Gamma_0(N)$. Together $j(\tau)$ and $f(\tau)$ generate the function field for $X_0(N)$. The polynomial $\Psi_N(X, Y)$ such that $\Psi_p(f(\tau), j(\tau)) = 0$ is called the *canonical modular polynomial*. The Atkin–Lehner involution sends $f(\tau)$ to $N^s/f(\tau)$, so that the relation $\Psi_N(N^s/f(\tau), j(N\tau)) = 0$ also holds.

The third distinguished class of modular polynomials are the *Atkin modular polynomials* $\Xi_N(X, Y)$, satisfying the property that $\Xi_N(f(\tau), j(\tau)) = 0$ for a modular function $f(\tau)$ on $X_0(N)$ invariant under the Atkin-Lehner involution. Since the function $f(\tau)$ is well-defined as a function on the Atkin-Lehner quotient curve $X_0^+(N)$, sometimes denoted $X_0^*(N)$, these polynomials are also referred to as *star* modular polynomials in the literature. The construction of $f(\tau)$ and the modular polynomial $\Xi_N(X, Y)$ is described in articles of Elkies [Elk98] and Morain [Mor95]. The database of Atkin modular polynomials were provided by A.O.L. Atkin.

`AtkinModularPolynomial(N)`

Given a prime number N, represented in the database of Atkin modular curves, this function returns the Atkin modular polynomial $\Xi_N(X, Y)$.

`CanonicalModularPolynomial(N)`

Given a prime number N represented in the database of canonical modular curves, this function returns the canonical modular polynomial $\Psi_N(X, Y)$.

`ClassicalModularPolynomial(N)`

Given an integer N represented in the database of classical modular curves, this function returns the defining classical modular polynomial $\Phi_N(X, Y)$.

`ModularCurveDatabase(t)`

`ModularCurveDatabase(t,i)`

Given an identifier string t, which must be one of "Atkin", "Canonical", or "Classical", this function returns the corresponding database object of modular curves $X_0(N)$.

Because of its size, the Atkin database is split into database objects of levels $200(i-1)+1 \leq N < 200i$ for i in $1, 2, 3, 4, 5$, of which only the first two are provided by default. Additional datafiles are available from the MAGMA website. The canonical database contains a subset of the curves for prime levels below 200, and only curves for levels below 60 are present in the classical modular curve database.

`N in D`

Returns `true` if and only if N is a level represented in the given modular curve database D.

`ExistsModularCurveDatabase(t)`

`ExistsModularCurveDatabase(t,i)`

Returns `true` if and only if the data file given by the string t and integer i exists

Example H134E2

Here we compare the defining polynomials for the Atkin, Canonical, and Classical models for the modular curves $X_0(N)$. For the modular curve $X_0(3)$ we list the corresponding polynomial.

```
> P2<x,j> := PolynomialRing(Integers(),2);
> P2!AtkinModularPolynomial(3);
x^4 - x^3*j + 744*x^3 + 193752*x^2 + 2348*x*j + 19712160*x + j^2
  + 24528*j + 538141968
> P2!CanonicalModularPolynomial(3);
x^4 + 36*x^3 + 270*x^2 - x*j + 756*x + 729
> P2!ClassicalModularPolynomial(3);
x^4 - x^3*j^3 + 2232*x^3*j^2 - 1069956*x^3*j + 36864000*x^3 +
    2232*x^2*j^3 + 2587918086*x^2*j^2 + 8900222976000*x^2*j +
    452984832000000*x^2 - 1069956*x*j^3 + 8900222976000*x*j^2 -
    770845966336000000*x*j + 1855425871872000000000*x + j^4 +
    36864000*j^3 + 452984832000000*j^2 + 1855425871872000000000*j
```

For larger values of N the Atkin modular polynomials tend to have smaller coefficients.

```
> P2!CanonicalModularPolynomial(11);
x^12 - 5940*x^11 + 14701434*x^10 - 139755*x^9*j - 19264518900*x^9 +
    723797800*x^8*j + 13849401061815*x^8 + 67496*x^7*j^2 -
    1327909897380*x^7*j - 4875351166521000*x^7 + 2291468355*x^6*j^2 +
    1036871615940600*x^6*j + 400050977713074380*x^6 - 5346*x^5*j^3 +
    4231762569540*x^5*j^2 - 310557763459301490*x^5*j +
    1224711544564336158000*x^5 + 161201040*x^4*j^3 +
    755793774757450*x^4*j^2 + 1730954664564250620 0*x^4*j +
    651339173406982403 1615*x^4 + 132*x^3*j^4 - 49836805205*x^3*j^3 +
```

```
    6941543075967060*x^3*j^2 - 6481517942976139866O*x^3*j +
    1042648844831301800367OO*x^3 + 468754*x^2*j^4 + 51801406800*x^2*j^3
    + 214437541826475*x^2*j^2 + 77380735840203400*x^2*j +
    804140494949359194*x^2 - x*j^5 + 3732*x*j^4 - 4586706*x*j^3 +
    2059075976*x*j^2 - 253478654715*x*j + 2067305393340*x + 1771561
> P2!CanonicalModularPolynomial(13);
x^14 + 26*x^13 + 325*x^12 + 2548*x^11 + 13832*x^10 + 54340*x^9 +
    157118*x^8 + 333580*x^7 + 509366*x^6 + 534820*x^5 + 354536*x^4 +
    124852*x^3 + 15145*x^2 - x*j + 746*x + 13
> P2!AtkinModularPolynomial(11);
x^12 - x^11*j + 744*x^11 + 196680*x^10 + 187*x^9*j + 21354080*x^9 +
    506*x^8*j + 830467440*x^8 - 11440*x^7*j + 16875327744*x^7 -
    57442*x^6*j + 208564958976*x^6 + 184184*x^5*j + 1678582287360*x^5 +
    1675784*x^4*j + 9031525113600*x^4 + 1867712*x^3*j +
    32349979904000*x^3 - 8252640*x^2*j + 74246810880000*x^2 -
    19849600*x*j + 98997734400000*x + j^2 - 8720000*j + 58411072000000
> P2!AtkinModularPolynomial(13);
x^14 - x^13*j + 744*x^13 + 196716*x^12 + 156*x^11*j + 21377304*x^11 +
    364*x^10*j + 835688022*x^10 - 8502*x^9*j + 17348897592*x^9 -
    37596*x^8*j + 224269358092*x^8 + 149786*x^7*j + 1949972557416*x^7 +
    1161420*x^6*j + 11858099339697*x^6 + 700323*x^5*j +
    51262531538400*x^5 - 9614956*x^4*j + 157275877324800*x^4 -
    23669490*x^3*j + 335383326720000*x^3 - 5859360*x^2*j +
    473336381440000*x^2 + 32384000*x*j + 397934592000000*x + j^2 +
    24576000*j + 150994944000000
```

In general the Atkin modular polynomials have coefficients of a smaller size when $N \equiv 11 \bmod 12$ and are largest when $N \equiv 1 \bmod 13$, while the opposite is true for the canonical modular polynomials.

134.5 Parametrized Structures

The modular curves $X_0(N)$ parametrize elliptic curves together with the structure of a cyclic isogeny, or equivalently, a cyclic subgroup scheme of the N-torsion of the elliptic curve. over the modular curve, singularities of the chosen surface may obstruct the construction of the corresponding isogeny.

`Isogeny(E,P)`

Given an elliptic curve E and a point P on some $X_0(N)$ corresponding to a cyclic level structure on E, the function returns an isogeny $f : E \to F$ corresponding to P. The isogeny is defined by the formulae of Velu in such a way that the pull-back of the invariant differential of F is the invariant differential on E.

`SubgroupScheme(E,P)`

Given an elliptic curve E and a point P on some $X_0(N)$ corresponding to a cyclic level structure on E, the function returns the subgroup scheme of E parameterized by $X_0(N)$.

Example H134E3

In this example we create the function field of the modular curve and compute the corresponding parameterized subgroup scheme for the canonical model of the modular curve $X_0(7)$.

```
> A2 := AffineSpace(RationalField(),2);
> X0 := ModularCurve(A2,"Canonical",7);
> K0<u,j> := FunctionField(X0);
> j;
(u^8 + 28*u^7 + 322*u^6 + 1904*u^5 + 5915*u^4 + 8624*u^3 + 4018*u^2 +
   748*u + 49)/u
```

We can now create an elliptic curve with this j-invariant and compute the corresponding subgroup scheme.

```
> E := EllipticCurveFromjInvariant(j);
> E;
Elliptic Curve defined by y^2 + x*y = x^3 - 36*u/(u^8 + 28*u^7 + 322*u^6 +
 1904*u^5 + 5915*u^4 + 8624*u^3 + 4018*u^2 - 980*u + 49)*x - u/(u^8 +
 28*u^7 + 322*u^6 + 1904*u^5 + 5915*u^4 + 8624*u^3 + 4018*u^2 - 980*u + 49)
 over Function Field of Modular Curve over Rational Field defined by
 $.1^8 + 28*$.1^7*$.3 + 322*$.1^6*$.3^2 + 1904*$.1^5*$.3^3 +
 5915*$.1^4*$.3^4 + 8624*$.1^3*$.3^5 + 4018*$.1^2*$.3^6 - $.1*$.2*$.3^6 +
 748*$.1*$.3^7 + 49*$.3^8
> ModuliPoints(X0,E);
[ (u, (u^8 + 28*u^7 + 322*u^6 + 1904*u^5 + 5915*u^4 + 8624*u^3 +
  4018*u^2 + 748*u + 49)/u) ]
> P := $1[1];
> SubgroupScheme(E,P);
Subgroup of E defined by x^3 + (-u^3 - 13*u^2 - 47*u - 14)/(u^4 + 14*u^3 +
 63*u^2 + 70*u - 7)*x^2 + (u^5 + 19*u^4 + 133*u^3 + 373*u^2 + 271*u + 49)/
 (u^8 + 28*u^7 + 322*u^6 + 1904*u^5 + 5915*u^4 + 8624*u^3 + 4018*u^2 -
 980*u + 49)*x + (-u^6 - 21*u^5 - 166*u^4 - 569*u^3 - 750*u^2 - 349*u -
 49)/(u^12 + 42*u^11 + 777*u^10 + 8246*u^9 + 54810*u^8 + 233730*u^7 +
 628425*u^6 + 999306*u^5 + 801738*u^4 + 159838*u^3 - 93639*u^2 +
 10290*u - 343)
```

We now replay the same computation for the Atkin model for $X_0(7)$.

```
> X0 := ModularCurve(A2,"Atkin",7);
> K0<u,j> := FunctionField(X0);
> j;
j
> E := EllipticCurveFromjInvariant(j);
> E;
Elliptic Curve defined by y^2 + x*y = x^3 + (36/(u^8 - 984*u^7 + 196476*u^6 +
 21843416*u^5 + 805505190*u^4 + 14493138072*u^3 + 138563855164*u^2 +
 677923505640*u + 1338887352609)*j + (-36*u^7 + 12852*u^5 + 51408*u^4 -
 1139292*u^3 - 7372512*u^2 + 6730380*u + 76688208)/(u^8 - 984*u^7 +
 196476*u^6 + 21843416*u^5 + 805505190*u^4 + 14493138072*u^3 +
 138563855164*u^2 + 677923505640*u + 1338887352609))*x + (1/(u^8 - 984*u^7 +
```

```
 196476*u^6 + 21843416*u^5 + 805505190*u^4 + 14493138072*u^3 +
 138563855164*u^2 + 677923505640*u + 1338887352609)*j + (-u^7 + 357*u^5 +
 1428*u^4 - 31647*u^3 - 204792*u^2 + 186955*u + 2130228)/(u^8 - 984*u^7 +
 196476*u^6 + 21843416*u^5 + 805505190*u^4 + 14493138072*u^3 +
 138563855164*u^2 + 677923505640*u + 1338887352609)) over
 Function Field of Modular Curve over Rational Field defined by
 $.1^8 - $.1^7*$.2 + 744*$.1^7*$.3 + 196476*$.1^6*$.3^2 +
 357*$.1^5*$.2*$.3^2 + 21226520*$.1^5*$.3^3 + 1428*$.1^4*$.2*$.3^3 +
 803037606*$.1^4*$.3^4 - 31647*$.1^3*$.2*$.3^4 + 14547824088*$.1^3*$.3^5 -
 204792*$.1^2*$.2*$.3^5 + 138917735740*$.1^2*$.3^6 + 186955*$.1*$.2*$.3^6 +
 $.2^2*$.3^6 + 677600447400*$.1*$.3^7 + 2128500*$.2*$.3^7 + 1335206318625*$.3^8
> P := X0![u,j];
> SubgroupScheme(E,P);
Subgroup of E defined by x^3 + ((-2*u^2 - 316*u - 2906)/(u^9 - 497*u^8 -
 20532*u^7 - 81388*u^6 + 4746950*u^5 + 56167290*u^4 - 1279028*u^3 -
 2650748588*u^2 - 11224313439*u - 8626202865)*j + (2*u^9 + 315*u^8 +
 2686*u^7 - 93700*u^6 - 1255594*u^5 + 3135966*u^4 + 104728146*u^3 +
 352853636*u^2 - 681445096*u - 3227101785)/(u^9 - 497*u^8 - 20532*u^7 -
 81388*u^6 + 4746950*u^5 + 56167290*u^4 - 1279028*u^3 - 2650748588*u^2 -
 11224313439*u - 8626202865))*x^2 + ((-u^6 - 262*u^5 - 16695*u^4 - 248404*u^3 -
 457567*u^2 + 11521914*u + 54297783)/(u^13 - 989*u^12 + 201198*u^11 +
 21056034*u^10 + 657230331*u^9 + 6165550233*u^8 - 88238124492*u^7 -
 2578695909108*u^6 - 20624257862361*u^5 + 1318238025445*u^4 +
 1038081350842750*u^3 + 6551865190346034*u^2 + 15514646620480317*u +
 9981405213700095)*j + (u^13 + 262*u^12 + 16338*u^11 + 153443*u^10 -
 5846023*u^9 - 115347903*u^8 + 30945748*u^7 + 15440847094*u^6 +
 109278156555*u^5 - 206898429120*u^4 - 5031013591446*u^3 -
 17024110451577*u^2 - 2556327655701*u + 47383402701465)/(u^13 - 989*u^12 +
 201198*u^11 + 21056034*u^10 + 657230331*u^9 + 6165550233*u^8 -
 88238124492*u^7 - 2578695909108*u^6 - 20624257862361*u^5 + 1318238025445*u^4 +
 1038081350842750*u^3 + 6551865190346034*u^2 + 15514646620480317*u +
 9981405213700095))*x + (-u^10 - 338*u^9 - 33893*u^8 - 1121560*u^7 -
 8257018*u^6 + 187293764*u^5 + 3504845638*u^4 + 15046974856*u^3 -
 62184511493*u^2 - 623227561058*u - 1183475711457)/(7*u^17 - 10367*u^16 +
 4654944*u^15 - 389781392*u^14 - 97999195364*u^13 - 5984563052076*u^12 -
 171524908893072*u^11 - 2216173007598816*u^10 + 4849421263003170*u^9 +
 61349830231624030*u^8 + 929601834094601 2480*u^7 + 5943878355618241 6176*u^6 -
 237426233800123901 96*u^5 - 323811129579449249 9900*u^4 -
 243531549841757418195 36*u^3 - 864749171915268570 48384*u^2 -
 146131978942592594496657*u - 808465974189161471 73495)*j +
 (u^17 + 338*u^16 + 33536*u^15 + 999466*u^14 - 4293800*u^13 - 625188410*u^12 -
 6912544240*u^11 + 82395591738*u^10 + 2064805256370*u^9 + 6482044820554*u^8 -
 152652278669056*u^7 - 1482689798210194*u^6 - 1214725952426000*u^5 +
 43848981757984690*u^4 + 215958476310275824*u^3 + 192371928062911406*u^2 -
 828594020518663419*u - 1409818017397793940)/(7*u^17 - 10367*u^16 + 4654944*u^15 -
 389781392*u^14 - 97999195364*u^13 - 5984563052076*u^12 - 171524908893072*u^11 -
 2216173007598816*u^10 + 4849421263003170*u^9 + 61349830231624030*u^8 +
 929601834094601 2480*u^7 + 5943878355618241 6176*u^6 - 237426233800123901 96*u^5 -
```

```
32381112957944924999000*u^4 - 24353154984175741819536*u^3 -
8647491719152685704838 4*u^2 - 14613197894259259449 6657*u -
8084659741891614717 3495)
```

134.6 Associated Structures

`FunctionField(X)`

Returns the function field of the modular curve X.

`jFunction(X)`

Given a modular curve X over a field, returns the j-invariant as a function on the curve.

`BaseCurve(X)`

Given one of the standard models X for the modular curve $X_0(N)$, returns the base model curve $X(1)$ and the morphism $\pi : X_0(N) \to X(1)$.

Example H134E4

In this example we demonstrate the relation of a modular curve with its base curve $X(1)$.

```
> D := ModularCurveDatabase("Atkin");
> X0 := ModularCurve(D,17);
> X1, pi := BaseCurve(X0);
```

The discriminants -4, -8, -16, -19, -43, and -67 are the class number 1 discriminants in which 17 is a split prime. We use this to construct the corresponding moduli points on the curve $X_0(17)$ and map these back down to the curve $X(1)$. Refer to Section 134.8 for a description of the function `HilbertClassPolynomial`.

```
> discs := [ -4, -8, -16, -19, -43, -67 ];
> jinvs := [ Roots(HilbertClassPolynomial(D))[1][1] : D in discs ];
> jinvs;
[ 1728, 8000, 287496, -884736, -884736000, -147197952000 ]
> pnts := &cat[ ModuliPoints(X0, EllipticCurveFromjInvariant(j))
>               : j in jinvs ];
> pnts;
[ (-2, 1728), (-3, 8000), (-4, 287496), (-1, -884736), (2, -884736000),
(7, -147197952000) ]
> [ pi(P) : P in pnts ];
[ (1728, 1728), (8000, 8000), (287496, 287496), (-884736, -884736),
(-884736000, -884736000), (-147197952000, -147197952000) ]
```

We note that $X(1)$ is defined to be the classical modular curve, defined by the diagonal image of the j-line in $\mathbf{P}^2$.

134.7 Automorphisms

`CanonicalInvolution(X)`

`AtkinLehnerInvolution(X,N)`

Given a projective modular curve $X = X_0(N)$, the function returns the Atkin-Lehner involution of the modular curve as a map of schemes. Currently, the only Atkin-Lehner involution returned is that for N equal to the level of X.

134.8 Class Polynomials

Class polynomials are invariants of elliptic curves with complex multiplication by an imaginary quadratic order of discriminant D. As such the Hilbert class polynomials can be interpreted as defining a subscheme or divisor on the modular curve $X(1) \cong \mathbf{P}^1$, while the Weber variants define a subscheme of a modular curve of higher level.

`HilbertClassPolynomial(D)`

Given a negative discriminant D, returns the Hilbert class polynomial, defined as the minimal polynomial of $j(\tau)$, where $\mathbf{Z}[\tau]$ is an imaginary quadratic order of discriminant D.

`WeberClassPolynomial(D)`

Given a negative discriminant D not congruent to 5 modulo 8, returns the Weber class polynomial, defined as the minimal polynomial of $f(\tau)$, where $\mathbf{Z}[\tau]$ is an imaginary quadratic order of discriminant D and f is a particular normalized Weber function generating the same class field as $j(\tau)$. The particular $f(\tau)$ used depends on whether D is odd ($\equiv 1 \bmod 8$) or even, in which case it also depends on $D/4 \bmod 8$, as well as whether 3 divides D or not. A root $f(\tau)$ of the Weber class polynomial is an algebraic integer (a unit in most cases and always a unit outside of 2) generating the ring class field related to the corresponding root $j(\tau)$ of the Hilbert class polynomial by an expression $j(\tau) = F(f(\tau))$. Here F is a rational function of the form $A(Bx^r + C)^3/x^r$ with $r|24$ and A,B,C rational integers which are positive or negative powers of 2. The function F is also returned. For example, if $D \equiv 1 \bmod 8$ then

$$j(\tau) = \frac{(f(\tau)^{24} - 16)^3}{f(\tau)^{24}},$$

where $\mathrm{GCD}(D, 3) = 1$, and

$$j(\tau) = \frac{(f(\tau)^{8} - 16)^3}{f(\tau)^{8}},$$

if 3 divides D and $f(\tau)$ is a unit.

In fact, $f(\tau)$ can only be a non-unit when $4|D$ and $D/4 \equiv 0, 4$ or $5 \bmod 8$. For further details, consult Yui and Zagier [YZ97] for the case of odd D and Schertz [Sch76] for the case of even D.

`WeberToHilbertClassPolynomial(f,D)`

`Al`	MonStgElt	*Default :* "*Roots*"

Given a negative discriminant D, and the corresponding Weber class polynomial f, returns the Hilbert class polynomial for D. The default algorithm, as specified by the `Al` parameter, is to compute complex approximations to the roots of the latter polynomial from approximations to the roots of the Weber polynomial and the rational function F linking the two. The other method is algebraic and uses resultants and the function F which was discussed in the description of the intrinsic `WeberClassPolynomial`.

Example H134E5

Class polynomials are typically used for constructing elliptic curves with a known endomorphism ring or known number of points over some finite field. The Weber (and other) variants of the class polynomials were introduced as a means of obtaining class invariants – defining the j-invariant of curves with given CM discriminant – with much smaller coefficients. In this example we give the classical example of $D = -71$, where the

```
> HilbertClassPolynomial(-71);
x^7 + 313645809715*x^6 - 3091990138604570*x^5 + 98394038810047812049302*x^4
- 82353426343973077996809138 9*x^3 + 513880036645397678032372632 9446*x^2 -
42531947394613960327460515118 7659*x + 73770708676073111335771424 1006081263
> WeberClassPolynomial(-71);
x^7 + x^6 - x^5 - x^4 - x^3 + x^2 + 2*x - 1
```

As indicated by the constant term -1, the roots of the WeberClassPolynomial are units in a particular ring class order.

134.9 Modular Curves and Quotients (Canonical Embeddings)

`ModularCurveQuotient(N,A)`

`Raw`	BoolElt	*Default :* `false`
`Reduce`	BoolElt	*Default :* `false`

Given a level N and a (possibly empty) set of Atkin-Lehner involutions represented as a sequence of integers A, this function computes a model for a quotient of $X_0(N)$ by A. When `Raw` is not set to `true`, an initial "semi-reduction" is not performed. If the `Reduce` option is `true`, then a complete `LLL`-reduction is performed on the resulting equations (this is impractical for genus greater than 50 or so). The returned curve can be: P^1 for a genus 0 curve, an elliptic or hyperelliptic curve, or the canonical embedding of the curve in P^{g-1} where g is the genus of the quotient curve (in this general case, the coordinates correspond to cusp forms invariant under the specified Atkin-Lehner involutions).

Example H134E6

We compute a model for $X_0(13 \cdot 29)$ quotiented by the Atkin-Lehner involutions w_{13} and w_{29}.

```
> C := X0NQuotient(13*29,[13,29]); C; // defined by cubics in P^4
Curve over Rational Field defined by
-x[1]*x[2]^2 + x[1]*x[2]*x[3] - x[2]^2*x[3] + x[1]*x[2]*x[4] +
    x[2]^2*x[4] + x[2]*x[4]^2 + x[1]*x[2]*x[5],
x[1]*x[2]^2 - x[2]^3 - x[2]^2*x[3] + x[1]*x[2]*x[5] + x[2]^2*x[5] +
    x[2]*x[4]*x[5],
x[1]*x[2]*x[3] - x[2]^2*x[3] - x[2]*x[3]^2 + x[1]*x[3]*x[5] +
    x[2]*x[3]*x[5] + x[3]*x[4]*x[5],
x[1]*x[2]*x[5] - x[2]^2*x[5] - x[2]*x[3]*x[5] + x[1]*x[5]^2 +
    x[2]*x[5]^2 + x[4]*x[5]^2,
-x[1]^2*x[2] + x[1]*x[2]^2 + x[1]*x[2]*x[3] - x[1]^2*x[5] -
    x[1]*x[2]*x[5] - x[1]*x[4]*x[5],
x[1]*x[2]*x[3] + x[2]^2*x[4] + x[2]*x[3]*x[4] + x[2]*x[4]*x[5] +
    x[3]*x[4]*x[5] + x[2]*x[5]^2 + x[4]*x[5]^2,
x[1]*x[2]*x[3] + x[1]*x[2]*x[4] + x[1]*x[2]*x[5] - x[1]*x[3]*x[5] -
    x[3]^2*x[5] + x[1]*x[4]*x[5] - x[2]*x[4]*x[5] - x[3]*x[4]*x[5] +
    x[1]*x[5]^2 + x[3]*x[5]^2,
-x[1]*x[2]*x[4] + x[1]*x[3]*x[4] - x[2]*x[3]*x[4] + x[1]*x[4]^2 +
    x[2]*x[4]^2 + x[4]^3 + x[1]*x[4]*x[5],
-x[1]*x[2]*x[3] + x[1]*x[3]^2 - x[2]*x[3]^2 + x[1]*x[3]*x[4] +
    x[2]*x[3]*x[4] + x[3]*x[4]^2 + x[1]*x[3]*x[5],
-x[1]*x[2]*x[5] + x[1]*x[3]*x[5] - x[2]*x[3]*x[5] + x[1]*x[4]*x[5] +
    x[2]*x[4]*x[5] + x[4]^2*x[5] + x[1]*x[5]^2,
x[1]*x[2]*x[4] - x[2]^2*x[4] - x[2]*x[3]*x[4] + x[1]*x[2]*x[5] -
    x[1]*x[3]*x[5] + x[2]*x[3]*x[5] - x[1]*x[5]^2,
-x[1]^2*x[3] - x[1]*x[2]*x[3] + x[1]*x[3]^2 - x[2]*x[3]^2 -
    x[1]*x[2]*x[4] - x[3]^2*x[4] - x[2]*x[4]^2 + x[1]*x[3]*x[5] -
    x[2]*x[4]*x[5] - x[4]^2*x[5] - x[1]*x[5]^2,
-x[1]^2*x[2] + x[1]*x[2]^2 + x[2]^2*x[3] + x[1]^2*x[4] -
    x[1]*x[3]*x[4] + x[2]*x[3]*x[4] + x[3]*x[4]^2 + x[1]^2*x[5] -
    x[1]*x[3]*x[5] - x[3]^2*x[5] - x[1]*x[4]*x[5] - x[3]*x[4]*x[5] +
    x[2]*x[5]^2 + x[3]*x[5]^2 + x[4]*x[5]^2,
-x[1]^2*x[2] + x[1]^2*x[3] - x[1]*x[2]*x[3] + x[1]^2*x[4] +
    x[1]*x[2]*x[4] + x[1]*x[4]^2 + x[1]^2*x[5],
x[1]^2*x[2] + x[1]^2*x[3] - x[1]*x[2]*x[3] + x[2]^2*x[3] - x[1]*x[3]^2
    - x[3]^3 - x[1]*x[2]*x[4] + x[1]*x[3]*x[4] - x[2]*x[3]*x[4] -
    x[3]^2*x[4] + x[1]^2*x[5] + x[1]*x[3]*x[5] - x[2]*x[3]*x[5] +
    x[3]^2*x[5] - x[3]*x[4]*x[5] + x[1]*x[5]^2
> Genus(C);
5
```

134.10 Modular Curves of Given Level and Genus

The intrinsics described in this section are designed to construct curves C over $\mathbf{Q}$ of genus at least 2 which are images of $X_1(N)$ (or $X_0(N)$) under a morphism $\pi : X_1(N) \to C$ defined over $\mathbf{Q}$. Such curves will be referred to as modular curves. The code uses some of the ideas and methods from the papers [GJG03], [BGJGP05] and [GJO10].

The morphism π induces an isogeny π^* from the Jacobian of C, $Jac(C)$ onto a $\mathbf{Q}$-rational abelian subvariety B of $J_1(N)$ (or $J_0(N)$), i.e. a $\mathbf{Q}$-rational modular abelian variety of dimension g, the genus of C. The space of holomorphic differentials of $Jac(C)$ identified with those of C pull back under π^* to the holomorphic differentials of B which can be identified with the space of weight 2 cusp forms associated to B.

If $f_1, \ldots, f_g$ is a $\mathbf{Q}$-basis of this space, then these forms will satisfy the canonical relations for some canonical embedding of C in the non-hyperelliptic case. Conversely, if a basis of the space of cusp forms for a $\mathbf{Q}$-rational modular abelian variety B of dimension g satisfies the polynomial relations defining a canonical curve C of genus g, then C is a modular curve, but the pullback of the space of holomorphic differentials under the obvious map $\pi : z \mapsto [f_1(z) : f_2(z) : \ldots : f_g(z)]$ from $X_1(N)$ to C may NOT give the space of differentials $\langle f_1, \ldots, f_g \rangle$ and $Jac(C)$ may be isogenous to a different modular abelian subvariety of $J_1(N)$ to B.

In the hyperelliptic case, criteria for the pullback of the differentials of the curve C to give precisely the space of forms of a particular modular abelian variety B are given in [GJG03] and [BGJGP05], provided that B lies in the new part of $J_1(N)$. In the genus 3 non-hyperelliptic case, a simple explicit criterion for $Jac(C)$ to pull back to B is given in [GJO10].

Modular curves C such that the pullback $\pi^*(Jac(C))$ lies in the *new* part of $J_1(N)$ (or $J_0(N)$) are referred to as *new* modular curves.

Intrinsics are provided that determine all of the new modular hyperelliptic curves or all of the new modular genus 3 non-hyperelliptic curves of $X_1(N)$ (or $X_0(N)$) for a given level N. There are also intrinsics that determine whether a given modular abelian subvariety B corresponds to a hyperelliptic or genus 3 non-hyperelliptic curve C and give the equations of C (as a Weierstrass model or canonical model, respectively) in the affirmative case.

In future releases it is planned to add intrinsics to deal with non-hyperelliptic modular curves of genus greater than 3 and to add more functionality to cover non-new cases.

`SetVerbose("ModularCurve", v)`

Set the printing level for verbose output for the following intrinsics. Currently the legal values for v are `true`, `false`, 0, 1, 2 (`false` is the same as 0, and `true` is the same as 1).

`NewModularHyperellipticCurves(N, g)`

`check`	BoolElt	*Default :* `false`
`prec`	RngIntElt	*Default :* 100
`gamma`	RngIntElt	*Default :* 1

For level N, this function returns a list of all hyperelliptic curves of genus $g \geq 2$ which are new modular curves for $X_1(N)$ if parameter `gamma` equals 1 (the default) or $X_0(N)$, if `gamma` equals 0.

A hyperelliptic curve is returned as a univariate polynomial $f(x)$ such that $y^2 = f(x)$ is a Weierstrass equation of the actual curve.

The parameter `prec` (default 100) is the precision to which modular forms are expanded in the computations. If parameter `check` (default `false`) is `true`, the forms are actually computed to a precision slightly in excess of the precision $prec_1$ needed to guarantee that polynomial relations on them (of the degrees that are used in the computations) vanish if they vanish to precision $prec_1$.

`NewModularHyperellipticCurve(B)`

`check`	BoolElt	*Default :* `false`
`prec`	RngIntElt	*Default :* 100
`gamma`	RngIntElt	*Default :* 1

Given a sequence B of distinct modular abelian subvarieties of $J_1(N)^{new}$, presented as subspaces of modular symbols, this function `true` if the modular abelian variety M which is the direct sum of the B corresponds exactly to a hyperelliptic curve C as described in the introduction (M is isogenous to $Jac(C)$). If so, a univariate polynomial $f(x)$ is also returned such that $y^2 = f(x)$ is a Weierstrass equation for C.

The parameters `check` and `prec` have the same meaning as in the previous intrinsic. The parameter `gamma` is only relevant if `check` is `true`, is 1 by default, and should only be set to 0 if all of the B are abelian subvarieties of $J_0(N)$. It is then used to get sharper bounds for the precision required for the q-expansions.

`NewModularHyperellipticCurve(F)`

This is a variant on the intrinsic directly above where, instead of the sequence of modular abelian subvarieties as argument, the sequence of q-expansions of the basis of weight 2 forms for the modular abelian subvariety M is given instead.

`ModularHyperellipticCurve(B)`

`prec`	RngIntElt	*Default :* 100

Given a sequence B of distinct modular abelian subvarieties of $J_1(N)^{new}$, where the direct sum modular abelian variety M that B defines does not have to lie in the new part $J_1(N)^{new}$, the function determines whether the basis of the differentials (forms) of M satisfy the correct relations so as to arise from a hyperelliptic curve C of genus g. If so, the intrinsic also returns a univariate polynomial $f(x)$, such that $y^2 = f(x)$ is a Weierstrass equation for C. The difference between this and the new case is that, although C is a modular curve, it is not guaranteed that $Jac(C)$ is isogenous to M: it may be isogenous to a different modular abelian subvariety of $J_1(N)$.

The parameter `prec` (default 100) is the precision to which modular forms are expanded in the computations.

`ModularHyperellipticCurve(F)`

This is a variant on the intrinsic immediately above where, instead of a sequence of modular abelian varieties B with direct sum M passed as the argument, a sequence of q-expansions of the basis of weight 2 cusp forms of such an M is given instead.

`NewModularNonHyperellipticCurvesGenus3(N)`

`check`	BoolElt	*Default :* `false`
`prec`	RngIntElt	*Default :* 100
`gamma`	RngIntElt	*Default :* 1

Given an integer N, this function returns a list of all non-hyperelliptic curves of genus 3 which are new modular curves, for $X_1(N)$ if the parameter `gamma` equals 1 (the default), or for $X_0(N)$ if `gamma` equals 0.

The parameter `prec` (default 100) is the precision to which modular forms are computed to in the computations. If parameter `check` (default `false`) is `true`, the forms are actually computed to a little over the precision $prec_1$ needed to check that polynomial relations on them (of the degrees that are used in the computations) are definitely guaranteed to vanish if they vanish to precision $prec_1$.

`NewModularNonHyperellipticCurveGenus3(B)`

`check`	BoolElt	*Default :* `false`
`prec`	RngIntElt	*Default :* 100
`gamma`	RngIntElt	*Default :* 1

Given a sequence B of distinct modular abelian subvarieties of $J_1(N)^{new}$, where the direct sum modular abelian variety M that B defines does not have to lie in the new part $J_1(N)^{new}$, the function determines whether the basis of the differentials (forms) of M satisfy the correct relations so as to arise from a non-hyperelliptic curve C of genus 3. If so, the intrinsic also returns a a defining polynomial for the canonical image of C. The parameters `check` and `prec` have the same meaning as for the intrinsic `NewModularHyperellipticCurves`. The parameter `gamma`, which is only relevant if `check` is `true`, and is 1 by default, may be set to 0 if M is an abelian subvariety of $J_0(N)$ when sharper bounds for the required precision of q-expansions will be used. This intrinsic is similar to `NewModularHyperellipticCurve(B)`.

`NewModularNonHyperellipticCurveGenus3(F)`

This is a variant on the intrinsic `NewModularNonHyperellipticCurveGenus3(B)` above where, instead of the sequence of modular abelian varieties B with direct sum M passed as the argument, a sequence F of q-expansions of the basis of weight 2 cusp forms of such an M is given instead.

`ModularNonHyperellipticCurveGenus3(F)`

This is the same as the intrinsic `NewModularNonHyperellipticCurveGenus3(F)` immediately preceding except that it is not required that the modular abelian variety M which corresponds to F lies in the new part $J_1(N)^{new}$. This may be used to search for non-new non-hyperelliptic genus 3 curves.

Example H134E7

We give some examples of the use of these intrinsics.
The modular curve $X_1(13)$ is of genus 2, therefore hyperelliptic, and the space of modular forms of weight 2 for $\Gamma_1(13)$ is generated by an unique newform f of nebentypus a Dirichlet character of order 6, such that the modular abelian variety attached to f, A_f, is $\mathbf{Q}$-isogenous to $J_1(13)$:

```
> chi := DirichletGroup(13,CyclotomicField(6)).1; //order 6 character mod 13
> A13:=Af(chi)[1];
> NewModularHyperellipticCurve([A13]);
true x^6 + 4*x^5 + 6*x^4 + 2*x^3 + x^2 + 2*x + 1
> f13:=qIntegralBasis(A13,100);
> NewModularHyperellipticCurve(f13);
true x^6 + 4*x^5 + 6*x^4 + 2*x^3 + x^2 + 2*x + 1
> SetVerbose("ModularCurve",1);
> NewModularHyperellipticCurve([A13]:check:=true);
Checking ...
          ... bound =113
true x^6 + 4*x^5 + 6*x^4 + 2*x^3 + x^2 + 2*x + 1
```

Let us compute all the new modular hyperelliptic curves parameterized by $X_0(80)$:

```
> NewModularHyperellipticCurves(80,0: gamma:=0);
     Candidates:=3
     All the curves
1 2 3
[
    x^5 + 2*x^4 - 26*x^3 - 132*x^2 - 231*x - 142
]
```

Finally, we calculate the new modular hyperelliptic curves parameterized by $X_1(80)$ (the default):

```
> SetVerbose("ModularCurve",0);
> NewModularHyperellipticCurves(80,0);
[
    x^5 + 2*x^4 - 26*x^3 - 132*x^2 - 231*x - 142,
    x^5 - 2*x^4 - 2*x^3 + 20*x^2 - 47*x + 30,
    x^7 - 4*x^6 - 4*x^5 + 39*x^4 - 64*x^3 + 40*x^2 - 8*x,
    x^7 + 2*x^5 + 7*x^4 - 4*x^3 - 20*x^2 - 16*x - 4
]
```

Example H134E8

In this exercise, an interesting example from [JK07] is presented.

```
> S:=CuspidalSubspace(ModularSymbolsH(21,[1,8,13,20],2,+1));
> S;
Modular symbols space of level 21, weight 2, and dimension 3 over Rational Field
(multi-character)
> ModularHyperellipticCurve([S]);
true x^8 - 6*x^6 + 4*x^5 + 11*x^4 - 24*x^3 + 22*x^2 - 8*x + 1
```

The above curve is the unique hyperelliptic intermediate modular curve $X_\Delta(N)$ between $X_1(N)$ and $X_0(N)$, where Δ is a subgroup of $(\mathbf{Z}/N\mathbf{Z})^*/\{\pm 1\}$ and $X_\Delta(N)$ is the modular curve associated to some congruence subgroup of $PSL_2(\mathbf{Z})$ attached to Δ. In fact, the above hyperelliptic curve is the new modular curve denoted by $C^A_{21A_{\{0,2\}}}$ in [BGJGP05].

Example H134E9

Given a modular abelian subvariety of $J_0(97)$, we determine whether it corresponds to a new modular non-hyperelliptic curve of genus 3 and level 97:

```
> M:=ModularSymbols(97,2,1);
> NN:=SortDecomposition(NewformDecomposition(NewSubspace(CuspidalSubspace(M))));
> NN;
[
    Modular symbols space for Gamma_0(97) of weight 2 and dimension 3 over
    Rational Field,
    Modular symbols space for Gamma_0(97) of weight 2 and dimension 4 over
    Rational Field
]
> A97:=NN[1];
> NewModularNonHyperellipticCurveGenus3([A97]);
true -x^2*y^2 + x*y^3 + x^3*z + x*y^2*z - 5*x^2*z^2 + 3*x*y*z^2 - 3*y^2*z^2 +
  6*x*z^3 - y*z^3 - 2*z^4
```

Example H134E10

We calculate all of the new modular non-hyperelliptic curves of genus 3 parameterized by $X_1(20)$:

```
> NewModularNonHyperellipticCurvesGenus3(20);
[
    -x^2*y^2 + x*y^3 + x^3*z - 3*x^2*z^2 + 4*x*z^3 - 2*z^4
]
```

We now construct an example of a non-new modular non-hyperelliptic curve of genus 3:

```
> S1:=ModularSymbols(178,2,1);
> N1:=SortDecomposition(NewformDecomposition(NewSubspace(CuspidalSubspace(S1))));
> A:=N1[3];A;
Modular symbols space for Gamma_0(178) of weight 2 and dimension 2 over
Rational Field
```

```
> S2:=ModularSymbols(89,2,1);
> N2:=SortDecomposition(NewformDecomposition(NewSubspace(CuspidalSubspace(S2))));
> B:=N2[1];B;
Modular symbols space for Gamma_0(89) of weight 2 and dimension 1 over
Rational Field
> fA:=qIntegralBasis(A,100);
> fB:=qIntegralBasis(B,100);
> q := Universe(fB).1;
> fB2:=&+[Coefficient(fB[1],k)*q^(2*k) : k in [1..99]];
> g:=fB[1]+2*fB2;
> ModularNonHyperellipticCurveGenus3([g,fA[1],fA[2]]);
true -x^4 + 2*x^2*y^2 - y^4 + 8*x^2*y*z + 8*y^3*z - 6*x^2*z^2 - 38*y^2*z^2 +
    24*y*z^3 + 7*z^4
>
```

The above curve is denoted by C_{178C}^{89A} in [GJO10]. Its jacobian is $\mathbf{Q}$-isogenous to $A \times B$.

134.11 Bibliography

[**BGJGP05**] M. Baker, E. Gonzalez-Jimenez, J. Gonzalez, and B. Poonen. Finiteness results for modular curves of genus at least 2. *Amer. J. Math.*, 127:1325–1387, 2005.

[**Elk98**] N. Elkies. Elliptic and modular curves over finite fields and related computational issues. In *Computational Perspectives on Number Theory A conference in honor of A.O.L. Atkin*, 1998.

[**GJG03**] E. Gonzalez-Jimenez and J. Gonzalez. Modular curves of genus 2. *Math. Comp.*, 72:397–418, 2003.

[**GJO10**] E. Gonzalez-Jimenez and Roger Oyono. Non-hyperelliptic modular curves of genus 3. *J. Number Th.*, 130:862–878, 2010.

[**JK07**] D. Jeon and C. H. Kim. On the arithmetic of certain modular curves. *Acta Arith.*, 130:181–193, 2007.

[**Mor95**] F. Morain. Calcul du nombre de points sur une courbe elliptique dans un corps fini: aspects algorithmiques. *J. Théorie des Nombres de Bordeaux*, 7:255–282, 1995.

[**Sch76**] R. Schertz. Die singularen Werte der Weberschen Funktionen $\mathbf{f}$,$\mathbf{f}_1$,$\mathbf{f}_2$,γ_2,γ_3. *J. reine angew. Math.*, 286/287:46–74, 1976.

[**YZ97**] N. Yui and D. Zagier. On the singular values of Weber modular functions. *Mathematics of Computation*, 66(220):1645–1662, 1997.

135 SMALL MODULAR CURVES

Chapter 135
SMALL MODULAR CURVES

135.1 Introduction

The Small Modular Curve database is a database of simple models over the rational numbers for the $X_0(N)$ modular curves along with the standard automorphisms, cusps and non-cuspidal rational points in the positive genus cases, the various projection maps from $X_0(N) \to X_0(M)$ when M divides N and expressions for standard functions $(j(z), j(Nz))$ and forms (e.g. E_4, E_6) in terms of rational functions or k-differentials on the model. We refer to them as "small", because the models are of low degree, are defined by reasonably sparse polynomials with small integer coefficients, and are non-singular or have only a few simple singularities.

Because the MAGMA type of the model may be `CrvEll`, `CrvHyp`, `CrvPln` or just general `Crv` and it is not possible for a type to extend different subtypes of `Crv`, we have not introduced a special type like `CrvMod` (or used this type) for small modular curves. The modular curve component of the arguments to most of the intrinsics consists of a projective curve, which should be a base change of the model over $\mathbf{Q}$ to a field of characteristic zero, and the level N. This is a little "clunky" but hopefully the user won't find it too awkward. For efficiency, there is currently no initial check that the curve argument really is a base change of the database model, which the user can obtain via the `SmallModularCurve` call. The call will almost certainly result in an error if this is not the case.

For simplicity, in the initial implementation, the intrinsics require the modular curve to be defined over a field of characteristic zero, although the models are all defined by polynomials with integral coefficients which reduce nicely for primes not dividing the level. Again, we do not check this condition in all intrinsics: the user may find that if he carries out a mod p reduction on the database model himself and uses it as the argument to some of the intrinsics, they may still work.

135.2 Small Modular Curve Models

The models are projective models of the complete curve $X_0(N)$ over $\mathbf{Q}$. In the genus 0 case, we simple take the projective line $\mathbf{P}^1$ with the point at infinity corresponding to the cusp at infinity ∞. In the genus 1 case, we take a `CrvEll` which is the standard minimal Weierstrass model of the elliptic curve $(X_0(N), \infty)$ with the cusp ∞ as the 0 point for the group law. In the hyperelliptic cases (see [Ogg74]), we take a minimal Weierstrass model with two rational points at infinity, the cusp ∞ (which is never a hyperelliptic Weierstrass point) and its image under the hyperelliptic involution.

In all other cases (non-subhyperelliptic), we have followed the rule of taking a plane model (in $\mathbf{P}^2$) or a non-singular model in $\mathbf{P}^3$. In fact, we have only used $\mathbf{P}^3$ models in the genus 4 cases. For genus 3 and 4, we take a canonical model (non-singular plane quartic

and non-singular complete intersection of a quadric and a cubic respectively). For genus 5 and 6, we find a smallest degree singular birational plane model. These plane curves are of degree 6 in all cases. The singularities are mainly nodes (type A_2) and simple cusps (type A_3) and often under cuspidal points, though there are some more complex ones of higher A_n type. All of the non-singular models that we have produced reduce mod p to non-singular models of the reduction of $X_0(N)$ for p not dividing the level N. The singular plane models reduce mod p (p not dividing N again) to singular plane models of the reduction of $X_0(N)$ with singularities of the same type, except for a few cases with small p where nodes become cusps or two singularities coalesce into a more complex one.

The initial version of the database contains data for all subhyperelliptic $X_0(N)$ and all other cases with genus ≤ 6 (with a few genus 5 and 6 cases not yet added). This covers all $N < 60$ along with about half of the N between 60 and 80 and $N = 81, 121$.

We briefly indicate how the models were arrived at. Equations in the genus 0 and elliptic cases and modular functions giving the coordinate generators are reasonably well-known (see [Lig75] for the elliptic cases), though we found them again anyway as part of our general procedure of searching for small degree rational functions on $X_0(N)$. We considered functions generated by Dedekind eta products and weight 2 integral forms coming from eta products, various types of theta series and Eisenstein series. For the hyperelliptic cases with genus g, functions x and y with poles at ∞ giving a $y^2 = f(x)$ type Weierstrass equation are determined from constructing weight two cusp forms G and F with q-expansions $q^{g-1} + ..$ and $q^g + ..$ respectively. These were found in terms of eta products and theta forms and the results were checked against the output of MAGMA's modular forms package.

In the non-subhyperelliptic cases, we started from a canonical image simplified by applying LLL as output by `ModularCurveQuotient`, occasionally slightly adapting this to get good reduction at 2. In the genus 5 and 6 cases, we determined minimal degree (singular) plane models from these. Genus 5 is fairly straightforward. For the method to find degree 6 plane images in the genus 6 case, see [Har13] where the genus 5 case is also discussed.

Having determined a model as the image of the mapping $X_0(N)$ into $\mathbf{P}^{r-1}$ by $z \mapsto [f_1(z) : \ldots : f_r(z)]$, $r = 3$ or 4, we then found expressions for the weight 2 forms f_i of the type described above (eta products etc.). For more information on this, see the subsection `135.8` where we also give intrinsics to return a symbolic description of the construction of the generating modular functions/forms and to return q-expansions up to a desired precision. We wished to have concrete expressions for these functions/forms in terms of certain basic types, independent of the generic modular symbol method of generating q-expansions for bases of forms. It also allows for slightly faster reconstruction of q-expansions.

`SmallModularCurve(N)`

`SmallModularCurve(N,K)`

The first intrinsic returns the model for $X_0(N)$ over the rationals from the small modular curve database. The second returns the base change of this to K, which should be a characteristic zero field.

If there is no database entry yet for level N, a runtime error results.

`IsInSmallModularCurveDatabase(N)`

Returns whether or not there is a data for level N in the small modular curves database.

Example H135E1

```
> IsInSmallModularCurveDatabase(79);
false
> IsInSmallModularCurveDatabase(35);
true
> SmallModularCurve(35);
Hyperelliptic Curve defined by y^2 + (-x^4 - x^2 - 1)*y = -x^7 - 2*x^6 - x^5 -
    3*x^4 + x^3 - 2*x^2 + x over Rational Field
> C<x,y,z> := SmallModularCurve(63);
> C;
Curve over Rational Field defined by
x^5*y - 2*x^4*y^2 + 3*x^3*y^3 - 2*x^2*y^4 + x*y^5 - 2*x^3*z^3 + x^2*y*z^3 +
    x*y^2*z^3 - 2*y^3*z^3 + z^6
```

135.3 Projection Maps

The database contains information allowing the reconstruction of the standard projection maps between the models of the level N and level M curves for any M dividing N.

`ProjectionMap(CN,N,CM,M)`

The curves CN and CM should be base changes to the same characteristic zero field K of the small modular database curves of levels N and M with $M|N$.

Returns the natural projection map $CN \to CM$ that corresponds to $z \mapsto z$ in terms of the upper half-plane quotient models and to $(E, C) \mapsto (E, (N/M)C)$ in the moduli space interpretation where non-cuspidal points correspond to isomorphism classes of an elliptic curves E with a cyclic subgroup C of order N.

`ProjectionMap(CN,N,CM,M,r)`

The curves CN and CM should be base changes to the same characteristic zero field K of the small modular database curves of levels N and M with $M|N$ and r should be a positive integer divisor of N/M.

Returns the projection map $CN \to CM$ that corresponds to $z \mapsto rz$ in terms of the upper half-plane quotient models and to

$$(E, C) \mapsto (E/(N/r)C, (N/(Mr))C/(N/r)C)$$

in the moduli space interpretation where non-cuspidal points correspond to isomorphism classes of an elliptic curves E with a cyclic subgroup C of order N.

Example H135E2

We use a 3-projection (case $r = 3$ in the above) in our extended example at the end of the section. Here, we just give some simple examples of the calls.

```
> C63<x,y,z> := SmallModularCurve(63);
> C63;
Curve over Rational Field defined by
x^5*y - 2*x^4*y^2 + 3*x^3*y^3 - 2*x^2*y^4 + x*y^5 - 2*x^3*z^3 + x^2*y*z^3 +
    x*y^2*z^3 - 2*y^3*z^3 + z^6
> C21 := SmallModularCurve(21);
> C21;
Elliptic Curve defined by y^2 + x*y = x^3 - 4*x - 1 over Rational Field
> C3 := SmallModularCurve(3);
> C3;
Curve over Rational Field defined by
0
> ProjectionMap(C63,63,C21,21);
Mapping from: Crv: C63 to CrvEll: C21
with equations :
8*x^3*y - 6*x^2*y^2 + 6*x*y^3 + 2*y^4 - 7*x^3*z + 15*x^2*y*z - 10*x*y^2*z +
    4*y^3*z - 7*x*y*z^2 + 7*y^2*z^2 + 5*x*z^3 - 15*y*z^3 - 2*z^4
-7*x^4 + 17*x^3*y - 18*x^2*y^2 + 11*x*y^3 - y^4 + 3*x^2*y*z + 5*x*y^2*z +
    5*y^3*z + 21*x^2*z^2 - 28*x*y*z^2 + 7*y^2*z^2 + x*z^3 - 3*y*z^3 - 13*z^4
-4*x^3*y + 3*x^2*y^2 - 3*x*y^3 - y^4 + 3*x^2*y*z - 2*x*y^2*z - 2*y^3*z +
    7*x*y*z^2 - 7*y^2*z^2 + x*z^3 + 4*y*z^3 + z^4
> ProjectionMap(C63,63,C3,3,7); //7-projection
Mapping from: Crv: C63 to Crv: C3
Composition of Mapping from: Crv: C63 to Crv: C
with equations :
x^2 - x*y + y^2 + 2*x*z - y*z - 2*z^2
x^2 - x*y + y^2 - x*z + 2*y*z - 2*z^2
x^2 - x*y + y^2 - x*z - y*z + z^2 and
Mapping from: Crv: C63 to Curve over Rational Field defined by
0
with equations :
x^3 + 3*x^2*y - 3*x*y^2 + 4*y^3 - 3*x^2*z - 2*z^3
x^2*z and
Mapping from: Curve over Rational Field defined by
0 to Curve over Rational Field defined by
0
with equations :
27*$.2
$.1 and
Mapping from: Curve over Rational Field defined by
0 to Crv: C3
with equations :
$.1^3
```

```
$.1^2*$.2 + 9*$.1*$.2^2 + 27*$.2^3
```

135.4 Automorphisms

For modular curves $X_0(N)$, there is a finite group of automorphisms $B_0(N)$ which come from matrices acting on the complex upper half-plane. This group is isomorphic to $\mathrm{Nm}_{\mathrm{SL}_2(\mathbf{R})}(\Gamma_0(\mathrm{N}))/\Gamma_0(\mathrm{N})$ where $\mathrm{Nm}_{\mathrm{SL}_2(\mathbf{R})}(\Gamma_0(\mathrm{N}))$ is the normaliser of $\Gamma_0(N)$ in $\mathrm{SL}_2(\mathbf{R})$. It is generated by Atkin-Lehner involutions and, if 4 or 9 divides N, a transformation of the form $z \mapsto z + (1/r)$ for some $1 < r \,|\, 24$ (see [AL70] and also [Bar08] for the correct structure in all cases). We denote the latter transformations by S_r and write w_d for the d-th Atkin-Lehner involution ($d|N$, $(d, N/d) = 1$). These transformations have a natural modular interpretation.

A famous result of Kenku and Momose [KM88], completed by Elkies for $N = 63$ [Elk90], says that $B_0(N)$ gives the *full* group of algebraic automorphisms of $X_0(N)$ (in characteristic zero) when the genus of $X_0(N)$ is at least 2, except for $N = 37$ and 63. In fact, there is a slight error in their analysis, and $N = 108$ is also an exception [Har]. In these three cases $B_0(N)$ is of index two in the full automorphism group $A_0(N)$ of $X_0(N)$.

We have precomputed and stored the w_d and S_r curve isomorphisms in the database. Let $g(N)$ denote the genus of $X_0(N)$. There are intrinsics to return these individual isomorphisms and a special intrinsic to compute and return the automorphism group $B_0(N)$, when $g \leq 1$, or $A_0(N)$, when $g \geq 2$, over various fields as a `GrpAutCrv`. We sometimes refer to the elements of these groups as modular automorphisms to emphasise that they are not all of the curve automorphisms in the $g(N) = 0$ or 1 cases (when the automorphism group is infinite).

Except in the $N = 108$ case, all automorphisms are defined over the cyclotomic field $\mathbf{Q}(\mu_r)$ where r is the largest divisor of 24 such that $r^2|N$ (for this field of definition, r can be replaced by $r/2$ if r is twice an odd number). Specifically, the w_d are all defined over $\mathbf{Q}$ and S_r has $\mathbf{Q}(\mu_r)$ as its field of definition.

The automorphism group intrinsics include in various ways an integer n that specifies a cyclotomic level and they return the subgroup of $A_0(N)$, or $B_0(N)$ for $g \leq 1$, containing all elements defined over $\mathbf{Q}(\mu_n)$. In all cases, the group of automorphism returned is a semi-direct product of groups, one for each prime p dividing N. For $p > 3$ this group is just C_2, generated by an Atkin-Lehner involution. For $p = 2$ or 3, if $p^2|N$ and n is appropriately divisible, the p-component is more complex. We build up the automorphism group in the various cases from the known structure. This is much faster than using the generic algorithm to compute it. The only real time goes into building up the total set of function field automorphisms (starting from the stored data for the generators) that is cached in the `GrpAutCrv` object.

`AtkinLehnerInvolution(CN,N,d)`

The curve CN should be a base change of the small modular database curve of level N to a field of characteristic 0. d is a divisor of N with $(d, N/d) = 1$. Returns the Atkin-Lehner involution w_d as a scheme automorphism of CN. If the full automorphism group G of CN has already been computed and cached with CN (see the intrinsics below and the section in the Algebraic Curves chapter on automorphism groups) then the intrinsic will return w_d as a `GrpAutCrvElt` in G. Note that the intrinsic should terminate immediately in the first case, when it is just retrieving the equations from the database file, but has to do a little bit of work in the second case.

`SrAutomorphism(CN,N,r,u)`

The curve CN should be a base change of the small modular database curve of level N to a field K of characteristic 0. r is a divisor of 24 such that $r^2|N$. u is a primitive r-th root-of-unity in K. Returns S_r as a scheme automorphism of CN that corresponds to $z \mapsto z + (1/r)$ if K is embedded into $\mathbf{C}$ so that u maps to $e^{2\pi i/r}$. If the full automorphism group G of CN has already been computed and cached with CN (see the intrinsics below and the section in the Algebraic Curves chapter on automorphism groups) then the intrinsic will return S_r as a `GrpAutCrvElt` in G. Note that the intrinsic will usually terminate more quickly in the first case, when it is just retrieving equations from the database file and maybe performing a composition, whereas in the second case it is doing an extra little bit of work.

`ExtraAutomorphism(CN,N,u)`

The curve CN should be a base change of the small modular database curve of level N to a field K of characteristic 0. N is 37, 63 or 108. u is a primitive r-th root-of-unity in K where $r = 1$ (so $u = 1$) when $N = 37$ and $r = 3$ in the other cases. Returns an automorphism of CN in $A_0(N)$ that is not in $B_0(N)$ (i.e. not given by the action of a matrix on the complex upper half-plane: see the introduction). This "extra" automorphism generates $A_0(N)$ over $B_0(N)$ and is an involution when $N = 37$ (the hyperelliptic involution) or $N = 108$ and is of order 4 when $N = 63$ with square equal to the Atkin-Lehner involution w_9. If the full automorphism group G of CN has already been computed and cached with CN (see the intrinsics below and the section in the Algebraic Curves chapter on automorphism groups) then the intrinsic will return the extra automorphism as a `GrpAutCrvElt` in G. Note that the intrinsic should terminate immediately in the first case, when it is just retrieving the equations from the database file, but has to do a little bit of work in the second case.

`AutomorphismGroupOverQ(CN,N)`

`Install` BoolElt *Default :* `true`

The curve CN should be a base change of the small modular database curve of level N to a field K of characteristic 0. Computes and returns the full group of automorphisms of CN (automorphisms in $B_0(N)$ if the genus of $X_0(N)$ is ≤ 1: see

the introduction) defined over **Q**. If the parameter `Install` has the value `true`, which is the default, the return value is internally installed and cached as the (full) automorphism group of CN, so the user should be careful that K doesn't contains roots of unity which generate a field over which some extra modular automorphisms are defined (viz. roots of unity of exact order r where $2 < r|24$ and $r^2|N$).

`AutomorphismGroupOverCyclotomicExtension(CN,N,n)`

`Install` BoolElt *Default :* `true`

The curve CN should be a base change of the small modular database curve of level N to a field K which is the cyclotomic extension of **Q** obtained by adjoining the n-th roots of unity (the precise condition is that $K.1$ should be a primitive n-th root of unity). Computes and returns the full group of automorphisms of CN (automorphisms in $B_0(N)$ if the genus of $X_0(N)$ is ≤ 1: see the introduction) defined over K. If the parameter `Install` has the value `true`, which is the default, the return value is internally installed and cached as the (full) automorphism group of CN.

`AutomorphismGroupOverExtension(CN,N,n,u)`

`Install` BoolElt *Default :* `true`

The curve CN should be a base change of the small modular database curve of level N to a field K which is an extension of **Q** containing the n-th roots of unity. u should be a primitive n-th root of unity in K. Computes and returns the full group of automorphisms of CN (automorphisms in $B_0(N)$ if the genus of $X_0(N)$ is ≤ 1: see the introduction) defined over the cyclotomic field $\mathbf{Q}(\mu_n)$. If the parameter `Install` has the value `true`, which is the default, the return value is internally installed and cached as the (full) automorphism group of CN. In that case, the user should be careful that n is well-chosen so that K doesn't contain higher order roots of unity which generate a subfield over which some modular automorphisms are defined that are not defined over $\mathbf{Q}(\mu_n)$.

Example H135E3

```
> C := SmallModularCurve(37);
> C;
Hyperelliptic Curve defined by y^2 + (-x^3)*y = 2*x^5 - 5*x^4 + 7*x^3 - 6*x^2
    + 3*x - 1 over Rational Field
> AtkinLehnerInvolution(C,37,37);
Mapping from: CrvHyp: C to CrvHyp: C
with equations :
$.1
$.2
$.1 - $.3
and inverse
$.1
$.2
```

```
$.1 - $.3
> ExtraAutomorphism(C,37,1); //the hyperelliptic involution
Mapping from: CrvHyp: C to CrvHyp: C
with equations :
$.1
$.1^3 - $.2
$.3
and inverse
$.1
$.1^3 - $.2
$.3
> C<x,y,z> := SmallModularCurve(48);
> C;
Hyperelliptic Curve defined by y^2 + (-x^4 - 1)*y = 3*x^4 over Rational Field
> AtkinLehnerInvolution(C,48,3);
Mapping from: CrvHyp: C to CrvHyp: C
with equations :
-z
x^4 - y + z^4
x
and inverse
-z
x^4 - y + z^4
x
> Type($1);
MapAutSch
> G := AutomorphismGroupOverQ(C,48);
> #G;
16
> AtkinLehnerInvolution(C,48,3);
Mapping from: CrvHyp: C to CrvHyp: C
with equations :
-z
x^4 - y + z^4
x
and inverse
-z
x^4 - y + z^4
x
> Type($1);
GrpAutCrvElt
> PermutationRepresentation(G);
Permutation group acting on a set of cardinality 6
Order = 16 = 2^4
    (2, 3)
    (1, 2)(3, 4)
    (5, 6)
Mapping from: GrpAutCrv: G to GrpPerm: $, Degree 6, Order 2^4 given by a rule
```

```
> // D8 X C2
> K<i> := QuadraticField(-1);
> CK<x,y,z> := SmallModularCurve(48,K);
> G := AutomorphismGroupOverCyclotomicExtension(CK,48,4);
> #G;
48
> SrAutomorphism(CK,48,4,i); // z -> z+(1/4)
Mapping from: CrvHyp: CK to CrvHyp: CK
with equations :
-i*x
y
z
and inverse
i*x
y
z
```

135.5 Cusps and Rational Points

Letting H^* denote the extended complex upper half-plane $\{z \in C|\mathrm{Im(z)} > 0\} \cup \mathbf{Q} \cup \infty$, $X_0(N)(\mathbf{C})$ identifies with the quotient space $H^*/\Gamma_0(N)$ and the cusps are the images of $\mathbf{Q} \cup \infty$. A complete set of representatives of cusps are the images of the points a/d where d runs through positive divisors of N and, for each d, a runs through a set of integer representatives of $(\mathbf{Z}/(d, N/d)\mathbf{Z})^\times$ relatively prime to d. In particular, if $(d, N/d) \leq 2$, we just get $1/d$. The image of the cusp at ∞ identifies with the image of $1/N$.

With respect to the rational structure of $X_0(N)$, the cusps are all algebraic points defined over cyclotomic fields. Specifically, for a given $d|N$, the a/d are a set of Galois-conjugate points, each one having $\mathbf{Q}(\mu_{(d,N/d)})$ as the field of definition, and the Galois group of $\mathbf{Q}(\mu_{(d,N/d)})$ over $\mathbf{Q}$ acts simply transitively on this set. In particular, if $(d, N/d) \leq 2$, the cusp $1/d$ is a $\mathbf{Q}$-rational point on $X_0(N)$. Thus, $\infty \sim 1/N$ and $0 = 0/1 \sim 1/1$ are always rational cusps.

The cuspidal points on the models for small modular curves are stored in the database. More precisely, for each $d|N$, the database stores either a rational point for the cusp $1/d$, if $(d, N/d) \leq 2$, or a cluster that defines the $\mathbf{Q}$-conjugate set of cusps a/d, if $(d, N/d) > 2$. The user can access these through an intrinsic. If the model C for $X_0(N)$ is singular, some cusps may correspond to singular points on C. There may be two cusps that lie over the same node of C. In any case, the *place* over each conjugate class of cusps a/d is unique (different for different d) and the database contains information to determine which place belongs to which cusp for places above a singular cuspidal point on the model. There is an intrinsic to return the curve place above each $\mathbf{Q}$-conjugate set of cusps a/d.

Non-cuspidal rational points are of particular interest since they correspond to classes of elliptic curves defined over $\mathbf{Q}$ with a cyclic N-isogeny also defined over $\mathbf{Q}$ up to twist. On the complete set of curves $X_0(N)$ of genus > 0, there are very few non-cuspidal rational points and all such were determined by Mazur and others in the 1970s. The database

stores the list of non-cuspidal rational points for each level N. These are never singular points on the chosen models.

`Cusp(CN,N,d)`

The curve CN should be a base change of the small modular database curve of level N to a field K of characteristic 0. d is a positive divisor of N. Returns the point on CN corresponding to the cusp $1/d$, if $(d, N/d) \leq 2$, or returns the cluster (zero-dimensional scheme) defined over K that is the reduced subscheme of CN consisting of the $\phi((d, N/d))$ cusps a/d, if $(d, N/d) > 2$ where ϕ is Euler's totient function.

`CuspIsSingular(N,d)`

Returns whether the points lying under the cusps a/d for $d|N$ on the small modular curve database model for $X_0(N)$ are singular. For a given d, they either all are or all are not.

`CuspPlaces(CN,N,d)`

The curve CN should be a base change of the small modular database curve of level N to a field K of characteristic 0. d is a positive divisor of N. Returns the sequence of places of CN that correspond to the $\phi((d, N/d))$ cusps a/d. If $(d, N/d) \leq 2$ or $K = \mathbf{Q}$ there will only be one place. However, if K is a proper extension of $\mathbf{Q}$, the $\mathbf{Q}$-conjugate cusps a/d may split into several Galois orbits over K.

`NonCuspidalQRationalPoints(CN,N)`

The curve CN should be a base change of the small modular database curve of level N of genus > 0 to a field K of characteristic 0. Returns the sequence of points on CN that correspond to non-cuspidal points in $X_0(N)(\mathbf{Q})$. There are only a small number of N for which this is a non-empty set and the rational points are non-singular ones on all of our models.

Example H135E4

```
> C := SmallModularCurve(32);
> C;
Elliptic Curve defined by y^2 = x^3 + 4*x over Rational Field
> Cusp(C,32,32); //cusp at infinity
(0 : 1 : 0)
> Cusp(C,32,8);
Cluster over Rational Field defined by
$.1^2 + 4*$.3^2,
$.2
> Degree($1);
2
> C<x,y,z> := SmallModularCurve(63);
> Cusp(C,63,1); // cusp at 0
```

```
(1 : 1 : 1)
> Cusp(C,63,7); // cusp at 1/7
(1 : 1 : 1)
> CuspIsSingular(63,1);
true
> CuspIsSingular(63,7);
true
> CuspPlaces(C,63,1);
[
    Place (2) at (1 : 1 : 1)
]
> CuspPlaces(C,63,7);
[
    Place (1) at (1 : 1 : 1)
]
> Cusp(C,63,3);
Cluster over Rational Field defined by
x - y,
y^2 + y*z + z^2
> Cusp(C,63,21);
Cluster over Rational Field defined by
x - y,
y^2 + y*z + z^2
> CuspPlaces(C,63,3)[1];
Place at ($.1 : $.1 : 1)
> CuspPlaces(C,63,21)[1];
Place at ($.1 : $.1 : 1)
> $1 eq $2;
false
```

135.6 Standard Functions and Forms

This section contains intrinsics that return the j-invariant as a rational function and normalised Eisenstein forms as meromorphic k-differentials on the small database models of $X_0(N)$. Standard variants of these can be obtained by pulling back by Atkin-Lehner involutions or pulling back the corresponding objects from lower level curves by the projection or r-projection maps.

The database actually only contains precomputed expressions for $E_2^{(N)}$ (see below), E_4 and E_6 for prime levels N and reconstructs everything else from these objects using projection maps.

`jInvariant(CN,N)`

`jFunction(CN,N)`

The curve CN should be a base change of the small modular database curve of level N to a field of characteristic 0. These intrinsics return the j-invariant $j(z)$ as a rational function on CN. The second returns it as an element of the function field of CN. The first returns it as an element in the field of fractions of the coordinate ring of the ambient of CN, which is sometimes more convenient to use.

`jInvariant(p,N)`

The point or place p is a non-cuspidal point or a non-cuspidal place on a base change CN of the small modular database curve of level N to a field of characteristic 0. Returns the value of $j(z)$ at p, the value lying in L if p is a point in $CN(L)$ or in the residue class field of p if p is a place of CN. If p is a point, it should be non-singular. However, if the j function is defined at p, the intrinsic will still return a value.

`jNInvariant(p,N)`

Exactly as `jInvariant` above, except that the intrinsic gives the value of rational function $j(Nz)$ at the point or place. This is equivalent to computing the j-invariant value on the image of p under the Fricke involution w_N.

`E2NForm(CN,N)`

The curve CN should be a base change of the small modular database curve of level N to a field of characteristic 0. $E_2^{(N)}(z) = NE_2(Nz) - E_2(z)$ is a weight 2 integral form for $\Gamma_0(N)$ where $E_2(z) = 1 - 24e^{2\pi iz} + \ldots$ is the normalised weight 2 Eisenstein series. $E_2^{(N)}(z)$ corresponds to a meromorphic differential (defined over $\mathbf{Q}$) on $X_0(N)$. The intrinsic returns $E_2^{(N)}(z)$ as a meromorphic differential in the function field of CN.

`E4Form(CN,N)`

The curve CN should be a base change of the small modular database curve of level N to a field of characteristic 0. Returns a rational function f and a differential form ω in the function field of CN such that the Eisenstein series $E_4(z) = 1+240e^{2\pi iz}+\ldots$ as a meromorphic 2-differential on CN is given by $f\omega^2$.

`E6Form(CN,N)`

The curve CN should be a base change of the small modular database curve of level N to a field of characteristic 0. Returns a rational function f and a differential form ω in the function field of CN such that the Eisenstein series $E_6(z) = 1-504e^{2\pi iz}+\ldots$ as a meromorphic 3-differential on CN is given by $f\omega^3$. Note that the same differential ω is returned by the `E4Form` intrinsic.

135.7 Parametrized Structures

As with the other modular equation databases in MAGMA there is functionality to explicitly compute a cyclic N-isogeny or cyclic subgroup of order N on an elliptic curve that is represented by a non-cuspidal point or place on the $X_0(N)$ models with the usual moduli space interpretation of the modular curves. The points represent equivalence classes of these structures (elliptic curve E with cyclic subgroup C, (E, C), or cyclic N-isogeny $\phi : E \to F$) up to isomorphism. In particular, if (E, C) is an elliptic curve with cyclic subgroup with both E and C defined over a field K, then any quadratic twist E_1 of E with the twisted subgroup C_1, which is the image of C under the isomorphism from E to E_1 over a quadratic extension, is represented by the same point on $X_0(N)$. If $j \neq 0, 1728$, any two elliptic curves over a field K with j-invariant j are quadratic twists of each other. Because of this, if $j(p) \neq 0, 1728$, any elliptic curve E over the field of definition K of the point p and j-invariant $j(p)$ will have a cyclic subgroup C defined over K such that (E, C) is represented by p or, equivalently, there is a cyclic N-isogeny defined over K and with domain E that is represented by p. We allow the user to pass in his choice of E for the base elliptic curve.

If z is a non-real complex number, let Λ_z be the two dimensional complex lattice $\mathbf{Z} \oplus \mathbf{Z}z$. Under the identification of the complex points of our model of $X_0(N)$ with $H^*/\Gamma_0(N)$ which follows from the identification of coordinate functions with specific modular functions or forms (see the next section), if a non-cuspidal point p corresponds to a point z in the upper halfplane H, the point p in the moduli interpretation is represented by (E_z, C), where $E_z = \mathbf{C}/\Lambda_z$ and C is the cyclic subgroup generated by $(1/N) \bmod \Lambda_z$, or the cyclic N-isogeny $\phi : E_z \to E_{Nz}$, $t \mapsto Nt$.

The computation of structures uses well-known methods. If d is a suitable differential on $X_0(N)$, we start with the data

$$p_1 = (N/12)(E_2^{(N)}/d)(p) \qquad E_4 = (E_4/d^2)(p) \qquad E_6 = (E_6/d^3)(p)$$

$$\tilde{E}_4 = N^2((E_4/d^2)(w_N(p))(w_N^*(d)/d)(p)^2 \qquad \tilde{E}_6 = N^3((E_6/d^3)(w_N(p))(w_N^*(d)/d)(p)^3$$

p is represented by (E, C) where E is given by $y^2 = x^3 - (E_4/48)x - (E_6/864)$ and the monic polynomial in x that defines C is computed from $p_1, E_4, E_6, \tilde{E}_4, \tilde{E}_6$ by the same algorithm as used in the SEA Elkies variant of Schoof's algorithm. The corresponding isogeny $\phi : E \to F$ with kernel C can then be computed using Velu's formulae.

```
SubgroupScheme(p,N)
```

```
SubgroupScheme(p,N,E)
```

Here p is a non-cuspidal point or place on C_N, a base change of the curve from the small modular curve database of level N to a field of characteristic zero. When p is given as a point, it should be non-singular point on C_N. The function returns an elliptic curve E_1 and a subgroup scheme G of E_1, both defined over the field of definition K of p (K if p is a point in $C_N(K)$; the residue class field of p if p is a place) such that G gives a cyclic subgroup of order N on E_1 and (E_1, G) represents p in the moduli space interpretation.

In the second version, when $j(p)$ should not be 0 or 1728, E should be an elliptic curve over a subfield of K with j-invariant $j(p)$. In this case, E_1 is taken as E or the base change of E to K, and only G is returned.

`Isogeny(p,N)`

`Isogeny(p,N,E)`

Here p is a non-cuspidal point or place on C_N, a base change of the curve from the small modular curve database of level N to a field of characteristic zero. When p is given as a point, it should be non-singular point on C_N. The function returns a cyclic N-isogeny of elliptic curves $\phi : E_1 \to F_1$, all defined over the field of definition K of p (K if p is a point in $C_N(K)$; the residue class field of p if p is a place), which represents p in the moduli space interpretation.

In the second version, when $j(p)$ should not be 0 or 1728, E should be an elliptic curve over a subfield of K with j-invariant $j(p)$. In this case, E_1 is taken as E or the base change of E to K.

Example H135E5

```
> C := SmallModularCurve(14);
> rats := NonCuspidalQRationalPoints(C,14);
> rats;
[ (2 : 2 : 1), (9 : -33 : 1) ]
> jInvariant(rats[1],14);
16581375
> jNInvariant(rats[1],14);
-3375
> jInvariant(rats[2],14);
-3375
> jNInvariant(rats[2],14);
16581375
> G,E := SubgroupScheme(rats[2],14);
> E;
Elliptic Curve defined by y^2 + x*y = x^3 - x^2 - 2*x - 1 over Rational Field
> G;
Subgroup scheme of E defined by x^7 + 7*x^6 - 7*x^5 - 35*x^4 + 7*x^3 + 35*x^2
    + 7*x - 2
> jInvariant(E);
-3375
> phi := Isogeny(rats[2],14);
> phi;
Elliptic curve isogeny from: Elliptic Curve defined by y^2 + x*y = x^3 - x^2 -
2*x - 1 over Rational Field to Elliptic Curve defined by y^2 + x*y = x^3 - x^2
- 1822*x + 30393 over Rational Field
taking (x : y : 1) to ((x^14 + 16*x^13 + 431*x^12 + 1604*x^11 - 768*x^10 -
    12344*x^9 - 7979*x^8 + 25044*x^7 + 29067*x^6 - 4796*x^5 - 12252*x^4 +
    3244*x^3 + 6789*x^2 + 2504*x + 371) / (x^13 + 16*x^12 + 67*x^11 - 34*x^10
```

```
    - 495*x^9 - 332*x^8 + 967*x^7 + 1048*x^6 - 431*x^5 - 834*x^4 - 205*x^3 +
    52*x^2 + 13*x - 2) : (x^20*y + 23*x^19*y - 364*x^19 - 192*x^18*y -
    2093*x^18 + 288*x^17*y - 2821*x^17 + 1248*x^16*y + 19201*x^16 -
    5484*x^15*y + 73430*x^15 + 5293*x^14*y - 59871*x^14 - 10840*x^13*y -
    398076*x^13 - 50135*x^12*y - 153384*x^12 - 107060*x^11*y + 723331*x^11 -
    175648*x^10*y + 619619*x^10 - 15884*x^9*y - 395549*x^9 + 540867*x^8*y -
    167643*x^8 + 1295892*x^7*y + 895937*x^7 + 1548467*x^6*y + 848974*x^6 +
    947584*x^5*y + 19075*x^5 + 32804*x^4*y - 348579*x^4 - 407788*x^3*y -
    218505*x^3 - 305738*x^2*y - 56084*x^2 - 95233*x*y - 3710*x - 9831*y + 371)
/ (x^20 + 23*x^19 + 172*x^18 + 288*x^17 - 1755*x^16 - 5757*x^15 + 4334*x^14 +
    29683*x^13 + 7538*x^12 - 65053*x^11 - 47954*x^10 + 59387*x^9 + 72098*x^8 -
    8621*x^7 - 37663*x^6 - 12228*x^5 + 2214*x^4 + 1215*x^3 - 83*x^2 - 40*x +
    4) : 1)
> F := Codomain($1);
> jInvariant(F);
16581375
```

135.8 Modular Generators and q-Expansions

In the genus 0 cases, the database model is just the projective line $\mathbf{P}^1$ and the uniformising parameter $t = x/y$ (where x, y are the projective coordinates of $\mathbf{P}^1$) is a modular function on $X_0(N)$. We have normalised so that this function has a simple zero at the cusp 0 and a simple pole at the cusp ∞ with q-expansion of the form $q^{-1} + \ldots$ when $N > 1$. For $N = 1$ we just take the j-invariant as our parameter.

In the elliptic and hyperelliptic cases, the model is a minimal Weierstrass equation $y^2 + h(x)y = f(x)$ and the x and y coordinate functions are modular functions on $X_0(N)$. We have normalised so that one of the points at infinity of the model is the cusp ∞, so x and y have poles there. In the elliptic cases, there is only one point at infinity. In the hyperelliptic cases, the other point at infinity is always the image of ∞ under the hyperelliptic involution: ∞ is never a Weierstrass point in these cases.

We sometimes refer to genus zero, elliptic and hyperelliptic curves by the semi-standard term subhyperelliptic. The models in the non-subhyperelliptic cases are ordinary projective and are sub-canonical images of $X_0(N)$. This means that there are linearly-independent weight 2 cusp forms $f_1, \ldots, f_r$ (which correspond to holomorphic differentials on $X_0(N)$) such that our model is the image of $X_0(N)$ under the map $z \mapsto [f_1(z) : \ldots : f_r(z)]$ into $\mathbf{P}^{r-1}$.

As stated earlier, although we sometimes used MAGMA's modular symbols machinery to compute a basis for forms when deriving models, we also worked out expressions for the modular functions t, x and y and the forms f_i in terms of certain basic types. This gives the models a degree of independence from the modular symbol machinery and also allows for faster reconstruction of q-expansions of the generating functions/forms. We describe the basic types here. There is an intrinsic that the user may call to retrieve the information as to how the forms are built up and one to return the actual q-expansions up to any desired precision. In some cases, we have used the cusp forms associated to an

elliptic curve of conductor N. We would have liked to avoid this and maybe with a bit more work we may be able to find alternative expressions that avoid using them. In fact, we should be using more of the theta series associated to quaternion algebras (this was an oversight only discovered recently: we are currently only using the theta series attached to one isomorphism class of maximal order for each quaternion algebra). Leaving aside the forms associated to elliptic curves, the basic types that we have looked at are all classical. We could also have tried to use some other basic types like the generalised eta products of Y. Yang.

The basic types fall into a number of categories:

(i) Dedekind eta products.

(ii) Theta series of binary quadratic forms of various kinds and theta series of quaternion algebras.

(iii) Weight 2 Eisenstein series of prime level.

(iv) Weight two cusp forms of elliptic curves.

Dedekind eta products for level N are of the form $\eta(d_1z)^{r_1}\eta(d_2z)^{r_2}\dots\eta(d_nz)^{r_n}$ where the d_i run over a subset of the positive divisors of N, the r_i are integers and $\eta(z)$ is the Dedekind η-function which has q-expansion $q^{1/24}\prod_{n=1}^{\infty}(1-q^n)$. If $\sum r_i$ is even and $\sum r_i d_i$ and $\sum r_i(N/d_i)$ are divisible by 24, the product gives a modular form of weight $(1/2)\sum r_i$ for $\Gamma_0(N)$ with a certain character mod N. The form only has zeroes and poles at cusps with order given by a formula depending on N, r_i and d_i. We use these products to give modular functions and weight 2 forms and occasionally weight one forms to be used in combination with weight one forms coming from theta series.

The theta series associated to binary quadratic forms come in several flavours.

If $Q(x,y)=ax^2+bxy+cz^2$ (with $a,b,c\in\mathbf{Z}, a>0$) has discriminant $-N$, then the usual theta series $\theta_Q(z)=\sum_{(x,y)\in\mathbf{Z}^2}q^{Q(x,y)}$ is a weight one form of level N for the quadratic character given by the Jacobi symbol $(\frac{-N}{\cdot})$. The product of two of these gives a weight 2 integral form for $\Gamma_0(N)$.

If $N\equiv 1 \bmod 4$ and $Q(x,y)=ax^2+bxy+cz^2$ is a form with discriminant $-4N$ with a odd and b,c even, the function $\theta_Q^{(1)}(z)=\sum_{(x,y)\in\mathbf{Z}^2,xodd}(-1)^y q^{Q(x,y)/4}$ is a weight one form of level N for the character $(\frac{\cdot}{N})\chi^a$ where χ is the quartic character giving the action of $\Gamma_0(1)$ on $\eta(z)^6$. If Q_1,Q_2 are two such forms, one of which has $a\equiv 1 \bmod 4$ and $a\equiv 3 \bmod 4$ for the other, then $\theta_{Q_1}^{(1)}\theta_{Q_2}^{(1)}$ is a weight 2 integral form for $\Gamma_0(N)$. Additionally, the function $\theta_Q^{(2)}(z)=\sum_{(x,y)\in\mathbf{Z}^2,xodd}q^{Q(x,y)/4}$ is a weight one form of level $2N$ for the character $(\frac{\cdot}{N})\chi^a\delta$ where δ is the unique non-trivial character that factors through the quotient $\Gamma_0(2N)/\Gamma_0(4N)$. Again, $\theta_{Q_1}^{(2)}\theta_{Q_2}^{(2)}$ is a weight 2 form for $\Gamma_0(2N)$ if the as for Q_1 and Q_2 are non-congruent mod 4.

If $N\equiv 3 \bmod 8$ and $Q(x,y)=ax^2+bxy+cz^2$ is a form with discriminant $-N$ (here a,b,c must all be odd), the function $\theta_Q^{(3)}(z)=\sum_{(x,y)\in\mathbf{Z}^2,xodd}(-1)^y q^{Q(x,y)/2}$ is a weight one form of level N for the character $(\frac{-N}{\cdot})\chi$ where χ is the quadratic character giving the action of $\Gamma_0(1)$ on $\eta(z)^{12}$. The product of any two of these gives a weight 2 integral form for $\Gamma_0(N)$.

If $N = p_1 \dots p_r$ is a product of an odd number of distinct primes, let H_N be the quaternion algebra over $\mathbf{Q}$ which is ramified precisely at ∞ and the primes p_i. Let O_N be a maximal order of H_N, which is a $\mathbf{Z}$-lattice of rank 4, and Q_N the positive-definite quadratic form on it given by the reduced norm Nm of H_N. Then the theta function $\theta_{Q_N}(z) = \sum_{x \in O_N} q^{Nm(x)}$ is a weight two integral form for $\Gamma_0(N)$.

The weight 2 Eisenstein series of prime level p is the form $pE_2(pz) - E_2(z)$, which is an integral form of weight 2 for $\Gamma_0(p)$. Here $E_2(z)$ is the usual normalised weight 2 Eisenstein series (which isn't a modular form for any level) with q-expansion $1 - 24q - 72q^2 - \dots$. We usually normalise the prime level Eisenstein series by dividing by $GCD(24, p-1)$ which is the GCD of its integral coefficients.

If E is an elliptic curve over Q of conductor N, then the modular form $f_E(z)$ associated to E, which is returned by the call `ModularForm(E)`, is a weight 2 cusp form for $\Gamma_0(N)$.

We also use three standard operations on forms/functions.

If $f(z)$ is a modular function on $X_0(N)$ then $D(f) = (1/2\pi i)(df/dz)$ is a weight 2 form for $\Gamma_0(N)$.

If $f(z)$ is a form/function on $X_0(M)$ and d is a positive integer dividing N/M then $f(dz)$ is a form/function on $X_0(N)$.

If f is a form for $\Gamma_0(N)$ and χ is the quadratic character with conductor d where $|d|^2$ divides N, then the twisted form $f \otimes \chi$ is also a form for $\Gamma_0(N)$ where if f has the q-expansion $\sum a_n q^n$ then $f \otimes \chi$ has q-expansion $\sum a_n \chi(n) q^n$.

The basic types are forms or functions of types (i)-(iv), possibly operated on by one of the above 3 operations. Basic functions are functions of basic type or quotients of forms of basic type of the same weight. We build modular functions as rational functions of basic functions. We build modular forms of weight 2 as linear combinations of weight 2 forms of basic type or products of modular functions with weight 2 forms of basic type. We give some examples below.

`qExpansionExpressions(N)`

This procedure prints out a mini-program that is used to compute the modular functions that correspond to t or to x, y on $X_0(N)$ in the subhyperelliptic cases or the weight 2 cusp forms $f_1, \dots, f_r$ on $X_0(N)$ in the other cases. These give expressions for the forms/functions in terms of forms/functions of basic type. The format of the mini-program is as follows.

There are a sequence of statements. Each except the last is of the form $v = \langle expr \rangle$ where v is a variable name, $\langle expr \rangle$ is an expression and the statement assigns the value of the expression (a modular form or function on $X_0(N)$) to variable v. Each expression is a string of terms separated by binary arithmetic operators $\hat{} \; * \; / \; + \; -$ which have their usual meaning, are listed in precedence order and evaluate left to right for operators of the same precedence. There may also be subexpressions in () brackets. Subexpressions in brackets are evaluated first and replaced with their value, inner bracketed subexpressions being evaluated before outer ones. Terms are of one of the following types, possibly modified by an operator:

(i) An integer. This either multiplies a form/function, is an exponent for powering or gives a constant function.

(ii) q. This is just $e^{2\pi iz}$, the 'q' of the q-expansion.

(iii) A variable that has previously been assigned to.

(iv) An eta product. $e\{\langle d_1\ r_1\rangle\langle d_2\ r_2\rangle\dots\langle d_n\ r_n\rangle\}$ represents the eta product $\eta(d_1z)^{r_1}\eta(d_2z)^{r_2}\dots\eta(d_nz)^{r_n}$.

(v) Binary theta functions as described in the introduction. For a positive-definite binary quadratic form $Q(x,y)=Ax^2+Bxy+Cy^2$,
$th0\{A\ B\ C\}$ is the usual theta series of Q.
$th1\{A\ B\ C\}$ is $(1/2)q^{-a/4}\theta_Q^{(1)}$ where a is A mod 4. This has q-integral q-expansion.
$th2\{A\ B\ C\}$ is $(1/2)q^{-a/4}\theta_Q^{(2)}$ where a is A mod 4. This has q-integral q-expansion.
$th3\{A\ B\ C\}$ is $(1/2)q^{-1/2}\theta_Q^{(3)}$. This has q-integral q-expansion.

(vi) Theta functions of quaternion algebras as described in the introduction. $thQ\{N\}$ represents the weight 2 theta function attached to a maximal order of the quaternion algebra of level N, where N is a product of an odd number of distinct primes. The notation is deficient here. The theta series depends on the isomorphism class of the maximal order chosen. Currently, we are only using the maximal order returned by MAGMA in the call `MaximalOrder(QuaternionAlgebra(N))`.

(vii) Eisenstein series. $E\{p\}$ represents the weight 2 normalised Eisenstein series of prime level p, $(pE_2(pz)-E_2(z))/GCD(24,p-1)$.

(viii) Cusp forms from elliptic curves. $f\{a_1\ a_2\ a_3\ a_4\ a_6\}$ represents the weight two newform associated to the elliptic curve over $\mathbf{Q}$ with equation $y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$.

Modifying operators are

(i) D. $D(f)$ is $(1/2\pi i)(df/dz)$, where f is a modular function on $X_0(N)$.

(ii) $[d]$. d is a positive integer. If f represents a form or function $f(z)$, then $f[d]$ represents $f(dz)$.

(iii) $\langle d\rangle$. If f is a form, $f\langle d\rangle$ represents the form twisted by the quadratic character of conductor d.

The last line of the mini-program is of the form $[t,1]$ if $X_0(N)$ is of genus 0, $[x,y,1]$ if $X_0(N)$ is elliptic or hyperelliptic, and $[f_1,\dots,f_r]$ otherwise, where t,x,y,f_i are variables that were assigned to in earlier lines. These give the generating functions/forms for the $X_0(N)$ model.

`qExpansionsOfGenerators(N,R,r)`

Returns q-expansions to precision r as Laurent series in the Laurent series ring R over $\mathbf{Q}$ for the sequence of modular forms/functions of level N that define the small modular curve database model of $X_0(N)$. As described in the introduction, the

return sequence is of the form $[t]$ if $X_0(N)$ is of genus 0, $[x, y]$ if $X_0(N)$ is elliptic or hyperelliptic, and $[f_1, \ldots, f_r]$ otherwise. Here t, x and y are Laurent series giving the q-expansions of functions which will have poles at the cusp ∞, so have negative power terms, and the f_i will actually be power series giving the q-expansions of weight 2 cusp forms.

The q-expansions are computed from the expressions which are given in the mini-program output by `qExpansionExpressions(N)`, generating the forms/functions from ones of basic type for which there are fast routines to compute q-expansions.

Example H135E6

We give some examples of the mini-programs for various levels

```
> qExpansionExpressions(8); //genus 0 case
x=e{<1 4><2 -2><4 2><8 -4>}
[x,1]
> qExpansionExpressions(15); //genus 1 case
F=e{<1 1><3 1><5 1><15 1>}
t=e{<1 -3><3 3><5 3><15 -3>}
s=-D(t)/F
x=(t^2-5*t-3+s)/2
y=(t^3-8*t^2-4*t+3+s*(t-3))/2
[x,y,1]
> qExpansionExpressions(30); //hyperelliptic case
F=e{<1 -1><2 3><3 1><5 1><6 -1><10 -1><15 -1><30 3>}
x=e{<1 2><2 -2><3 -1><5 -1><6 1><10 1><15 2><30 -2>}
s=-D(x)/F
y=(s+x^4+x^3+x^2)/2
[x,y,1]
> qExpansionExpressions(64); //genus 3 case
F1=e{<4 2><8 2>}
F2=F1<-8>
[(F2-F1)/4,(F2+F1)/2,F1[2]]
> qExpansionExpressions(53); //genus 4 case
t1=th1{1 2 54}
t2=th1{3 2 18}
t3=th1{9 2 6}
F1=q*t1*t2
F2=q*t2*t3
F3=(13*thQ{53}-E{53})/18
F4=f{1 -1 1 0 0}
[F1,(5*F1-3*F2-F3-F4)/2,-4*F1+2*F2+F3,-4*F1+F2+F3]
> qExpansionExpressions(63); //genus 5 case
F1=(th0{1 1 16}-th0{4 1 4})*th0{2 1 8}/2
F2=q*th1{1 2 22}*th1{3 6 10}
F3=F2[3]
F4=e{<3 2><21 2>}
[(F1-F2-F3+F4)/2,(F1-F2-F3-F4)/2,F3]
```

```
> R<q> := LaurentSeriesRing(Rationals());
> qExpansionsOfGenerators(49,R,30); //genus 1 case
[
    q^-2 - q^-1 + 1 + q^3 + q^5 - q^10 - q^13 + q^20 + q^24 + O(q^31),
    q^-3 - 2*q^-2 + 2*q^-1 - 1 - q + q^2 - 2*q^3 + q^4 - 2*q^5 - q^9 + 2*q^10
        + q^11 + 2*q^13 + q^15 + q^16 - 2*q^20 - q^22 - q^23 - 2*q^24 - q^25 -
        q^29 + q^30 + O(q^31)
]
```

135.9 Extended Example

We give an example that combines functionality for small modular curves with MAGMA's machinery for curve quotients and elliptic curves to compute the j-invariant of a special class of elliptic curves over $\mathbf{Q}$ up to quadratic twist. This class of curves arose in Wiles' original paper on Fermat's Last Theorem and the Shimura-Tanayama-Weil (STW) conjecture.

Example H135E7

In the Breuil-Conrad-Diamond-Taylor-Wiles proof of the STW conjecture, a couple of special cases arise that lead to a finite number of classes of elliptic curves up to $\bar{\mathbf{Q}}$-isomorphism that have to be determined and then checked directly for modularity. If $E[p]$ is the p-torsion subgroup of E considered as a $\mathrm{Gal}(\bar{\mathbf{Q}}/\mathbf{Q})$-module, these are cases where $E[3]$ and $E[5]$ are both reducible, $E[3]$ is reducible and $E[5]$ is irreducible but absolutely reducible as a $\mathrm{Gal}(\bar{\mathbf{Q}}/\mathbf{Q}(\sqrt{5}))$-module, or $E[5]$ is reducible and $E[3]$ is irreducible but absolutely reducible as a $\mathrm{Gal}(\bar{\mathbf{Q}}/\mathbf{Q}(\sqrt{-3}))$-module. The first case corresponds to non-cuspidal rational points on $X_0(15)$ and leads, up to isogeny, to one class of twists with $j = -25/2$ as may easily be verified using the intrinsics here. The second case corresponds to non-cuspidal rational points on a quotient of $X_0(75)$ and leads to curves with $j = 0$. We consider the third case in this example, showing that, up to isogeny, there is one class (up to quadratic twist) with j-invariant $(11/2)^3$.

The curves we want are precisely those that have a cyclic subgroup of order 5 rational over $\mathbf{Q}$ and for which the image of $\mathrm{Gal}(\bar{\mathbf{Q}}/\mathbf{Q})$ in $GL_2(\mathbf{F}_3)$, giving the action on $E[3]$, is the normaliser of a split Cartan subgroup.

Curves just satisfying the $E[3]$ condition are classified by non-cuspidal rational points on $X_0(9)/w_9$ that are not the image of a rational point of $X_0(9)$. Such a point p lifts to a point p_1 on $X_0(9)$ defined over some quadratic field K such that $p_1^\sigma = w_9(p_1)$ where σ is the nontrivial automorphism of K over $\mathbf{Q}$. If (E, C) is an elliptic curve/K with cyclic subgroup $C = \langle P \rangle$ rational over K that is represented by the moduli point p_1, then $E_1 = E/\langle 3P \rangle$ can be defined over $\mathbf{Q}$ so that the order 3 subgroups $C/\langle 3P \rangle$ and $(C/\langle 3P \rangle)^\sigma$ generate $E_1[3]$, are rational over K and are swapped by σ.

In the same way, adding the extra condition that there is a rational subgroup of order 5 leads to curves classified by non-cuspidal rational points of $X_0(45)/w_9$. These do not lift to rational points because $X_0(45)$ has no non-cuspidal rational points. $X_0(45)/w_9$ is $\mathbf{Q}$-isogenous to the rank 0 elliptic curve $X_0(15)$ so only has finitely many rational points. If p_1 is a lift, the image of p_1 under the 3-projection map $X_0(45) \to X_0(5)$ ($z \mapsto 3z$ on complex points) is a rational point that corresponds to the desired curve E_1 (with its cyclic 5-subgroup). We could apply the 3-projection

all the way down to $X_0(1)$ of course, but it is slightly more efficient to just project to level 5 and remove duplicate images before computing j-invariants.

```
> X45<x,y,z> := SmallModularCurve(45);
> w9 := AtkinLehnerInvolution(X45,45,9);
> G := AutomorphismGroup(X45,[w9]);
> C,prjC := CurveQuotient(G);
> c_inf := Cusp(X45,45,45);
> ptE := prjC(c_inf);
> E1,mp1 := EllipticCurve(C,ptE);
> E,mp2 := MinimalModel(E1);
> prjE := Expand(prjC*mp1*mp2);
```

E is a minimal model for $X_0(45)/w_9$ and $prjE$ is the projection from $X_0(45)$ to E that takes the cusp ∞ to the zero point of E. We compute the image of the cusps under $prjE$ so as to discount these when we find all the rational points on E. Since p and $w_9(p)$ map to the same point under $prjE$, we only need consider cusps up to w_9 equivalence. The non-rational conjugate cusps $\pm 1/3$ are defined over $\mathbf{Q}(\sqrt{-3})$ and are swapped by w_9, so map to the same rational point. The same holds for the cusps $\pm 1/15$.

```
> i0 := prjE(Cusp(X45,45,1));
> K := QuadraticField(-3);
> c3 := Cusp(X45,45,3);
> c3p := X45(K)!Representative(Support(c3,K));
> i3 := E!(prjE(c3p));
> c15 := Cusp(X45,45,15);
> c15p := X45(K)!Representative(Support(c15,K));
> i15 := E!(prjE(c15p));
> Ecusps := [E!0,i0,i3,i15];
```

E has rank zero, so we can recover all of its rational points using `TorsionSubgroup`. There are 8 in all, 4 of which are images of cusps. We find the 4 non-cuspidal ones and compute the pullbacks of them to $X_0(45)$. In each case, the pull back is a pair of conjugate points defined over a quadratic field. It is easiest to work with places here (and we don't have to worry about the base scheme of $prjE$). The pullback of each point gives a single place corresponding to the two conjugate points.

```
> T,mp := TorsionSubgroup(E);
> Epts := [mp(g) : g in T];
> Eptsnc := [P : P in Epts | P notin Ecusps];
> plcs := [Support(Pullback(prjE,Place(p)))[1] : p in Eptsnc];
```

We now perform the 3-projection to $X_0(5)$ on these places and discard duplicate images.

```
> X5 := SmallModularCurve(5);
> prj3 := ProjectionMap(X45,45,X5,5,3);
> prj3 := Expand(prj3);
> plcs5 := [Support(Pushforward(prj3,p))[1] : p in plcs];
```

```
> plcs5 := Setseq(Seqset(plcs5));
```

We compute the j-invariants of the two images and verify that they represent isogenous curves under the cyclic 5-isogeny coming from the rational 5-subgroup (i.e. they are images of each other under w_5).

```
> js := [jInvariant(p,5) : p in plcs5];
> js;
[ -1680914269/32768, 1331/8 ]
> w5 := AtkinLehnerInvolution(X5,5,5);
> Pullback(w5,plcs5[1]) eq plcs5[2];
true
```

Finally, we can use MAGMA intrinsics to check that the elliptic curves with these j-invariants actually do satisfy the property that the image of the action of Galois on 3-torsion is the normaliser of a split Cartan subgroup (D_8). As this property remains true for quadratic twists and is unchanged by images under a rational 5-isogeny, we only need check it for one curve over $\mathbf{Q}$ with one of the j-invariants. We also check that a minimal twist has conductor 338 from which modularity can be checked explicitly.

```
> Ej := EllipticCurveWithjInvariant(js[2]);
> Ej := MinimalModel(MinimalTwist(Ej));
> Conductor(Ej);
338
> ThreeTorsionType(Ej);
Dihedral
```

135.10 Bibliography

[**AL70**] A.O.L. Atkin and J. Lehner. Hecke operators on $\Gamma_0(N)$. *Math. Annalen*, 185:134–160, 1970.

[**Bar08**] F. Bars. The group structure of the normaliser of $\Gamma_0(N)$ after Atkin-Lehner. *Communications in Algebra*, 36:2160–2170, 2008.

[**Elk90**] N. Elkies. The automorphism group of the modular curve $X_0(63)$. *Compositio Mathematica*, 74:203–208, 1990.

[**Har**] M. C. Harrison. A new automorphism of $X_0(108)$. Preprint: online at arXiv as arXiv:1108.5595v3[math.NT].

[**Har13**] M. C. Harrison. Explicit solution by radicals, gonal maps and plane models of algebraic curves of genus 5 or 6. *J. Symb. Comp.*, 51:3–21, 2013.

[**KM88**] M.A. Kenku and F. Momose. Automorphism groups of the modular curves $X_0(N)$. *Compositio Mathematica*, 65:51–80, 1988.

[**Lig75**] G. Ligozat. Courbes modulaires de genre 1. *Bull. Soc. Math. France (Suppl.), Memoire 43*, 1975.

[**Ogg74**] A. Ogg. Hyperelliptic modular curves. *Bull. Soc. Math. France*, 228:449–462, 1974.

136 CONGRUENCE SUBGROUPS OF $\mathrm{PSL}_2(\mathbf{R})$

Chapter 136
CONGRUENCE SUBGROUPS OF $\mathrm{PSL}_2(\mathbf{R})$

136.1 Introduction

The group $GL_2^+(\mathbf{R})$ of 2 by 2 matrices defined over $\mathbf{R}$ with positive determinant acts on the upper half complex plane $\mathbf{H} = \{x \in \mathbf{C} | \mathrm{Im(x)} > 0\}$ by fractional linear transformation:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} : z \mapsto \frac{az+b}{cz+d}.$$

Any subgroup Γ of $GL_2^+(\mathbf{R})$ also acts on $\mathbf{H}$. To compactify the quotient $\Gamma\backslash\mathbf{H}$ it is necessary to adjoin the cusps, which are points in $\mathbf{P}^1(\mathbf{Q}) = \mathbf{Q} \cup \{\infty\}$. We set $\mathbf{H}^* := \mathbf{H} \cup \mathbf{P}^1(\mathbf{Q})$.

A fundamental domain for the action of Γ is a region of $\mathbf{H}^*$ containing a representative of each orbit of the action. The fundamental domain is most useful when it can be taken to be compact with finite area under the hyperbolic metric on $\mathbf{H}$. This is the situation for congruence subgroups.

This chapter gives a description of how to work in Magma with $\mathbf{H}^*$ and with congruence subgroups and their action on $\mathbf{H}^*$. This allows the computation of generators for congruence subgroups, and various other information, such as coset representatives. Farey symbols are used for this computation, and can be computed and manipulated by the user. Procedures are provided for producing graphical images, in the form of postscript files, which illustrate the fundamental domains. These procedures can be used to draw geodesics and polygons in the upper half plane.

This package has been written by Helena Verrill, and is partly based on a java program for drawing fundamental domains ([Ver00]). The Farey sequence algorithm used here is that described by Kulkarni [Kul91].

Example H136E1

An example of what can be computed with this package.

```
> G := CongruenceSubgroup(0,35);
> G;
Gamma_0(35)
> Generators(G);
[
    [1 1]
    [0 1],
    [ 11  -3]
    [ 70 -19],
    [ 13  -8]
    [ 70 -43],
    [  8  -3]
```

```
    [ 35 -13],
    [ 29 -21]
    [105 -76],
    [ 11  -6]
    [ 35 -19],
    [ 29 -17]
    [ 70 -41],
    [ 16 -11]
    [ 35 -24],
    [ 22 -17]
    [ 35 -27]
]
> C := CosetRepresentatives(G);
> H := UpperHalfPlaneWithCusps();
> triangle := [H|Infinity(),H.1,H.2];
> triangle_translates := [g*triangle : g in C];
>  DisplayPolygons(triangle_translates,"/tmp/Gamma035.ps": Show := false);
[ 0.E-57, 1.0000000000000000000000000000, 1.5000000000000000000000000000, 300 ]
```

In the example the command `DisplayPolygons` produces a postscript file "/tmp/Gamma035.ps" of a drawing of all the translates of the given triangle under the cosets of $\Gamma_0(35)$. If the `Show` option is set to `true`, a system command is issued and a window pops up for viewing the file created. Continuing with the above example, one can also find the vertices of a polygon defining the Fundamental domain of the group:

```
> FD := FundamentalDomain(G);
> // Print a FD in a reasonable format:
> &cat[Sprintf(" %o ",c) : c in FD];
 oo  0  1/6  1/5  1/4  3/11  2/7  1/3  2/5  3/7  1/2  4/7  3/5  5/8  2/3  5/7
3/4  1  oo
>  DisplayPolygons([FD],"/tmp/Gamma0_35_Domain.ps": Show := false);
[ 0.E-28, 1.0000000000000000000000000000, 1.5000000000000000000000000000, 300 ]
```

136.2 Congruence Subgroups

We denote by $\mathrm{SL}_2(\mathbf{Z})$ the group of 2 by 2 matrices with integer coefficients and determinant 1. The group $\mathrm{PSL}_2(\mathbf{Z})$ is the projectivization of $\mathrm{SL}_2(\mathbf{Z})$. For any integer N we have groups

$$\Gamma_0(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbf{Z}) \;\middle|\; \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv \begin{pmatrix} * & * \\ 0 & * \end{pmatrix} \bmod N \right\}$$

$$\Gamma_1(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbf{Z}) \;\middle|\; \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv \begin{pmatrix} 1 & * \\ 0 & 1 \end{pmatrix} \bmod N \right\}$$

$$\Gamma(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbf{Z}) \;\middle|\; \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \bmod N \right\}$$

$$\Gamma^1(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbf{Z}) \;\middle|\; \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 \\ * & 1 \end{pmatrix} \bmod N \right\}$$

$$\Gamma^0(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbf{Z}) \;\middle|\; \begin{pmatrix} a & b \\ c & d \end{pmatrix} \equiv \begin{pmatrix} * & 0 \\ * & * \end{pmatrix} \bmod N \right\}$$

A *congruence subgroup* is any discrete subgroup Γ of $\mathrm{SL}_2(\mathbf{R})$ which is commensurable with $\mathrm{SL}_2(\mathbf{Z})$, that is, $\Gamma \cap \mathrm{SL}_2(\mathbf{Z})$ has finite index in Γ and in $\mathrm{SL}_2(\mathbf{Z})$, and such that $\Gamma(N)$ is contained in G for some N. The *level* N of a congruence subgroup G is the greatest integer N such that $\Gamma(N)$ is contained in Γ. We will abuse notation and also refer to the projectivizations of these groups by the same names.

136.2.1 Creation of Subgroups of $\mathrm{PSL}_2(\mathbf{R})$

`PSL2(R)`

Returns $\mathrm{PSL}_2(R)$, the projective linear group over the ring R.

`Gamma0(N)`

The group $\Gamma_0(N)$ for any positive integer N.

`Gamma1(N)`

The group $\Gamma_1(N)$ for any positive integer N.

`GammaUpper0(N)`

The group $\Gamma^0(N)$ for any positive integer N.

`GammaUpper1(N)`

The group $\Gamma^1(N)$ for any positive integer N.

`CongruenceSubgroup(N)`

The group $\Gamma(N)$ for any positive integer N.

`CongruenceSubgroup(i,N)`

For a positive integer N and $i = 0, 1, 2, 3$, or 4, this is the group $\Gamma_0(N)$, $\Gamma_1(N)$, $\Gamma(N)$, $\Gamma^1(N)$ or $\Gamma^0(N)$ respectively.

`CongruenceSubgroup([N,M,P])`

This is the congruence subgroup consisting of 2 by 2 matrices with integer coefficients $[a, b, c, d]$ with $b = 0 \pmod P$, $c = 0 \pmod N$, and $a = d = 1 \pmod M$. It is required that M divides NP.

`Intersection(G,H)`

`G meet H`

The intersection of congruence subgroups G and H.

Example H136E2

Examples of defining different congruence subgroups:

```
> G := PSL2(Integers());
> H := CongruenceSubgroup([2,3,6]);
> H;
Gamma_0(2) intersection Gamma^1(3) intersection Gamma^0(2)
> K := CongruenceSubgroup(0,5);
> K meet H;
Gamma_0(10) intersection Gamma^1(3) intersection Gamma^0(2)
```

136.2.2 Relations

`G eq H`

Returns `true` if and only if the congruence subgroups G and H are equal.

`H subset G`

For congruence subgroups G and H contained in $\mathrm{PSL}_2(\mathbf{Z})$, returns `true` if and only if H is a subgroup of G.

`Index(G,H)`

For congruence subgroups G and H, returns the index of G in H provided G is a subgroup of H.

`Index(G)`

For G a congruence subgroup in $\mathrm{PSL}_2(\mathbf{Z})$, returns the index in $\mathrm{PSL}_2(\mathbf{Z})$.

136.2.3 Basic Attributes

`Level(G)`

The level of a congruence subgroup G.

`IsCongruence(G)`

Returns `true` if and only if G is a congruence subgroup.

`IsGamma0(G)`

Returns `true` if and only if G is equal to $\Gamma_0(N)$ for some integer N.

`IsGamma1(G)`

Returns `true` if and only if G is equal to $\Gamma_1(N)$ for some integer N.

`BaseRing(G)`

Returns the base ring over which matrices of the congruence subgroup G are defined.

`Identity(G)`

Returns the identity matrix in the congruence subgroup G.

136.3 Structure of Congruence Subgroups

`CosetRepresentatives(G)`

If G is a subgroup of finite index in $\mathrm{PSL}_2(\mathbf{Z})$, then returns a sequence of coset representatives of G in $\mathrm{PSL}_2(\mathbf{Z})$.

`Generators(G)`

Returns a sequence of generators of the congruence subgroup G.

`FindWord(G,g)`

For a congruence subgroup G, and an element g of G, this function returns a sequence of integers corresponding to an expression for g in terms of a fixed set of generators for G. Let L be the list of generators for G output by the function `Generators`. Then the return sequence $[e_1n_1, e_2n_2, \ldots, e_mn_m]$, where n_i are positive integers, and $e_i = 1$ or -1, means that $g = L[n_1]^{e_1}L[n_2]^{e_2}\ldots L[n_m]^{e_m}$. Note that since the computation is in $PSL_2(\mathbf{R})$, this equality only holds up to multiplication by ± 1.

`Genus(G)`

The genus of the upper half plane quotiented by the congruence subgroup G.

`FundamentalDomain(G)`

For G a subgroup of $\mathrm{PSL}_2(\mathbf{Z})$ returns a sequence of points in the Upper Half plane which are the vertices of a fundamental domain for G.

Example H136E3

In this example we compute a set of generators for $\Gamma_0(12)$.

```
> G := CongruenceSubgroup(0,12);
> Generators(G);
[
    [1 1]
    [0 1],
    [ 5 -1]
    [36 -7],
    [  5  -4]
    [ 24 -19],
    [  7  -5]
    [ 24 -17],
    [ 5 -3]
    [12 -7]
]
> C := CosetRepresentatives(G);
> H<i,r> := UpperHalfPlaneWithCusps();
> triangle := [H|Infinity(),r,r-1];
> translates := [g*triangle : g in C];
```

Example H136E4

This example illustrates how any element of a congruence subgroup can be written in terms of the set of generators output by the `generators` function.

```
> N := 34;
> Chi := DirichletGroup(N, CyclotomicField(EulerPhi(N)));
> GaloisConjugacyRepresentatives(Chi);
[
    1,
    $.1,
    $.1^2,
    $.1^4,
    $.1^8
]
> char := Chi.1^8;
> G := CongruenceSubgroup([N,Conductor(char),1],char);
> G;
Gamma_0(2) intersection Gamma_1(17) with character $.1^8
> gens := Generators(G);
> #gens;
21
> g := G! [21, 4, 68, 13];
> // express g in terms of Generators(G)
> FindWord(G, g);
[ -8, 1 ]
> // This means that up to sign, g = gens[8]^(-1) * gens[1]
> gens[8]^(-1) * gens[1];
[-21  -4]
[-68 -13]
```

136.3.1 Cusps and Elliptic Points of Congruence Subgroups

`Cusps(G)`

Returns a sequence of inequivalent cusps of the congruence subgroup G.

`CuspWidth(G,x)`

Returns the width of x as a cusp of the congruence subgroup G.

`EllipticPoints(G)`

`EllipticPoints(G,H)`

Returns a list of inequivalent elliptic points for the congruence subgroup G. A second argument may be given to specify the upper half plane H containing these elliptic points.

Example H136E5

We can compute a set of representative cusps for $\Gamma_1(12)$, and their widths as follows:

```
> G := CongruenceSubgroup(0,12);
> Cusps(G);
[
    oo,
    0,
    1/6,
    1/4,
    1/3,
    1/2
]
> Widths(G);
[ 1, 12, 1, 3, 4, 3 ]
> // Note that the sum of the cusp widths is the same as the Index:
> &+Widths(G);
24
> Index(G);
24
```

In the following example we find which group $\Gamma_0(N)$ has the most elliptic points for N less than 20, and list the elliptic points in this case.

```
> H := UpperHalfPlaneWithCusps();
> [#EllipticPoints(Gamma0(N),H) : N in [1..20]];
[ 2, 1, 1, 0, 2, 0, 2, 0, 0, 2, 0, 0, 4, 0, 0, 0, 2, 0, 2, 0 ]
> // find the index where the maximal number of elliptic points is attained:
> Max($1);
4 13
> // find the elliptic points for Gamma0(13):
> EllipticPoints(Gamma0(13));
[
    5/13 + (1/13)*root(-1),
    8/13 + (1/13)*root(-1),
    7/26 + (1/26)*root(-3),
    19/26 + (1/26)*root(-3)
]
```

136.4 Elements of $\mathrm{PSL}_2(\mathbf{R})$

136.4.1 Creation

`G ! x`

If x is a sequence $x = [a, b, c, d]$ of elements in the base ring of G, this function returns $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$, provided this is an element of G. If x is an integer the identity matrix is returned. If x is a matrix, it is coerced into G if possible.

`Random(G,m)`

Returns a random element of the projective linear group G, with m determining the size of the coefficients.

136.4.2 Membership and Equality Testing

`g eq h`

For g and h elements of $\mathrm{PSL}_2(\mathbf{Z})$, returns `true` if g, h have compatible coefficient rings and if $g = h$, `false` otherwise. Since the group is projective, returns `true` if the matrices are equal up to a nonzero scalar multiple.

`IsEquivalent(g,h,G)`

For g and h elements of $\mathrm{PSL}_2(\mathbf{Z})$, returns `true` if g and h are defined of the same field, and if $Gg = Gh$, i.e. if $gh^{-1} \in G$.

`g in G`

For g an elements of $\mathrm{PSL}_2(\mathbf{Z})$, returns `true` if g is in the congruence subgroup G, `false` otherwise.

136.4.3 Basic Functions

For a matrix g in a congruence subgroup, and an integer n, returns g^n.

`Eltseq(g)`

Returns the sequence of four numbers which are the entries of the matrix g.

`g * h`

If g and h have the same parent then this returns their product.

`g ^ n`

For a matrix g and integer n returns g^n.

Example H136E6

Define congruence subgroups as in the following examples:

```
> // examples of defining matrix elements of congruence subgroups:
>
> G := PSL2(Integers());
> G![2,0,0,2];
[1 0]
[0 1]
> H := CongruenceSubgroup([2,3,6]);
> H![7,6,8,7];
[7 6]
[8 7]
```

136.5 The Upper Half Plane

The *upper half complex plane* is defined by $\mathbf{H} := \{z \in \mathbf{C} \mid \mathrm{Im(z)} > 0\}$. The group $\mathrm{SL}_2(\mathbf{Z})$ acts on H by fractional linear transformations. The space $\mathbf{H}/SL_2(\mathbf{Z})$ is not compact; it is compactified by adding the cusps, which are points of $\mathbf{Q}$, together with ∞. Thus we define $\mathbf{H}^*$ to be the upper half plane union the cusps. Then $\mathbf{H}^*/\mathrm{SL}_2(\mathbf{Z})$ is compact. Thus we define a function which will return the space of points in the upper half complex plane, together with the set of cusps.

In $\mathbf{H}^*$ we define two distinguished points, the elliptic points $\sqrt{-1}$ and $(1+\sqrt{-3})/2$. In general, points constructed in $\mathbf{H}^*$ are allowed to come from at most quadratic extensions of $\mathbf{Q}$, since in this case there is a canonical embedding in $\mathbf{C}$.

136.5.1 Creation

`UpperHalfPlane()`

Creates a copy of the upper half complex plane, with the cusps included. As a set this consists of all complex numbers with positive imaginary part, together with all rational numbers, and the point at infinity.

`H ! x`

Returns x as a point in $\mathbf{H}$. Here x can be a cusp, rational, integer, in a quadratic extension of $\mathbf{Q}$, or a complex number with positive imaginary part.

Example H136E7

Example of creating some points in the upper half plane.

```
> H := UpperHalfPlaneWithCusps();
> // coerce a cusp into H:
> c := Cusps()!(1/2);
> H!c;
1/2
> // coerce an element of a quadratic extension of Q into H
> K := QuadraticField(-7);
> K<u> := QuadraticField(-7);
> H!(u+5);
5 + root(-7)
> // refer to the two distinguished elliptic points:
> H.1;
root(-1)
> H.2;
1/2 + (1/2)*root(-3)
> // Defining the names of the elliptic points when constructing H:
> H<i,rho> := UpperHalfPlaneWithCusps();
> i;
root(-1)
> rho;
1/2 + (1/2)*root(-3)
```

136.5.2 Basic Attributes

`Imaginary(z)`

Returns the imaginary part of the argument as an element of `RealField`.

`Real(z)`

Returns the real part of the argument as an element of `RealField`.

`IsReal(z)`

Returns `true` if and only if the element z of the upper half plane lies on the real line (and is not the infinite cusp).

`IsCusp(z)`

Returns `true` if and only if the element z of the upper half plane is a cusp.

`IsInfinite(z)`

Returns `true` if and only if the element z of the upper half plane is the cusp at infinity.

`IsExact(z)`

Returns `true` if and only if the element z of the upper half plane is a cusp or has an exact value defined in a quadratic extension of the rationals.

`ExactValue(z)`

For x an element of the upper half plane, if x is a cusp, returns the value of x as an object of type `SetCspElt`; if x has an exact value in a quadratic extension, returns this value, as an object of type `FldQuadElt`; otherwise returns a complex value of type `FldComElt`.

`ComplexValue(x)`

`Precision`	RngIntElt	*Default :*
`MaxValue`	RngIntElt	*Default :* 600

For x an element of the upper half place, this returns x as a complex number. When x is the cusp at infinity, the value returned is `MaxValue + i*MaxValue`.

`x eq y`

Returns `true` if and only if the points x and y in the upper half plane are equal.

136.6 Action of $\mathrm{PSL}_2(\mathbf{R})$ on the Upper Half Plane

`g * z`

For z of type `SpcHypElt, SetCspElt` or `[SpcHypElt]`, and when g is an element of a projective linear group, returns the image of z under the action of g. The type of the image is the same as the type of z. If g is a positive integer, returns return az, which his is equivalent to acting on z with the matrix $\begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix} \in \mathrm{PGL}_2(\mathbf{R})$.

`FixedPoints(g,H)`

Returns a sequence of points in H fixed by the action of g.

`IsEquivalent(G,a,b)`

If points a, b in the upper half plane are equivalent under the action of the group G, returns `true`, and the matrix g in G such that $g \cdot a = b$. Otherwise returns `false` and the identity.

`EquivalentPoint(x)`

For the point x in the upper half plane, returns a point z in the region with $-1/2 < z \leq 1/2$ and $|z| \geq 1$, and a matrix g in $\mathrm{PSL}_2(\mathbf{Z})$ with g*x = z

`Stabilizer(a,G)`

Returns a generator of the subgroup of G stabilizing a.

`FixedArc(g,H)`

If g is an element of $\mathrm{PSL}_2(\mathbf{Z})$ which is an involution, this returns the end points in the real line of the arc fixed by g, with mid point of the arc also fixed by g. Note that for any point b, the arc from b to $g \cdot b$ is fixed by g.

136.6.1 Arithmetic

`z + a`
`z - a`

For any integer a, and element z in the upper half plane, this returns the element $z + a$ in the same copy of the upper half plane.

`a * z`
`z * a`
`a * seq`
`z / a`

Given an element z (or a sequence of elements) in the upper half plane, and a positive rational number a, this returns the product (or products) in the same copy of the upper half plane.

136.6.2 Distances, Angles and Geodesics

`Distance(z,w)`

`Precision` RngIntElt *Default :*

Returns the hyperbolic distance between z and w.

`TangentAngle(x,y)`

`Precision` RngIntElt *Default :*

Returns the angle of the tangent at x of the geodescic from x to y, with given precision.

`Angle(e1,e2)`

`Precision` RngIntElt *Default :*

Given two sequences $e_1 = [z_1, z_2]$ and $e_2 = [z_1, z_3]$, where $z1, z2, z3$ are elements of the upper half plane, this returns the angle between the geodesics at z_1.

`ExtendGeodesic([z1,z2], H)`

Given elements $z1, z2$ in the upper half plane H, this extends the geodesic between $z1$ and $z2$ to a semicircle with endpoints on the real line, and returns the two real endpoints as elements of H.

`GeodesicsIntersection(x1,x2)`

`GeodesicsIntersection(x1,x2)`

The intersection in the upper half plane of the two geodesics whose endpoints are given by the sequences x_1 and x_2. If the geodesics intersect along a line, the empty sequence is returned.

136.7 Farey Symbols and Fundamental Domains

One method of finding fundamental domains for congruence subgroups is the method of Farey Symbols, as described by Kulkarni [Kul91].

A *generalized Farey sequence* is a sequence of rationals

$$\frac{a_1}{b_1} < \frac{a_2}{b_2} < \ldots < \frac{a_n}{b_n}$$

such that for a consecutive pair of fractions $\frac{b}{d}, \frac{a}{c}$ in the sequence, written in lowest terms, we have $ad-bc = 1$. We extend the rationals to $\mathbf{Q} \cup \{-\infty, \infty\}$, where we use the convention $-\infty = \frac{-1}{0}$ and $\infty = \frac{1}{0}$.

A *Farey Symbol* is a Farey sequence of length n starting with $\frac{-1}{0}$, and ending with $\frac{1}{0}$, together with a sequence of $n-1$ labels. We use the convention that labels can be any elements of $\mathbf{N}_{>0} \cup \{-2, -3\}$. The sequence of labels must satisfy the condition that each element of $\mathbf{N}_{>0}$ appears in the sequence either exactly twice or not at all. For example, the sequences $\left[\frac{-1}{0}, \frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{1}{1}, \frac{1}{0}\right]$ and $[1, 2, 2, -3, -2, -2, 1]$ define a Farey symbol, which is generally written in the following format:

$$\left\{ \frac{-1}{0} \overset{1}{,} \frac{0}{1} \overset{2}{,} \frac{1}{2} \overset{2}{,} \frac{2}{3} \overset{-3}{,} \frac{5}{7} \overset{-2}{,} \frac{3}{4} \overset{-3}{,} \frac{1}{1} \overset{1}{,} \frac{1}{0} \right\}.$$

Farey symbols are used to define certain fundamental domains for congruence subgroups of $\mathrm{PSL}_2(\mathbf{Z})$. The sequence of fractions gives cusps which are vertices of the domain, and the labels give edge identifications. For a_i, a_{i+1} in the Farey sequence, with corresponding label l_i not -3, the corresponding edge of the domain is a geodesic between a_i and a_{i+1}. If the label is -3, there is an extra elliptic point of order 3 on the boundary of the domain between the two cusps, and the two edges between these cusps are identified. The label $l_i = -2$ indicates an elliptic point of order 2 on the boundary between the two cusps a_i and a_{i+1}. This point is on the geodesic between a_i and a_{i+1}, and the two halves of the geodesic are identified.

`FareySymbol(G)`

Computes the Farey Symbol of a congruence subgroup G in $\mathrm{PSL}_2(\mathbf{Z})$.

`Cusps(FS)`

Returns the cusp sequence of the Farey symbol FS. Note, this is not a sequence of inequivalent cusps of the corresponding group.

`Labels(FS)`

Returns the sequence of edge labels of a Farey symbol FS.

`Generators(FS)`

Returns the generators of the congruence subgroup corresponding to the Farey symbol FS.

`Group(FS)`

Returns the congruence subgroup corresponding to the Farey Symbol FS.

`Widths(FS)`

Returns the sequence of integers giving twice the widths of the cusp list of the Farey symbol FS.

`Index(FS)`

Returns the index of `Group(FS)` in $\mathrm{PSL}_2(\mathbf{Z})$.

`FundamentalDomain(FS)`

`FundamentalDomain(FS,H)`

Returns the vertices in the upper half plane of the fundamental domain described by the Farey Sequence FS. A second argument may be given to specify the upper half plane H.

`CosetRepresentatives(FS)`

Returns the coset representatives of the congruence subgroup of $\mathrm{PSL}_2(\mathbf{Z})$ corresponding to the Farey symbol FS.

`InternalEdges(FS)`

Returns a sequence of pairs of cusps which are cusps of the Farey Symbol FS, and which are not adjacent in FS but which are images of 0 and infinity under some matrix in $\mathrm{PSL}_2(\mathbf{Z})$.

136.8 Points and Geodesics

In this section we describe some functions involving points and geodesics. Geodesics in the upper half plane are given by circles or lines which intersect the real axes at right angles. A geodesic is defined by its two end points. Points and geodesics can be drawn using the graphics functions described in the next section.

`GeodesicsIntersection(x,y)`

Computes the intersection in the upper half plane of the two geodesics x, y, where x and y are specified by their end points, which must be cusps. If the geodesics intersect along a line the empty sequence is returned.

Example H136E8

In the following example we find and display the intersection points of some geodesics:

```
> H := UpperHalfPlaneWithCusps();
> g1 := [H|-1,2];
> g2 := [H|0,6];
> g3 := [H|1,5];
> g4 := [H|2,Infinity()];
> lines := [g1,g2,g3,g4];
> points := [GeodesicsIntersection(x,y,H) : x in lines,y in lines];
>  DisplayPolygons(lines cat points,"/tmp/pic.ps":
>  Labels := [-1,0,1,2,5,6],Colours := [1,0,0],Show := true);
[ -1.0000000000000000000000000000, 6.0000000000000000000000000000,
3.3284271247461900976033774484, 91 ]
```

136.9 Graphical Output

When working with a particular congruence subgroup it is often useful to be able to produce a picture of a fundamental domain of the group. Also when considering images of geodesics in $\mathbf{H}^*$ it can be useful to draw the images. The following graphics functions provide a useful tool for this purpose.

`DisplayPolygons(P,file)`

`Colours`	SeqEnum	*Default :* $[1,1,0]$
`Outline`	BoolElt	*Default :* `true`
`Fill`	BoolElt	*Default :* `true`
`Show`	BoolElt	*Default :* `false`
`Labels`	SeqEnum	*Default :* $[0,1]$
`Fontsize`	RngIntElt	*Default :* 2
`Size`	SeqEnum	*Default :* [ ]
`Pixels`	RngIntElt	*Default :* 300

`Overwrite`	BoolElt	*Default :* `false`
`Radius`	FldReElt	*Default :* 0.5
`PenColours`	SeEnum	*Default :* $[0, 0, 0]$

Given a sequence of polygons, each of which is defined by a sequence of points in the upper half plane, produce the postscript drawing of these polygons, and write the result to the named file. The function returns a sequence of 4 real numbers, `[x0,x1,h,S]`, where `(x0,0)` are the coordinates of the lower left corner, `(x1,h)` are the coordinates of the upper right corner, and the scale is such that there are `S` pixels per unit. These values are either given by the user, or computed automatically.

`Colours` : Either a sequence of 3 numbers, or a sequence of such sequences, one for each polygon in the sequence of polygons given. A colour is represented by 3 real numbers between 0 and 1, giving the red, green and blue components of the colour. This colour is used for the filling of the polygon.

`PenColours` : A sequence of 3 numbers, or a sequence of such sequences, one for each polygon in the sequence of polygons given. A colour is represented by 3 real numbers between 0 and 1, giving the red, green and blue components of the colour. This colour is used for drawing the outline of the polygon.

`Labels` : A sequence of elements of type `SetCspElt`, `FldRatElt` or `RngIntElt`, which will be labeled on the real axis in the diagram.

`Fontsize` : The size of the font used in labeling the diagram.

`Pixels` : When using Autoscale, the scale is computed so that the width of the resulting diagram is given by `Pixels` in pixels, plus a 20 pixel border. The minimal possible value for `Pixels` is 10, and if any smaller value is given it will be set to 10.

`Size` : If `Size` is not the empty sequence, the values in the sequence `Size` are used to determine the size of the image. When `Size` is given by `[x0,x1,y,S]`, the picture is draw with coordinates `(x0,0)` in the lower left corner, `(x1,h)` in the upper right corner, and the scale is such that there are `S` pixels per unit. If `Size` is not given, then an appropriate size for the image is computed automatically.

`Overwrite` : If `Overwrite` is set to `true`, then if the name of the file given names a file which already exists, it will be overwritten. Otherwise the user will be asked whether they want to overwrite the file or not.

`Show` : If `Show` is set to `true` a System command is issued to make a window pop up showing the file just created. The system command is `System("gv file &")`. In future other options may be possible. If `Show` is `false` then

`Fill` : A boolean or sequence of booleans, determining whether the polygon is drawn filled in with a colour, the colour coming from the sequence

`Colours`. If both `Fill` and `Outline` are set to `false` outline is reset to `true`.

`Outline` : A boolean or sequence of booleans, determining whether the polygon is drawn with an outline. If both `Fill` and `Outline` are set to `false` outline is reset to `true`.

`Radius` : A real number giving the radius of points marked on the diagram. A point will be marked for any polygon given by a single point.

`DisplayFareySymbolDomain(FS,file)`

`DisplayFareySymbolDomain(G,file)`

`Colour`	SeqEnum	*Default :* $[1, 1, 0]$
`Show`	BoolElt	*Default :* `false`
`Fontsize`	RngIntElt	*Default :* 2
`Labelsize`	RngIntElt	*Default :* 3
`Autoscale`	BoolElt	*Default :* `true`
`Size`	SeqEnum	*Default :* []
`Pixels`	RngIntElt	*Default :* 300
`Overwrite`	BoolElt	*Default :* `false`
`ShowInternalEdges`	BoolElt	*Default :* `false`

`Display` : a fundamental domain corresponding to the Farey Symbol FS, or a group G, for which the Farey symbol will be computed, with edge identifications and cusps labeled. This function returns a sequence of 4 real numbers, `[x0,x1,h,S]`, where `(x0,0)` are the coordinates of the lower left corner, `(x1,h)` are the coordinates of the upper right corner, and the scale is such that there are `S` pixels per unit. These values are either given by the user, or computed automatically. A polygon can be a sequence of any number of points of type `HypSpcElt` or `SetCspElt`. When at least 3 points are given a polygon is drawn. For two points, a geodesic between them is drawn. For one point a small circle is drawn to mark the point.

`Colour` : This colour is used in colouring the domain drawn. A colour is represented by a sequence of 3 real numbers between 0 and 1, giving the red, green and blue components of the colour.

`Fontsize` : The size of the font used in labeling the cusps on the diagram.

`Labelsize` : The size of the font used in labeling the identification labels on the diagram.

`Autoscale` : If `true`, the scale is computed and the file is created so that all the polygons listed can be seen in the diagram, if this is feasible. if `Autoscale` is set to `false`, then the variable `Size` must be given.

`Pixels` : When using `Autoscale`, the scale is computed so that the width of the resulting diagram is given by `Pixels` in pixels, plus a 20 pixel border. The minimal possible value for `Pixels` is 10, and if any smaller value is given it will be set to 10.

`Size` : When `Autoscale` is `false`, the values in the sequence `Size` are used to determine the size of the image. When `Size` is given by `[x0,x1,y,S]`, the picture is draw with coordinates `(x0,0)` in the lower left corner, `(x1,h)` in the upper right corner, and the scale is such that there are `S` pixels per unit.

`Overwrite` : If `Overwrite` is set to `true`, then if the name of the file given names a file which already exists, it will be overwritten. Otherwise the user will be asked whether they want to overwrite the file or not.

`Show` : If `Show` is set to `true` a System command is issued to make a window pop up showing the file just created. The system command is `System("gv file &")`. In future other options may be possible. If `Show` is `false` then

`ShowInternalEdges` : If this is set to `true` then internal edges are drawn dividing the domain into fundamental domains for $\Gamma_0(2)$.

Example H136E9

In the first example we draw a fundamental domain for $\Gamma_0(12)$.

```
> G := Gamma0(24);
> FS := FareySymbol(G);
> C := Cusps(FS);
> Domain := FundamentalDomain(G);
> red := [1,0,0];
>  DisplayPolygons(Domain,"/tmp/Gamma0N12.ps":
>      Colours := red, Labels := C, Show := true, Fontsize := 1);
[ 0.E-28, 1.0000000000000000000000000000, 1.5000000000000000000000000000, 300 ]
```

The user could write a procedure as follows for drawing the fundamental domain as a union of fundamental domains for $\mathrm{PSL}_2(\mathbf{Z})$.

```
> procedure draw_fundamental_domain(Group,file)
>     cosets := CosetRepresentatives(Group);
>     H<i,r> := UpperHalfPlaneWithCusps();
>     tri := [H|Infinity(),0,r];
```

```
>     red := [1,0,0];
>     yellow := [1,1,0];
>     cyan := [0,1,1];
>     cols := &cat[[red,yellow,cyan] : i in [1..(Ceiling(#cosets/3))]];
>     trans := [g*tri : g in cosets];
>     Outlines := [false : g in cosets];
>     DisplayPolygons(trans,file:
>         Colours := cols, Outline := Outlines, Show := true);
> end procedure;
>
> // use the procedure to draw a domain for Gamma_0(19):
>  draw_fundamental_domain(Gamma0(19),"/tmp/gamma019.ps");
[ 0.E-28, 1.0000000000000000000000000000, 1.5000000000000000000000000000, 300 ]
```

In the following example, we use the cosets of $\Gamma_0(11)$ to draw a fundamental domain for $\Gamma_0(11)$, divided up into fundamental domains for the group of automorphisms of $\mathbf{H}^*$ generated by $\mathrm{PSL}_2(\mathbf{Z})$ together with the map $z \mapsto -\overline{z}$.

```
> G := Gamma0(11);
> H<i,rho> := UpperHalfPlaneWithCusps();
> tri := [H|Infinity(),i,rho];
> tri1 := [H|0,i,rho];
> C11 := CosetRepresentatives(G);
> Colours := [[0.6,0.2,0.9] : i in [1..#C11]] cat [[1,1,0] : i in [1..#C11]];
> triangles := [g*tri : g in C11] cat [g*tri1 : g in C11];
>  DisplayPolygons(triangles,"/tmp/Gamma_0_11.ps":
>      Colours := Colours, Show := false);
[ 0.E-57, 1.0000000000000000000000000000, 1.5000000000000000000000000000, 300 ]
```

Continuing from the above example, we can take any other polygon we wish, and look at it's translates under the action of the cosets of $\Gamma_0(11)$.

```
> tri2 := [H|Infinity(),2*i + 1,i,rho];
>  DisplayPolygons([g*tri2 : g in C11],"/tmp/pic.ps":
>      Colours := Colours, Show := true);
[ 0.E-57, 1.0000000000000000000000000000, 2.5000000000000000000000000000, 300 ]
```

In the next example we define three different triangles. Each of these is a fundamental domain for $\Gamma^0(2)$, though MAGMA currently does not verify this. For each choice we use a set of generators of $\Gamma^0(2)$ to draw a partial tiling of the upper half plane with equivalent fundamental domains.

```
> G := CongruenceSubgroup(4,2);
> H<i,r> := UpperHalfPlaneWithCusps();
> generators := Generators(G);
> M := generators cat [g^(-1) : g in generators] cat [G!1];
> tri1 := [H|Infinity(),r-1,0,r,r+1];
> tri2 := [H|Infinity(),i-1,0,i+1];
> tri3 := FundamentalDomain(G);
> X := [1..#M];
> L := [G!m : m in Set([Matrix(M[i]*M[j]*M[k]) : i in X, j in X, k in X])];
>  DisplayPolygons([g*tri1 : g in L],"/tmp/picture1.ps": Show := true);
```

```
[ -4.000000000000000000000000000, 3.500000000000000000000000000,
1.500000000000000000000000000, 200 ]
>  DisplayPolygons([g*tri2 : g in L],"/tmp/picture2.ps": Show := true);
[ -4.000000000000000000000000000, 3.000000000000000000000000000,
1.500000000000000000000000000, 200 ]
>  DisplayPolygons([g*tri3 : g in L],"/tmp/picture3.ps": Show := true);
[ -4.000000000000000000000000000, 3.000000000000000000000000000,
1.500000000000000000000000000, 200 ]
```

Edges and Polygons with different numbers of sides can be included in the same picture, as in the following example.

```
> H<i,r> := UpperHalfPlaneWithCusps();
> cosets := CosetRepresentatives(Gamma0(19));
> polygons := [g*tri : g in cosets, tri in [[H|i,r],[H|i,2*r,3*r-1]]];
> outlines := [true : i in cosets] cat [false : i in cosets];
> cyan := [0,1,1];
>  DisplayPolygons(polygons,"/tmp/pic.ps":
>      Colours := cyan, Outline := outlines, Labels := [], Show := false);
[ 0.E-57, 1.000000000000000000000000000, 3.098076211353315940291169512, 300 ]
```

In the following example a function is defined to determine the colouring of the polygons in terms of which Farey sequence pairs of end points of the polygons belong to.

```
> frac := func<a | a[1]/a[2]>;
>
> function FareyValue(m)
>   mat := Matrix(m);
>   Denominators := [mat[2,1],mat[2,2],mat[2,1]+mat[2,2]];
>   values := [Abs(v) : v in Denominators];
>   return &+ContinuedFraction(frac(Sort(values)));
> end function;
>
> procedure drawDomain(cosets)
>    H<i,r> := UpperHalfPlaneWithCusps();
>    tri := [H|Infinity(),0,r];
>    cols := [[0.5,0.2*FareyValue(c),1-0.2*FareyValue(c)] : c  in cosets];
>    trans := [g*tri : g in cosets];
>    DisplayPolygons(trans,"/tmp/pic.ps":
>        Outline := false, Colours := cols, Show := true);
> end procedure;
>
> // Now we can use the above functions and procedure, for example
> // take the cosets as follows:
> C := CosetRepresentatives(Gamma0(41));
> // Then draw the picture:
>  drawDomain(C);
[ 0.E-28, 1.000000000000000000000000000, 1.500000000000000000000000000, 300 ]
```

Example H136E10

Functions are provided for drawing pictures coming from Farey symbols and fundamental domains. These functions provide short cuts for producing some of the pictures in the previous examples, but are not so flexible.

```
> G := CongruenceSubgroup(5);
> FS := FareySymbol(G);
> FS;
[ 2, 2, 7, 8, 8, 10, 9, 3, 3, 9, 4, 1, 1, 4, 5, 5, 10, 11, 11, 7, 6, 6 ]
[ oo, 0, 1/5, 2/9, 1/4, 3/11, 5/18, 2/7, 1/3, 3/8, 5/13, 2/5, 1/2, 3/5, 5/8, 2/3,
5/7, 8/11, 3/4, 7/9, 4/5, 1, oo ]
> // The following command graphically displays the information contained in FS:
>  DisplayFareySymbolDomain(FS,"/tmp/Gamma5b.ps": Show := false);
[ 0, 1.0000, 0.50000, 800 ]
> // This picture represents a modular curve with genus 0:
> Genus(G);
0
```

Example H136E11

In the following example we show how to create a postscript file and include the resulting file in a latex file.
The example in question is $\Gamma_0(37)$. A simple question is to find what is the fundamental domain, and to give a list of inequivalent elliptic points of the group, and display this information graphically. First we create the group and find the information we are interested in:

```
> G := Gamma0(37);
> // To draw a picture of the fundamental domain we need to define
> // the upper half plane.
> H<i,r> := UpperHalfPlaneWithCusps();
> D := FundamentalDomain(G,H);
> E := EllipticPoints(G,H);
```

We can now take a look at the picture:

```
> // We need to make sure the polygons we want to display all
> // have the same type, so we have t create the following object:
> HH:=Parent(D);
> // now we make a list of polygons and points:
> P1:=[HH|D] cat [HH|[e] : e in E];
> // we take a look at the default picture of the situation:
>  DisplayPolygons(P1,"/tmp/pic.ps": Show := true);
[ 0.E-28, 1.0000, 1.5000, 300 ]
```

The points have been displayed in yellow, the default colour, but we'd prefer a different colour. Also the picture is not quite the right size. It has been shown with the real axis shown from 0 to 1, which is reasonable, but we'd prefer less height, and the scale 300 could be changed, as in this example:

```
> // use different colours:
```

```
> yellow := [1,1,0];
> red := [1,0,0];
> green := [0,1,0];
> cyan := [0,1,1];
> black := [0,0,0];
> Colours := [yellow] cat [red : e in E];
> // use a different size and scale:
> Size := [0,1,0.2,400];
> // view the result:
>  DisplayPolygons(P1,"/tmp/pic.ps":
>      Show := true, Colours := Colours, Size := Size);
[ 0, 1, 0.19999999999999999999999999999, 400 ]
```

After more experimenting we reach the following:

```
> // We use a slightly different list of polygons, to show more
> // structure.
> tri := [H|Infinity(),0,H.2];
> cosets := CosetRepresentatives(G);
> P := [HH|g*tri : g in cosets] cat [HH|[e] : e in E] cat [HH|D];
> // Choosing some colours:
> LotsOfColours := &cat[[yellow,green,cyan] : c in cosets];
> Colours := [LotsOfColours[i] : i in [1..#cosets]]
>            cat [black]
>            cat [red : e in E];
> // choose which polygons are filled, and which have outlines:
> outlines := [false : i in cosets] cat [false : i in E] cat [true];
> fill := [true : i in cosets] cat [true : i in E] cat [false];
> labels := Cusps(FareySymbol(G));
> // Create a file of the result:
>  DisplayPolygons(P,"/tmp/pic.ps":
>      Colours := Colours, Show := false, Size := [0,1,1,200],
>      Labels := labels, Outline := outlines, Fill := fill,
>      Fontsize := 1, Radius := 0.3);
[ 0, 1, 1, 200 ]
```

If we are satisfied with the resulting picture then we can include the resulting picture in a latex file as in the following example, where we use the Latex `graphicx` package to include the diagram.

```
\documentclass{article}
\usepackage[dvips]{graphicx}
\begin{document}
Here is a picture of a
fundamental domain for $\Gamma_0(37)$:
$${\includegraphics{/tmp/pic.ps}}$$
In the picture we have marked $4$ inequivalent
elliptic points.
```

```
\end{document}
```

Note that currently, given a list of points to draw, the postscript file will include all of the points, even if they are not in the area of the bounding box, which can produce bad results some times. This should be changed in a future version. Sometimes using the Fundamental domain function can produce results in a more reliable way, as in the following example, which produces a similar picture to the above, and a latex file can be written in the same way.

```
> G := Gamma0(37);
> H<i,r> := UpperHalfPlaneWithCusps();
> D := FundamentalDomain(G,H);
> HH:=Parent(D);
> E := EllipticPoints(G,H);
> P:=[HH|D] cat [HH|[e] : e in E];
> labels := Cusps(FareySymbol(G));
> yellow := [1,1,0];
> red := [1,0,0];
> Colours := [yellow] cat [red : e in E];
>  DisplayPolygons(P,"/tmp/pic.ps":
>      Colours := Colours, Show := false, Size := [0,1,0.2,200],
>      Labels := labels,
>      Fontsize := 1, Radius := 0.3);
[ 0, 1, 0.19999999999999999999999999999, 200 ]
```

136.10 Bibliography

[**Kul91**] Ravi S. Kulkarni. An arithmetic-geometric method in the study of the subgroups of the modular group. *Amer. J. Math.*, 113(6):1053–1133, 1991.

[**Ver00**] H. A. Verrill. Fundamental domain drawing program. URL:http://hverrill.net/fundomain/, 2000.

137 ARITHMETIC FUCHSIAN GROUPS AND SHIMURA CURVES

Chapter 137
ARITHMETIC FUCHSIAN GROUPS AND SHIMURA CURVES

In this chapter, we document algorithms for arithmetic Fuchsian groups. Let F be a totally real number field. Let A be a *quaternion algebra* over F, a central simple algebra of dimension 4 (see Chapter 91), such that A is split at exactly one real place, corresponding to the injective map $\iota_\infty : A \to M_2(\mathbf{R})$. Let $\mathcal{O}$ be a maximal order in A (see also Section 86.4), let $\mathcal{O}^*$ denote the group of elements of $\mathcal{O}$ of reduced norm 1, and let $\Gamma(1) = \iota_\infty(\mathcal{O}^*)/\{\pm 1\} \subset PSL_2(\mathbf{R})$. An *arithmetic Fuchsian group* Γ is a discrete subgroup of $PSL_2(\mathbf{R})$ which is commensurable with $\Gamma(1)$ (for some choice of F and A).

The classical case of the modular groups corresponds to $F = \mathbf{Q}$, $A = M_2(\mathbf{Q})$, $\mathcal{O} = M_2(\mathbf{Z})$, and $\Gamma(1) = \mathrm{PSL}_2(\mathbf{Z})$. Specialized algorithms for this case apply, and they are treated in detail in Chapter 136. To exclude this case, we assume throughout that A is a division ring, or equivalently $A \ncong M_2(\mathbf{Q})$.

The group Γ acts properly and discontinuously on the upper half-plane $\mathbf{H}$, and the quotient $\Gamma\backslash\mathbf{H} = X(\Gamma)$ can be given the structure of a compact Riemann surface, called a *Shimura curve*.

We exhibit methods for computing with arithmetic Fuchsian groups Γ, including the basic invariants of Γ and a fundamental domain for $X(\Gamma)$. We provide further specialized algorithms for triangle groups. Along the way, we also present an interface for computing with the unit disc, parallel to that for the upper half-plane (see Chapter 136).

The algorithm used to compute fundamental domains is described in [Voi09]. We recommend the following additional reading concerning the algorithms in this section. For an introduction to Fuchsian groups, see Katok [Kat92] and Beardon [Bea77], and for their relationship to quaternion algebras, see also Vignéras [Vig80]. For a computational perspective on arithmetic Fuchsian groups and Shimura curves, see Alsina-Bayer [AB04], Elkies [Elk98], and Voight [Voi05], and for a discussion of triangle groups, see Voight [Voi06].

137.1 Arithmetic Fuchsian Groups

In this section, we computing the basic invariants of an arithmetic Fuchsian group Γ.

137.1.1 Creation

We begin by giving the basic constructors for arithmetic Fuchsian groups.

`FuchsianGroup(O)`

Returns the arithmetic Fuchsian group Γ corresponding to the group $\mathcal{O}^*$ of units of reduced norm 1 in the quaternion order $\mathcal{O}$.

`FuchsianGroup(A)`

Returns the arithmetic Fuchsian group Γ corresponding to the group $\mathcal{O}^*$ of units of reduced norm 1 in a maximal order $\mathcal{O}$ in the quaternion algebra A.

`FuchsianGroup(A, N)`

Returns the arithmetic Fuchsian group Γ corresponding to the group $\mathcal{O}^*$ of units of reduced norm 1 in an order $\mathcal{O}$ of level $\mathfrak{N}$ in the quaternion algebra A.

Example H137E1

In this example, we construct arithmetic Fuchsian groups in three ways. First, we construct the group associated to the quaternion algebra of discriminant 6 over $\mathbf{Q}$.

```
> A := QuaternionAlgebra(6);
> G := FuchsianGroup(A);
> G;
Arithmetic Fuchsian group arising from order of Quaternion Algebra with base
ring Rational Field
> O := BaseRing(G);
> O;
Order of Quaternion Algebra with base ring Rational Field
with coefficient ring Integer Ring
> Discriminant(O);
6
> Algebra(O) eq A;
true
```

Next, we construct a group "by hand", associated to the quaternion algebra over the totally real subfield $F = \mathbf{Q}(\zeta_9)^+$ of $\mathbf{Q}(\zeta_9)$ ramified only at two of the three real infinite places.

```
> K<z> := CyclotomicField(9);
> F := sub<K | z+1/z >;
> Degree(F);
3
> Z_F := MaximalOrder(F);
> Foo := InfinitePlaces(F);
> A := QuaternionAlgebra(ideal<Z_F | 1>, Foo[2..3]);
> Discriminant(A);
Principal Ideal of Z_F
Generator:
    [1, 0, 0]
[ 2nd place at infinity, 3rd place at infinity ]
> O := MaximalOrder(A);
> G := FuchsianGroup(O);
> G;
Arithmetic Fuchsian group arising from order of Quaternion Algebra with base
ring Field of Fractions of Maximal Equation Order with defining polynomial x^3 -
```

```
3*x - 1 over its ground order
```

Lastly, we construct the group of level 3 inside a quaternion algebra ramified at $2\infty_1$ over $\mathbf{Q}(\sqrt{5})$.

```
> F<x> := NumberField(Polynomial([-5,0,1]));
> Z_F := MaximalOrder(F);
> A := QuaternionAlgebra(ideal<Z_F | 2>, InfinitePlaces(F)[1..1]);
> G := FuchsianGroup(A, ideal<Z_F | 3>);
> O := BaseRing(G);
> O;
Order of Quaternion Algebra with base ring Field of Fractions of Z_F
with coefficient ring Maximal Order of Equation Order with defining polynomial
x^2 - 5 over its ground order
> Discriminant(O);
Principal Ideal of Z_F
Generator:
    6/1*Z_F.1
> Discriminant(A);
Principal Prime Ideal of Z_F
Generator:
    [2, 0]
[ 1st place at infinity ]
> Level(G);
Principal Ideal of Z_F
Generator:
    3/1*Z_F.1
```

137.1.2 Quaternionic Functions

In this section, we provide some functions related quaternion algebras which underlie the other functions in this chapter.

Let A be a quaternion algebra over a totally real field F, represented in standard form by $\alpha, \beta \in A$ satisfying $\alpha^2 = a$, $\beta^2 = b$, and $\beta\alpha = -\alpha\beta$. Suppose that A has a unique split real place v; we then take this place to be the identity and consider F as a subfield of $\mathbf{R}$. Then at least one of $a, b > 0$, and without loss of generality we may assume that $a > 0$.

We then define the embedding $\iota : A \to M_2(\mathbf{R})$, given by

$$\alpha \mapsto \begin{pmatrix} \sqrt{a} & 0 \\ 0 & -\sqrt{a} \end{pmatrix}, \quad \beta \mapsto \begin{pmatrix} 0 & \sqrt{|b|} \\ \mathrm{sgn}(b)\sqrt{|b|} & 0 \end{pmatrix}.$$

This particular choice of embedding is governed by the following. The reduced norm $\mathrm{nrd} : A \to \mathbf{R}$ given by

$$\gamma = x + y\alpha + z\beta + w\alpha\beta \mapsto \mathrm{nrd}(\gamma) = x^2 - ay^2 - bz^2 + abw^2$$

is not a Euclidean norm, so it is natural to instead take

$$\gamma \mapsto \mathrm{nrd}'(\gamma) = x^2 + |a|y^2 + |b|z^2 + |ab|w^2,$$

which corresponds to

$$\mathrm{nrd}'(\gamma) = x^2 + y^2 + z^2 + w^2 \text{ if } \iota(\gamma) = \begin{pmatrix} x & y \\ z & w \end{pmatrix}.$$

Combining this new norm, defined for the identity real place, with the ones corresponding to the nonidentity real places σ, we may then define the definite norm

$$N(\gamma) = \mathrm{nrd}'(\gamma) + \sum_{\sigma \neq \mathrm{id}} \sigma(\mathrm{nrd}(\gamma)).$$

This norm (and the corresponding Gram matrix) on A are used in the application to fundamental domains (see the next section).

`QuaternionOrder(G)`

`BaseRing(G)`

The order in some quaternion algebra that was used to define the Fuchsian group G.

`QuaternionAlgebra(G)`

The quaternion algebra used to define the Fuchsian group G.

`SplitRealPlace(A)`

Returns the unique real place at which A is split, if it exists.

`FuchsianMatrixRepresentation(A)`

Returns the map $A \to M_2(\mathbf{R})$ when A has a unique split place, as defined above.

`DefiniteNorm(gamma)`

Returns the definite norm of $\gamma \in A$ for A a quaternion algebra with a unique split real place, as defined above.

`DefiniteGramMatrix(B)`

Returns the definite Gram matrix for the basis B of a quaternion algebra A with a unique split real place, as defined above.

Example H137E2

We begin by defining the quaternion algebra over $F = \mathbf{Q}(\sqrt{5})$ which is ramified at the prime ideal 2 and the second real place ∞_2, as indexed by Magma.

```
> F<w> := QuadraticField(5);
> Z_F := MaximalOrder(F);
> Foo := InfinitePlaces(F);
> A<alpha,beta> := QuaternionAlgebra(ideal<Z_F | 2>, [Foo[2]]);
> a, b := StandardForm(A);
> F!a, F!b;
-1 w + 2
```

The algorithm is random; entering the preceding input again will produce a different choice of a and b. Next we perform an independent check that a and b define the same algebra.

```
> IsIsomorphic(A, QuaternionAlgebra< F | a, b >);
true
```

We see that A is isomorphic to the algebra represented in standard form by

$$\alpha^2 = a,\ \beta^2 = b,\ \beta\alpha = -\alpha\beta$$

where $a = -1$, $b = 2 + \sqrt{5}$. (Note that the algorithm used to construct A is probabilistic, so a and b are not canonically given.)
We now demonstrate the real embedding corresponding to the split real place ∞_1.

```
> v := SplitRealPlace(A);
> v;
1st place at infinity
> iota := FuchsianMatrixRepresentation(A);
> iota(alpha);
[0.00000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000 1.000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000]
[-1.0000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000 0.000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000]
> iota(beta);
[2.05817102727149225032198104758045042123873009967781948628154235264047962315837
    8925542297104146968394 0.000000000000000000000000000000000000000000000
    00000000000000000000000000000000000000000000000000]
[0.00000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000000 -2.058171027271492250321981047580450421238730099677
    819486281542352640479623158378925542297104146968394]
> Sqrt(Evaluate(b, v));
2.05817102727149225032198104758045042123873009967781948628154235264047962315837
8925542297104146968394
> DefiniteNorm(alpha);
3.00000000000000000000000000000000000000000000000000000000000000000000000
```

```
00000000000000000000
```

We see indeed that

$$\alpha \mapsto \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \quad \beta \mapsto \begin{pmatrix} \sqrt{2+\sqrt{5}} & 0 \\ 0 & -\sqrt{2+\sqrt{5}} \end{pmatrix}$$

and

$$N(\alpha) = \mathrm{nrd}'(\alpha) + \sigma(\mathrm{nrd}(\alpha)) = (0^2 + 1^2 + (-1)^2 + 0^2) + 1 = 3.$$

`MultiplicativeOrder(gamma)`

`PlusMinus` BoolElt *Default :* `true`

Computes the order of the element γ of a quaternion algebra; either a finite number or 0 if the element has infinite order. If `PlusMinus` eq `true`, then compute the order in the group of units modulo ± 1.

`Quaternion(g)`

For g an element of an arithmetic Fuchsian group, return the underlying element of the quaternionic order.

137.1.3 Basic Invariants

In this section, we compute the basic invariants associated to an arithmetic Fuchsian group Γ.

The first is the hyperbolic volume $\mathrm{vol}(X)$ of the orbit space $X = \Gamma\backslash\mathbf{H}$, which can be computed via a formula of Shimizu:

$$\mathrm{vol}(X) = \left(\frac{1}{2}\right)^{d-2} \zeta_F(-1) \prod_{\mathfrak{p}|\mathrm{disc}A} (N(\mathfrak{p}) - 1)$$

where $d = [F : \mathbf{Q}]$. This volume is normalized so that an ideal triangle (i.e. a triangle with all vertices on the boundary of $\mathbf{H}$) has volume $1/2$. With this normalization, $\mathrm{vol}(X)$ takes rational values.

The next invariant is the signature of Γ. A point $z \in \mathbf{H}$ is an *elliptic point* of order $k \geq 2$ if the stabilizer of z under Γ, which is a finite cyclic group, has order k. There are only finitely many Γ-conjugacy classes of elliptic points, also known as *elliptic cycles*. If Γ has t elliptic cycles of order $m_1, \ldots, m_t$, and X has genus g, then we say that Γ has *signature* $(g; m_1, \ldots, m_t)$.

We compute the number of elliptic cycles of order k by a formula (too complicated to state here) which requires the computation of the relative class group of a CM extension of number fields. The genus g can then be calculated by the Riemann-Hurwitz formula:

$$\mathrm{vol}(X) = 2g - 2 + \sum_{i=1}^{t} \left(1 - \frac{1}{m_i}\right).$$

`ArithmeticVolume(G)`

Returns the hyperbolic volume of the quotient of the upper half-plane by G for an arithmetic Fuchsian group G. The volume is normalized arithmetic volume, so the "usual" volume is divided by 2π; this gives an ideal triangle volume $1/2$.

`EllipticInvariants(G)`

Returns the number of elliptic cycles of the arithmetic Fuchsian group G as a sequence of elliptic orders and their multiplicities.

`Signature(G)`

Returns the signature of the arithmetic Fuchsian group G.

137.1.4 Group Structure

We now compute a finite presentation for an arithmetic Fuchsian group Γ.

If Γ has signature $(g; m_1, \ldots, m_t)$, then there exists a standard presentation for Γ, generated (freely) by $\alpha_1, \beta_1, \ldots, \alpha_g, \beta_g, \gamma_1, \ldots, \gamma_t$ with the relations

$$\gamma_1^{m_1} = \ldots = \gamma_t^{m_t} = [\alpha_1, \beta_1] \cdots [\alpha_g, \beta_g] \gamma_1 \cdots \gamma_t = 1$$

where $[\alpha_i, \beta_i] = \alpha_i \beta_i \alpha_i^{-1} \beta_i^{-1}$ is the commutator. In the special case where $g = 0$, there exists an algorithm to reconstruct the standard presentation; if $g > 0$, we only obtain a finite presentation, and it is possible that the construction of a standard presentation may be algorithmically impractical.

Our algorithm uses a fundamental domain F for Γ (see section 137.3). In brief, we compute the set of elements which pair the sides of F; these elements are provably generators for the group Γ, and the relations between them are given as the set of minimal loops in a graph which records the side pairings.

`Group(G)`

Returns a presentation U for the arithmetic Fuchsian group G, a map $U \to G$, and a map $U \to \mathcal{O}$ where $\mathcal{O}$ is the quaternion order corresponding to G.

Example H137E3

In this example, we compute the basic invariants of the arithmetic Fuchsian group associated to a maximal order in the quaternion algebra of discriminant 6 over $\mathbf{Q}$.

```
> A := QuaternionAlgebra(6);
> O := MaximalOrder(A);
> G := FuchsianGroup(O);
> ArithmeticVolume(G);
1/3
> EllipticInvariants(G);
[ <2, 2>, <3, 2> ]
> Genus(G);
0
```

```
> Signature(G);
<0, [2, 2, 3, 3]>
```

We verify the Riemann-Hurwitz formula $1/3 = 2 \cdot 0 - 2 + 2(1 - \frac{1}{2}) + 2(1 - \frac{1}{3})$.
We can also compute a finite presentation of this group.

```
> U, m := Group(G);
> U;
Finitely presented group U on 3 generators
Relations
    U.1^2 = Id(U)
    U.2^3 = Id(U)
    U.3^3 = Id(U)
    (U.2 * U.1 * U.3)^2 = Id(U)
> [Matrix(m(U.i)) : i in [1..2]];
[
    [-0.44721359549995793928183473374634344441856698198020262470211905349191088
        22850343946302640398390408845 -1.09544511501033226913939565601820306361751136512834480425665136619270191
        17511365128344804256651366192701912074123047253565164680093741]
    [1.09544511501033226913939565601820306361751136512834480425665136619270191
        20741230472535651646800937 4 0.44721359549995793928183473374634344441856
        69819802026247021190534919108822850343946302640398390408845],
    [0.17082039324993690892275210061973565995395874811504763667302759126238306
        32123512881384895516677240 06 0.41842280112390929127226731105032485970030
        73307330865863518420141891341904873780366272041685003642481]
    [-2.86791254390708738946955138575621625166625481138975671478453458144009456
        79446930631670636016050044 83 -1.17082039324993690892275210061973565995395
        95874811504763667302759126238306632123512881384895516677240 1]
]
> [A!Quaternion(m(U.i)) : i in [1..2]];
[ -1/5*j + 1/5*k, -1/2 - 1/2*i + 3/10*j - 3/10*k ]
```

Next we compute a more complicated example, exhibiting an elliptic cycle of order 11. The group Γ corresponds to $F = \mathbf{Q}(\zeta_{11})^+$ and A the quaternion algebra only at real places. (See also the section on triangle groups.)

```
> K<z> := CyclotomicField(11);
> F := sub<K | z+1/z >;
> Foo := InfinitePlaces(F);
> Z_F := MaximalOrder(F);
> A := QuaternionAlgebra(ideal<Z_F | 1>, Foo[1..4]);
> G := FuchsianGroup(A);
> Signature(G);
<0, [ 2, 3, 11 ]>
> U, m := Group(G);
> U;
Finitely presented group U on 2 generators
Relations
    U.1^2 = Id(U)
    U.2^3 = Id(U)
```

```
    (U.1 * U.2^-1)^11 = Id(U)
```

As a final example, we show that the finite presentation of G may indeed be quite complicated. Here, we examine the (torsion-free) group associated to the quaternion algebra over $\mathbf{Q}$ with discriminant 35.

```
> G := FuchsianGroup(QuaternionOrder(35));
> Signature(G);
<3, []>;
> time U := Group(G);
Time: 78.870
> U;
Finitely presented group U on 6 generators
Relations
    U.5 * U.4^-1 * U.5^-1 * U.1 * U.3 * U.2^-1 * U.3^-1 * U.6 * U.4 *
    U.2 * U.1^-1 * U.6^-1 = Id(U)
```

137.2 Unit Disc

Let D denote the Poincaré unit disc,

$$D = \{z \in \mathbf{C} : |z| < 1\}$$

equipped with the hyperbolic metric

$$d(z, w) = \log\left(\frac{|1 - z\bar{w}| + |z - w|}{|1 - z\bar{w}| - |z - w|}\right).$$

We mimic the existing functionality for working with the upper half-plane (see Chapter 136) to compute in a similar way with the geometry of the unit disc, including computation of angles, intersections, areas, and so on.

137.2.1 Creation

`UnitDisc()`

`Center`	RngElt	*Default :* (9/10)i
`Precision`	RngElt	*Default :* 0

The hyperbolic unit disc, mapped conformally to the upper half-plane by mapping 0 in D to `Center`, with given `Precision`.

137.2.2 Basic Operations

`D ! x`

Coerces x into D, if possible.

`x eq y`

Returns `true` if and only if $x = y$.

`a * x`

Returns $a \cdot x$.

`x + y`

Returns $x + y$.

`x - y`

Returns $x - y$.

`x / a`

Returns $(1/a) \cdot x$.

137.2.3 Access Operations

`IsExact(z)`

Returns `true` (and the exact value of z) if and only if z is exact.

`ExactValue(z)`

Returns the exact value of z; if it does not exist, returns an error.

`ComplexValue(z)`

`Precision` RngIntElt *Default :* 0

Returns the complex value z, with given precision.

`Im(z)`
`Imaginary(z)`

`Precision` RngIntElt *Default :* 0

Returns the imaginary part of z, with given precision.

`Re(z)`
`Real(z)`

`Precision` RngIntElt *Default :* 0

Returns the real part of z, with given precision.

`Argument(z)`

`Precision` RngIntElt *Default :* 0

Returns the argument of z, with given precision.

`Abs(z)`
`AbsoluteValue(z)`

`Precision` RngIntElt *Default :* 0

Returns the absolute value of z, with given precision.

Example H137E4

In this example, we exhibit basic functions for the unit disc.

```
> D := UnitDisc();
> D;
Hyperbolic unit disc
> CC<I> := ComplexField();
> w := D ! ((1+I)/2);
> w;
0.50000000000000000000000000000 + 0.50000000000000000000000000000*I
> Re(w), Abs(w);
0.50000000000000000000000000000 0.70710678118654752440084436210 5
```

Note that since D does not form a ring, when binary operations are performed, we return elements according to the base ring in which they belong; if desired, they can be coerced back into D.

```
> z := D ! 0;
> z;
0
> IsExact(z);
true 0
> ComplexValue(z : Precision := 100);
0.00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
> 2*z;
0
> Type($1);
FldRatElt
> D!(z+w) eq w;
true
```

137.2.4 Distance and Angles

`Distance(z,w)`

`Precision` RngIntElt *Default :* 0

Returns the hyperbolic distance between z and w.

`Geodesic(z,w)`

`Precision` RngIntElt *Default :* 0

Returns the center and radius of the geodesic containing z and w, with given precision.

`TangentAngle(x,y)`

`Precision` RngIntElt *Default :* 0

Returns the angle of the tangent at x of the geodescic from x to y, with given precision.

`Angle(e1,e2)`

`Precision` RngIntElt *Default :* 0

Given two sequences $e_1 = [z_1, z_2]$ and $e_2 = [z_1, z_3]$, returns the angle between the geodesics at z_1.

`ArithmeticVolume(P)`

`Precision` RngIntElt *Default :* 0

The volume of the convex region specified the sequence of elements of the unit disc. The elements must be specified in counterclockwise order by their arguments. The volume is normalized "arithmetic" volume, so the "usual" volume is divided by 2π; this gives an ideal triangle volume $1/2$.

Example H137E5

In this example, we compute geodesics in the unit disc.

```
> D := UnitDisc();
> CC<I> := ComplexField();
> z0 := D!0;
> z1 := D!(1/2*I);
> z2 := D!(1/2);
> Distance(z0,z1);
1.0986122886681096913952452369 2
> Geodesic(z0,z1);
Infty 0.00000000000000000000000000000
> Geodesic(z1,z2);
1.25000000000000000000000000000 + 1.25000000000000000000000000000*I
1.4577379737113251177185382193 9
> TangentAngle(z0,z1);
```

```
1.5707963267948966192313216916 4
> TangentAngle(z1,z2);
-1.0303768265243124637877433270 3
> Angle([z1,z2],[z1,z0]);
2.6011731533192090830190650186 7
> ArithmeticVolume([D | 1/2+1/2*I, -1/2+1/2*I, -1/2-1/2*I, 1/2-1/2*I]);
0.5903344706017330967016043048 99
```

137.2.5 Structural Operations

`T * x`

Returns $T(x)$, using the identification of the unit disc with the upper half-plane.

`Center(D)`

Returns the element in the upper half-plane which maps to 0 in D.

`DiscToPlane(H,z)`

Maps z in a unit disc to **H**.

`PlaneToDisc(D,z)`

Maps z in an upper half-plane to D.

`Matrix(g,D)`

Returns the matrix representation of g acting on the unit disc D.

`FixedPoints(g,D)`

Returns the fixed points of g acting on D.

`IsometricCircle(g)`

`IsometricCircle(g,H)`

Returns the center and radius of the set of points in the upper half-plane H where g acts as a Euclidean isometry.

`IsometricCircle(g,D)`

Returns the center and radius of the set of points in the unit disc D where g acts as a Euclidean isometry.

`GeodesicsIntersection(x1,x2)`

Returns the intersection in the unit disc of the two geodesics x_1, x_2, where x and y are specified by their end points. If more than one intersection exists, returns a sequence.

`BoundaryIntersection(x)`

Computes the intersection of the geodesic x with the boundary of the unit disc.

Example H137E6

In this example, we illustrate the use of the structural operations on the unit disc. We begin by declaring a Fuchsian group over the totally real subfield $F = \mathbf{Q}(\zeta_9)^+$ of $\mathbf{Q}(\zeta_9)$.

```
> K<z> := CyclotomicField(9);
> F := sub<K | z+1/z >;
> b := F! (z+1/z);
> A<alpha,beta> := QuaternionAlgebra<F | -3, -b>;
> G := FuchsianGroup(A);
> O := BaseRing(G);
> A<alpha,beta> := Algebra(O);
```

Next, we compute the fixed point of the element $g = (1 + \alpha)/2$.

```
> g := G!((1+alpha)/2);
> H := UpperHalfPlane();
> FixedPoints(g, H);
[
    root(-1)
]
> Matrix(g);
[0.500000000000000000000000000000000000000000000000000000000000000000000000000000000000
    000000000000000000000 0.866025403784438646763723170752936183471402626905
    190314027903489725966508454400018540573093378624287 8]
[-0.866025403784438646763723170752936183471402626905190314027903489725966508454
    400018540573093378624287 8 0.500000000000000000000000000000000000000000000000
    000000000000000000000000000000000000000000000000000]
> g*H.1;
root(-1);
```

The fixed point is at $z = i$. Next, we compute the isometric circle $C(g)$ of g,

$$C(g) = \{z \in \mathbf{C} : |g(z)| = |z|\}.$$

(See also the next section on fundamental domains.)

```
> cH, rH := IsometricCircle(g);
> cH;
0.577350269189625764509148780501957455647601751270126876018602326483977672302933
3345693715395585749525 2
> rH;
1.154700538379251529018297561003914911295203502540253752037204652967955344605866
6691387430791171499050
```

The circle $C(g)$ has radius $r_H = 1.154\ldots$ and center $c_H = 0.577\ldots$. We verify that g acts by a Euclidean isometry at the point $c_H + r_H i \in C(g)$.

```
> CC<I> := ComplexField();
> Abs(Matrix(g)[2,1]*(cH+rH*I)+Matrix(g)[2,2]);
```

```
1.0000000000000000000000000000
```

We now repeat these same steps, with g acting on the unit disc D.

```
> D := UnitDisc(: Center := 9/10*H.1);
> Matrix(g, D);
[0.5000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000000000 + 0.8708366560276855281346327439238178790538869671
    9328752118954682448361215363361924889856628696809493679*$.1
    -0.0914137926216907460472818902464485833632141167921465689623861842053178 61
    4632935276105232033434805044 9*$.1]
[0.0914137926216907460472818902464485833632141167921465689623861842053178614632
    9352761052320334348050449*$.1 0.5000000000000000000000000000000000000000000000
    0000000000000000000000000000000000000000000000000000000000000000000000000000 -
    0.8708366560276855281346327439238178790538869671932875211895468244836121536
    3361924889856628696809493679*$.1]
> cD, rD := IsometricCircle(g, D);
> Abs(Matrix(g, D)[2,1]*PlaneToDisc(D, H!(cH+rH*I))+Matrix(g, D)[2,2]);
1.0000000000000000000000000000
```

137.3 Fundamental Domains

Let D denote the unit disc and Γ an arithmetic Fuchsian group. A *fundamental domain* in D for Γ is a closed, hyperbolically convex region $P \subset D$ such that the translates of P by Γ cover D and such that the interiors of all translates P are disjoint, i.e. $D = \bigcup_{\gamma\in\Gamma} \gamma P$ and $\mathrm{int}(P) \cap \mathrm{int}(\gamma P) = \emptyset$ for $\gamma \neq 1$, where $\mathrm{int}(P)$ denotes the interior of P.

Choosing a point $p \in D$ not fixed by any element of $\Gamma \setminus \{1\}$, we obtain a fundamental domain by letting

$$P = \{z \in D : d(z,p) \leq d(z,\gamma(p)) \text{ for all } \gamma \in \Gamma\}.$$

Typically, one takes $p = 0$.

One can similarly define a fundamental domain in the upper half-plane $\mathbf{H}$, and one can bijectively map fundamental domains in D to those in $\mathbf{H}$ via a choice of conformal map between them. The unit disc is a more natural setting for our algorithms.

For each $\gamma \in \Gamma \setminus \{1\}$, we define the *isometric circle* of γ to be the circle

$$C(\gamma) = \{z \in \mathbf{C} : |\gamma(z)| = |z|\}.$$

Then in fact P is the closure of the intersection of half-spaces given by

$$\bigcap_{\gamma\in\Gamma\setminus\{1\}} C(\gamma)^o$$

where $C(\gamma)^o$ denotes the exterior of $C(\gamma)$. Our algorithm recursively finds elements in Γ to decrease the hyperbolic volume of this intersection until the volume $\mathrm{vol}(X)$ is reached; it relies upon a "reduction theory" with respect to a set of generators of Γ.

`FundamentalDomain(G,D)`

Computes a fundamental domain in the unit disc D for the action of G.

`FundamentalDomain(G)`

Computes a fundamental domain in the upper half-plane for the action of G.

Example H137E7

We first compute a fundamental domain for a quaternion algebra defined over $\mathbf{Q}(\zeta_7)^+$ which turns out to be the $(2,3,7)$-triangle group.

```
> K<z> := CyclotomicField(7);
> F := sub<K | z+1/z >;
> b := F! (z+1/z);
> A<i,j,k> := QuaternionAlgebra<F | b, b>;
> O := MaximalOrder(A);
> G := FuchsianGroup(O);
> P := FundamentalDomain(G, UnitDisc());
> P;
[
    0.05634669176194547731955781246 39 -
      0.26528816249569116795706789925 7*$.1,
    0.05634669176194547731955781246 39 +
      0.26528816249569116795706789925 7*$.1,
    -0.08865048559477002646152542945 00 -
      4.57194956512909992886313419322E-100*$.1
]
> P := FundamentalDomain(G);
> P;
[
    0.49697042539518089622118039244 5 +
      (0.86776747823511624095153666569 6)*root(-1),
    -0.49697042539518089622118039244 5 +
      (0.86776747823511624095153666569 6)*root(-1),
    6.94379666203368024633240684073E-100 +
      (0.75342322794867759872523662413 0)*root(-1)
]
> ArithmeticVolume(P);
0.023809523809523809523809523809 2
> ArithmeticVolume(G);
1/42
> ($1)*1.0;
0.023809523809523809523809523809 5
```

We can visualize the domain P by using the postscript plotting tools of Chapter 136.

```
>  DisplayPolygons(P, "/tmp/quat237triang.ps" : Show := true);
[ -0.49697042539518089622118039244 5, 0.49697042539518089622118039244 5,
```

```
1.0500000000000000000000000000, 302.0000000000000000000000000000 ]
```

We repeat this with the quaternion algebra over $\mathbf{Q}$ of discriminant 10.

```
> G := FuchsianGroup(QuaternionOrder(10));
> P := FundamentalDomain(G);
>  DisplayPolygons(P, "/tmp/quat10.ps" : Show := true);
[ -1.3144841297278098202363616814O, 0.158113883008418966599944677206,
1.279124762273123946981795500S6, 235.0000000000000000000000000000 ]
> U, m := Group(G);
> U;
Finitely presented group U on 3 generators
Relations
    U.1^3 = Id(U)
    U.2^3 = Id(U)
    U.3^3 = Id(U)
    (U.1^-1 * U.2^-1 * U.3^-1)^3 = Id(U)
```

`ShimuraReduceUnit(delta, gammagens, G, D)`

`CreateWord`	BoolElt	*Default :* `false`
`NextSmallest`	BoolElt	*Default :* `false`
`z0`	SpcHydElt	*Default :* 0
`z1`	SpcHydElt	*Default :* 0

Reduce the unit delta moving z_0 to z_1 with respect to the generators in γ_{gens} by multiplying on the left or right by elements of γ_{gens} inside the arithmetic Fuchsian group G to minimize the distance to the origin in the unit disc D. Returns a sequence of triples containing reduced elements and the sequence on the left and right to obtain them. The argument `CreateWord` reduces δ even if δ is in γ_{gens}, disallowing the trivial word.

137.4 Triangle Groups

We now look at a certain class of arithmetic Fuchsian groups of particular interest. We assume that $\Gamma(1)$ is a *triangle group*, i.e. Γ has signature $(0; p, q, r)$ for $p, q, r \in \mathbf{Z}_{\geq} 2$. Hence there exists a presentation

$$\Gamma(1) \cong \langle \gamma_p, \gamma_q, \gamma_r \mid \gamma_p^p = \gamma_q^q = \gamma_r^r = \gamma_p\gamma_q\gamma_r = 1 \rangle.$$

Since $g = 0$, there exists a map j from $X(1)_{\mathbf{C}}$ to the projective line over $\mathbf{C}$ which is ramified only above the three *elliptic points*, the fixed points of $\gamma_p, \gamma_q, \gamma_r$, which we may take to be $0, 1, \infty$.

There are only finitely many such triples (p, q, r), and we have implemented a constructor for these triples which yields the group Γ as well as the arithmetic data including the maximal order $\mathcal{O}$ and the quaternion algebra A. We also compute the analytic map j for each of these groups, which includes methods for evaluating the ${}_2F_1$-hypergeometric series to extremely high precision at complex arguments.

137.4.1 Creation of Triangle Groups

We begin with the basic constructors.

`ArithmeticTriangleGroup(p,q,r)`

Returns the arithmetic triangle group of signature (p, q, r).

`AdmissableTriangleGroups()`

Returns the list of arithmetic triangle groups currently implemented.

`IsTriangleGroup(G)`

Returns `true` if and only if G was created as an arithmetic triangle group.

137.4.2 Fundamental Domain

`ReduceToTriangleVertices(G,z)`

Returns points in the G-orbit of z which are nearest to the vertices of the fundamental domain for G a triangle group.

137.4.3 CM Points

For triangle groups, we can compute CM points analytically. By work of Shimura, the curve $X(1)$ has an interpretation as a moduli space for certain abelian varieties and has a canonical model defined over $\mathbf{Q}$. Let K be a totally imaginary quadratic extension of F, and let $O_D \subset K$ be a quadratic order. Suppose that K splits A, i.e. $A \otimes_F K \cong M_2(K)$. Then there exists an embedding $\iota_K : K \hookrightarrow A$ such that $\iota_K(K) \cap \mathcal{O} = O_D$, given by an element $\mu \in \mathcal{O}$ such that $\mathbf{Z}_F[\mu]$ is isomorphic to O_D. The unique fixed point of $\iota_\infty(\mu)$ in $\mathbf{H}$ yields a *CM point* on $X(1)$ that is defined over an abelian extension H of F, and there is a reciprocity law which describes the action of $\mathrm{Gal}(H/K)$.

We exhibit algorithms for computing with these objects. Explicitly, given the data of a quadratic order O_D contained in a imaginary quadratic extension K of F, we compute the set of $\mathrm{Gal}(H/K)$-conjugates of a CM-point for O_D on $X(1)$. Our algorithm first computes representatives for the ring class group in terms of the Artin map. It then applies the Shimura reciprocity law for each of these representatives, which uses our architecture for principalization of ideals of quaternionic orders, to compute the conjugates. Given these points to high enough precision, we can recognize the CM point as a putative algebraic number.

`HypergeometricSeries2F1(A,B,C,z)`

Returns the value of the 2F1-hypergeometric function with parameters (A, B; C) at the complex number z.

Example H137E8

Although programmed for use by the hypergeometric reversion procedure, involved in the calculation of CM points, the `HypergeometricSeries2F1` function can also be called independently to evaluate the series at any complex number at any precision.

```
> HypergeometricSeries2F1(1/2, 1/2, 1, 1);
0.31830988618379067153776752674
> CC<I> := ComplexField(100);
> HypergeometricSeries2F1(1/2, 1/3, 1/4, 1+I);
0.18914889986718938009890989889270323251621296927695426144398500333164293730405
5659151692186899720481 9 + 1.3516500011026082918038421690042816399520000054582827705534621378149655804166701124643888726219267 52*I
```

`ShimuraConjugates(mu)`

Returns the set of conjugates of the quaternion order element μ under the Shimura reciprocity law.

`jParameter(G,z)`

`Bound`	RngIntElt	*Default :* 200
`Precision`	RngIntElt	*Default :* 100

Returns the value of the map $j : X(G) \to \mathbf{P}^1$ at z, for G an arithmetic triangle group.

Example H137E9

We begin by constructing the $(2, 3, 9)$-triangle group.

```
> G := ArithmeticTriangleGroup(2,3,9);
> O := BaseRing(G);
> Z_F := BaseRing(O);
```

Next, we compute the CM point corresponding to the imaginary quadratic extension K/F of discriminant -7. First, we define the extension.

```
> Z_K := ext<Z_F | Polynomial([2,-1,1])>;
> Discriminant(Z_K);
Principal Ideal of Z_F
Generator:
    [-7, 0, 0]
> IsMaximal(Z_K);
true
> ClassNumber(AbsoluteOrder(Z_K));
1
```

Since $\mathbf{Z}_K$ has class number 1, the corresponding CM point will be a rational number. We now embed $\mathbf{Z}_K$ into $\mathcal{O}$.

```
> mu := Embed(Z_K, O);
```

```
> MinimalPolynomial(mu);
$.1^2 - $.1 + 2/1*Z_F.1
```

Finally, we compute the fixed point z of μ in the upper half-plane, and compute the value of the uniformizing parameter $j(z)$.

```
> z := FixedPoints(G!mu, UpperHalfPlane())[1];
> j := jParameter(G, z);
> j;
-9594.703124999999999999999999999999999999999999999999999999999999999999999999999
998082712686344925455 41 - 8.607499388211909042811249895070271789213361939086074
091584429508486426059712018492327398804638684615E-77*I
> BestApproximation(Re(j), 10^50);
-614061/64
> Factorization(Numerator($1)), Factorization(Denominator($1));
[ <3, 5>, <7, 1>, <19, 2> ]
[ <2, 6> ]
```

We recognize the value $j(z) = \frac{3^5 7^1 19^2}{2^6}$ as a smooth rational number.

`CMPoints(G,mu)`

Returns the minimal polynomial of the Galois conjugates of the CM points corresponding to μ and their complex values for G a triangle group. The element μ must be Galois-stable, i.e. $F(\mu)$ must be Galois over F, where F is the defining base field of G. The polynomial returned is the "best guess" rational approximation to the polynomial with the CM points as roots.

Example H137E10

We now compute the minimal polynomial for CM points which generate a nontrivial extension: we again take the $(2, 3, 9)$-triangle group and now consider K of discriminant -11.

```
> G := ArithmeticTriangleGroup(2,3,9);
> O := BaseRing(G);
> Z_F := BaseRing(O);
> Z_K := ext<Z_F | Polynomial([3,-1,1])>;
> Discriminant(Z_K);
Principal Ideal of Z_F
Generator:
    [-11, 0, 0]
> IsMaximal(Z_K);
true
> ClassNumber(AbsoluteOrder(Z_K));
3
> mu := Embed(Z_K, O);
> time f, js := CMPoints(G, mu);
Time: 18.120
> f;
```

```
-127332762814175510528*x^3 + 297798194745682427904*x^2 - 545543748833233386651*x
    + 443957770932571793051
```

The number field $H = K(\alpha)$, where α is a root of f, is the Hilbert class field of K, which we have computed analytically!

```
> D := Discriminant(f);
> Denominator(D);
1
> Factorization(Numerator(D));
[ <2, 32>, <3, 18>, <11, 1>, <17, 6>, <19, 12>, <73, 2>, <107, 6>, <109, 2>,
<197, 2>, <271, 2>, <431, 2>, <701, 2> ]
> K := AbsoluteField(NumberField(Z_K));
> H := ext<K | f>;
> Factorization(Discriminant(MaximalOrder(H)));
[]
> IsAbelian(GaloisGroup(H));
true
```

137.5 Bibliography

[**AB04**] Montserrat Alsina and Pilar Bayer. *Quaternion orders, quadratic forms, and Shimura curves*, volume 22 of *CRM Monograph Series*. American Mathematical Society, Providence, RI, 2004.

[**Bea77**] A. F. Beardon. The geometry of discrete groups. In *Discrete groups and automorphic functions (Proc. Conf., Cambridge, 1975)*, pages 47–72. Academic Press, London, 1977.

[**Elk98**] Noam D. Elkies. Shimura curve computations. In *Algorithmic number theory (Portland, OR, 1998)*, volume 1423 of *LNCS*, pages 1–47. Springer, Berlin, 1998.

[**Kat92**] Svetlana Katok. *Fuchsian groups*. Chicago Lectures in Mathematics. University of Chicago Press, Chicago, IL, 1992.

[**Vig80**] M.-F. Vignéras. *Arithmétique des Algèbres de Quaternions*, volume 800 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1980.

[**Voi05**] John Voight. *Quadratic forms and quaternion algebras: Algorithms and arithmetic*. Dissertation, University of California, Berkeley, 2005.

[**Voi06**] John Voight. Computing CM points on Shimura curves arising from cocompact arithmetic triangle groups. In *Algorithmic number theory*, volume 4076 of *LNCS*, pages 406–420. Springer, Berlin, 2006.

[**Voi09**] J. Voight. Computing fundamental domains for cofinite Fuchsian groups. *J. Théor. Nombres Bordeaux*, 21(2):469–491, 2009.

138 MODULAR FORMS

Chapter 138
MODULAR FORMS

138.1 Introduction

138.1.1 Modular Forms

This theoretically-oriented section serves as a guide to the rest of the chapter. We recall the definition of modular forms, then briefly discuss q-expansions, Hecke operators, eigenforms, congruences, and modular symbols.

Fix positive integers N and k, let $\mathbf{H}$ denote the complex upper half plane. Denote by $M_k(\Gamma_1(N))$ the space of modular forms on $\Gamma_1(N)$ of weight k. This is the complex vector space of holomorphic functions $f : \mathbf{H} \to \mathbf{C}$ such that

$$f\left(\frac{az+b}{cz+d}\right) = (cz+d)^k f(z) \qquad \text{for all} \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma_1(N),$$

and $f(z)$ is holomorphic at each "cusp" in $\mathbf{P}^1(\mathbf{Q}) = \mathbf{Q} \cup \{\infty\}$ (see, e.g., [DI95] for a more precise definition.)

A *Dirichlet character* is a homomorphism $\varepsilon : (\mathbf{Z}/N\mathbf{Z})^* \to \mathbf{C}^*$ of abelian groups. Dirichlet characters are of interest because they decompose $M_k(\Gamma_1(N))$ into more manageable chunks. If V is any complex vector space equipped with an action $\rho : (\mathbf{Z}/N\mathbf{Z})^* \to \mathrm{Aut}(V)$ and ε is a Dirichlet character, we set

$$V(\varepsilon) = \{x \in V : \rho(a)x = \varepsilon(a)x \quad \text{all} \quad a \in (\mathbf{Z}/N\mathbf{Z})^*\}.$$

The space $M_k(\Gamma_1(N))$ is equipped with an action of $(\mathbf{Z}/N\mathbf{Z})^*$ by the diamond-bracket operators $\langle d \rangle$, which are defined as follows. Given $\overline{d} \in (\mathbf{Z}/N\mathbf{Z})^*$, choose a matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma_0(N)$ such that $d \bmod N = \overline{d}$. Then

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

We call $M_k(\Gamma_1(N))(\varepsilon)$ the space of modular forms of weight k, level N, and character ε. This is the complex vector space of holomorphic functions $f : \mathbf{H} \to \mathbf{C}$ such that

$$f\left(\frac{az+b}{cz+d}\right) = \varepsilon(a)(cz+d)^k f(z) \qquad \text{for all} \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \Gamma_0(N),$$

and which is holomorphic at the cusps. We let $M_k([\varepsilon])$ denote the direct sum of the spaces $M_k(\Gamma_1(N))(\varepsilon)$ *as* ε *varies over the* $\mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})$*-conjugates* of ε. It is unnecessary to specify the level because it is built into ε.

To summarize, for any integer k and positive integer N, there is a finite-dimensional $\mathbf{C}$-vector space $M_k(\Gamma_1(N))$. Moreover,

$$M_k(\Gamma_1(N)) = \bigoplus_{\text{all } \varepsilon} M_k(\Gamma_1(N))(\varepsilon) \quad = \quad \bigoplus_{\text{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})\text{-class reps. } \varepsilon} M_k([\varepsilon]).$$

In Section 145.2, we describe how to create the spaces $M_k(\Gamma_1(N))$ and $M_k([\varepsilon])$ in MAGMA, for any $k \geq 1$, $N \geq 1$, and character ε.

Let f be a modular form, and observe that since $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \in \Gamma_1(N)$, we have $f(z) = f(z+1)$. If we set $q = \exp(2\pi i z)$, there is a *q-expansion* representation for f:

$$f = a_0 + a_1 q + a_2 q^2 + a_3 q^3 + a_4 q^4 + \cdots.$$

The a_n are called the *Fourier coefficients* of f. MAGMA contains an algorithm for computing a basis of q-expansions for any space of modular forms of weight $k \geq 2$ (see Section 141.3).

Fix a positive integer N and let M be a sum of spaces $M_k(N, \varepsilon)$. Let $q{-}\exp : M \to \mathbf{C}[[q]]$ denote the map that associates to a modular form f its q-expansion. One can prove that there is a basis $f_1, \ldots, f_d$ of M that maps to a basis for the free $\mathbf{Z}$-module $q{-}\exp(M) \cap \mathbf{Z}[[q]]$. See Section 138.4 for how to compute such a basis in MAGMA. Let $M_{\mathbf{Z}}$ be the $\mathbf{Z}$-module spanned by $f_1, \ldots, f_d$. For any ring R, we define the *space of modular forms over* R to be $M_R = M_{\mathbf{Z}} \otimes_{\mathbf{Z}} R$. Thus M_R is a free R-module of rank d with basis the images of $f_1, \ldots, f_d$ in $M_{\mathbf{Z}} \otimes_{\mathbf{Z}} R$. The computation of M_R is discussed in Section 138.2.2.

Any space M of modular forms is equipped with an action of a commutative ring $\mathbf{T} = \mathbf{Z}[\ldots T_n \ldots]$ of Hecke operators. The computation of Hecke operators T_n and their characteristic polynomials is described in Section 142.14.

An *eigenform* is a simultaneous eigenvector for every element of the Hecke algebra $\mathbf{T}$. A *newform* is an eigenform that doesn't come from a space of lower level and is normalized so that the coefficient of q is 1. Section 138.13 describes how to find newforms. Computation of the mod p reductions and p-adic and complex embeddings of a newform is described in Section 138.14.

Computation of congruences is discussed in Section 138.15.

Modular symbols are closely related to modular forms. See Section 138.19 for the connection between the two.

138.1.2 About the Package

The modular forms package is in many ways an interface to the modular symbols machinery (Section 138.19). It also contains additional functionality (such as Eisenstein series), and features that are implemented independently (such as Hecke operators). In some situations however, it is better to work with modular symbols directly. In particular, spaces of modular forms are required to be Galois-stable over the rationals, while spaces of modular symbols are not. Moreover some features, such as Hecke operators of certain spaces, are

unavailable for a given space of modular forms but can be obtained using the corresponding modular symbols.

In MAGMA version 2.14 modular forms of weight one, and of half-integral weight, were added. Most of the existing functions now also work for these weights, however some are not implemented (for instance `Newforms`). Further functionality for half-integral weight will be added in future releases (including Hecke operators).

138.1.3 Categories

In MAGMA, spaces of modular forms belong to the category `ModFrm`, and the elements of spaces of modular forms belong to `ModFrmElt`.

138.1.4 Verbose Output

To set the verbosity level use the command `SetVerbose("ModularForms",n)`, where `n` is 0 (silent), 1 (verbose), or 2 (very verbose). The default verbose level is 0.

Example H138E1

In this example, we illustrate categories and verbosity for modular forms.

```
> M := ModularForms(11,2); M;
Space of modular forms on Gamma_0(11) of weight 2 and dimension 2 over
Integer Ring.
> Type(M);
ModFrm
> B := Basis(M); B;
[
    1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + O(q^8),
    q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
]
> Type(B[1]);
ModFrmElt
```

Using `SetVerbose`, we get some information about what is happening during computations.

```
> SetVerbose("ModularForms",2);
> M := ModularForms(30,4);
> EisensteinSubspace(M);
ModularForms: Computing eisenstein subspace.
ModularForms: Computing dimension.
Space of modular forms on Gamma_0(30) of weight 4 and dimension 8 over
Integer Ring.
> SetVerbose("ModularForms",0);   // turn off verbose mode
```

138.1.5 An Illustrative Overview

In this section, we give a longer example that serves as an overview of the modular forms package. It illustrates computations of modular forms of level 1, and illustrates the exceptional case of Serre's conjecture with a level 13 example.

Example H138E2

First, we compute the two-dimensional space of modular forms of weight 12 and level 1 over **Z**.

```
> M := ModularForms(Gamma0(1),12); M;
Space of modular forms on Gamma_0(1) of weight 12 and dimension 2 over
Integer Ring.
```

The default output precision is 8:

```
> Basis(M);
[
    1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + 4629381120*q^5 +
    34417656000*q^6 + 187489935360*q^7 + O(q^8),
    q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7
    + O(q^8)
]
> [PowerSeries(f,10) : f in Basis(M)];
[
    1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + 4629381120*q^5 +
        34417656000*q^6 + 187489935360*q^7 + 814879774800*q^8 +
        2975551488000*q^9 + O(q^10),
    q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7
        + 84480*q^8 - 113643*q^9 + O(q^10)
]
> f := Basis(M)[1];
> Coefficient(f,2);
196560
```

The `Newforms` command returns a list of the Galois-orbits of newforms.

```
> NumberOfNewformClasses(M);
2
> f := Newform(M,1); f;
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 +
O(q^8)
```

We can call the "q" in the q-expansion anything we want and also compute forms to higher precision.

```
> Mf<w> := Parent(f); Mf;
Space of modular forms on Gamma_0(1) of weight 12 and dimension 1 over
Rational Field.
> f + O(w^12);
w - 24*w^2 + 252*w^3 - 1472*w^4 + 4830*w^5 - 6048*w^6 - 16744*w^7 +
```

```
84480*w^8 - 113643*w^9 - 115920*w^10 + 534612*w^11 + O(w^12)
```

Typing `f + O(w^12)` is equivalent to typing `PowerSeries(f,12)`. The second newform orbit contains an Eisenstein series.

```
> E := Newform(M,2); PowerSeries(E,2);
691/65520 + q + O(q^2)
```

Next we compute the two conjugate newforms in $S_2(\Gamma_1(13))$:

```
> M := ModularForms(Gamma1(13),2);
> S := CuspidalSubspace(M); S;
Space of modular forms on Gamma_1(13) of weight 2 and dimension 2 over
Integer Ring.
> NumberOfNewformClasses(S);
1
> f := Newform(S, 1); f;
q + (-a - 1)*q^2 + (2*a - 2)*q^3 + a*q^4 + (-2*a + 1)*q^5 + (-2*a +
4)*q^6 + O(q^8)
```

Here a is a root of the polynomial $x^2 - x + 1$.

```
> Parent(f);
Space of modular forms on Gamma_1(13) of weight 2 and dimension 2 over
Number Field with defining polynomial x^2 - x + 1 over the Rational
Field.
```

The Galois-conjugacy class of f has two newforms in it.

```
> Degree(f);
2
> g := Newform(S, 1, 2); g;
q + (-b - 1)*q^2 + (2*b - 2)*q^3 + b*q^4 + (-2*b + 1)*q^5 + (-2*b +
4)*q^6 + O(q^8)
> BaseRing(Parent(g));
Number Field with defining polynomial x^2 - x + 1 over the Rational
Field
```

The parents of f and g are isomorphic (but distinct) abstract extensions of $\mathbf{Q}$ both isomorphic to $\mathbf{Q}(\sqrt{-3})$.

```
> Parent(f) eq Parent(g);
false
```

We can also list all of the newforms at once, gathered into Galois orbits, using the `Newforms` command:

```
> N := Newforms(M);
> #N;
8
> N[3];
[*
1/13*(-7*zeta_6 - 11) + q + (2*zeta_6 + 1)*q^2 + (-3*zeta_6 + 1)*q^3 +
```

```
(6*zeta_6 - 3)*q^4 - 4*q^5 + (-7*zeta_6 + 7)*q^6 + (-7*zeta_6 + 8)*q^7
+ O(q^8),
1/13*(7*zeta_6 - 18) + q + (-2*zeta_6 + 3)*q^2 + (3*zeta_6 - 2)*q^3 +
(-6*zeta_6 + 3)*q^4 - 4*q^5 + 7*zeta_6*q^6 + (7*zeta_6 + 1)*q^7 +
O(q^8)
*]
```

The "Nebentypus" or character of f has order 6.

```
> e := DirichletCharacter(f); Parent(e);
Group of Dirichlet characters of modulus 13 over Cyclotomic Field of
order 6 and degree 2
> Order(e);
6
```

This shows that there are no cuspidal newforms with Dirichlet character of order 2. As we see below, the mod-3 reduction of f has character $\overline{\varepsilon}$ that takes values in $\mathbf{F}_3$ and has order 2. The minimal lift of $\overline{\varepsilon}$ to characteristic 0 has order 2, while there are no newforms of level 13 with character of order 2. (This example illustrates the "exceptional case of Serre's conjecture".)

```
> f3 := Reductions(f,3); f3;
[* [*
q + 2*q^3 + 2*q^4 + O(q^8)
*] *]
> M3<q> := Parent(f3[1][1]);
> f3[1][1]+O(q^15);
q + 2*q^3 + 2*q^4 + q^9 + q^12 + 2*q^13 + O(q^15)
```

The modular forms package can also be used to quickly compute dimensions of spaces of modular forms.

```
> M := ModularForms(Gamma0(1),2048);
> Dimension(M);
171
> M := ModularForms(Gamma1(389),2);
> Dimension(M);
6499
> Dimension(CuspidalSubspace(M));
6112
```

Don't try to compute a basis of q-expansions for M!

```
> M := ModularForms(Gamma0(123456789),6);
> Dimension(CuspidalSubspace(M));
68624152
> Dimension(EisensteinSubspace(M));
16
```

138.2 Creation Functions

138.2.1 Ambient Spaces

The functions in this section are used to create spaces of modular forms. Spaces of half-integral weight can also be created (see the next section).

For information on Dirichlet characters, see Section 19.8.

`ModularForms(N)`

The space $M_2(\Gamma_0(N), \mathbf{Z})$ of modular forms on $\Gamma_0(N)$ of weight 2. See the documentation for `ModularForms(N,k)` below, with $k = 2$.

`ModularForms(N, k)`

The space $M_k(\Gamma_0(N), \mathbf{Z})$ of weight k modular forms on $\Gamma_0(N)$ over $\mathbf{Z}$.

`ModularForms(eps, k)`

Given a Dirichlet character *eps* and an integer k, this returns a space of modular forms over the integers, of weight k, which under base extension becomes equal to the direct sum of the spaces $M_k(\Gamma_1(N), eps1)$ of weight k and nebentypus character *eps1*, where *eps1* runs over all Galois conjugates *eps*.

`ModularForms(chars, k)`

The space of modular forms of weight k over the integers, formed as the direct sum of spaces `ModularForms(eps,k)`, summing over all *eps* in the given sequence *chars* of Dirichlet characters.

`ModularForms(G, k)`

`ModularForms(G)`

The space $M_k(G, \mathbf{Z})$, where G is a congruence subgroup. The groups $\Gamma_0(N)$ and $\Gamma_1(N)$ are currently supported, and can be created using the commands `Gamma0(N)` and `Gamma1(N)`, respectively. When not specified, $k = 2$.

`CuspForms(x)`

`CuspForms(x, y)`

These commands are a shortcut, and return the `CuspidalSubspace` of the corresponding full space of modular forms.

Example H138E3

In this example, we illustrate each of the above constructors in turn. First we create $M_2(\Gamma_0(65))$.

```
> M := ModularForms(65); M;
Space of modular forms on Gamma_0(65) of weight 2 and dimension 8 over
Integer Ring.
> Dimension(M);
8
> Basis(CuspidalSubspace(M));
[
    q + q^5 + 2*q^6 + q^7 + O(q^8),
    q^2 + 2*q^5 + 3*q^6 + 2*q^7 + O(q^8),
    q^3 + 2*q^5 + 2*q^6 + 2*q^7 + O(q^8),
    q^4 + 2*q^5 + 3*q^6 + 3*q^7 + O(q^8),
    3*q^5 + 5*q^6 + 2*q^7 + O(q^8)
]
```

Next we create $M_4(\Gamma_0(8))$.

```
> M := ModularForms(8,4); M;
Space of modular forms on Gamma_0(8) of weight 4 and dimension 5 over
Integer Ring.
> Dimension(M);
5
> Basis(CuspidalSubspace(M));
[
    q - 4*q^3 - 2*q^5 + 24*q^7 + O(q^8)
]
```

Now we create the space $M_3(N, \varepsilon)$, where ε is a character of level 20, conductor 5 and order 4.

```
> G := DirichletGroup(20,CyclotomicField(EulerPhi(20)));
> chars := Elements(G);
> #chars;
8
> [Conductor(eps) : eps in chars];
[ 1, 4, 5, 20, 5, 20, 5, 20 ]
> exists(eps){eps : eps in chars | Conductor(eps) eq 5 and IsOdd(eps)};
true
> Order(eps);
4
> M := ModularForms(eps, 3); M;
Space of modular forms on Gamma_1(20) with character all conjugates of
[$.2], weight 3, and dimension 18 over Integer Ring.
> Dimension(EisensteinSubspace(M));
12
> Dimension(CuspidalSubspace(M));
```

```
6
```

Next we create the direct sum of the spaces $M_k(20, \varepsilon)$ as ε varies over the four mod 20 characters of order at most 2, for $k = 2$ and 3.

```
> G := DirichletGroup(20, RationalField());   // (Z/20Z)^* --> Q^*
> chars := Elements(G); #chars;
4
> M := ModularForms(chars,2); M;
Space of modular forms on Gamma_1(20) with characters all
conjugates of [1, $.1, $.2, $.1*$.2], weight 2, and dimension 12
over Integer Ring.
> M := ModularForms(chars,3); M;
Space of modular forms on Gamma_1(20) with characters all
conjugates of [1, $.1, $.2, $.1*$.2], weight 3, and dimension 16
over Integer Ring.
```

Now we create the spaces $M_k(\Gamma_1(20))$ for $k = 2, 3$.

```
> ModularForms(Gamma1(20));
Space of modular forms on Gamma_1(20) of weight 2 and dimension 22
over Integer Ring.
> ModularForms(Gamma1(20),3);
Space of modular forms on Gamma_1(20) of weight 3 and dimension 34
over Integer Ring.
```

We can also create the subspace of cuspforms directly:

```
> CuspForms(Gamma1(20));
Space of modular forms on Gamma_1(20) of weight 2 and dimension 3
over Integer Ring.
> CuspForms(Gamma1(20),3);
Space of modular forms on Gamma_1(20) of weight 3 and dimension 14
over Integer Ring.
```

138.2.1.1 Half-integral Weight Forms

Spaces of modular forms of half-integral weight can also be constructed. For these spaces, `CuspidalSubspace` and `qExpansionBasis` are available, as well as basic functionality such as element arithmetic. More functionality will be added in future releases.

The algorithm for determining the q-expansion basis involves computing those of related integral-weight spaces (of weight either one half smaller or one half larger, and appropriate level and character).

`HalfIntegralWeightForms(N, w)`

The space of half-integral weight forms on `Gamma0(N)` and weight w. Here N should be a multiple of 4 and w a positive element of $\mathbf{Z} + 1/2$.

`HalfIntegralWeightForms(chi, w)`

The space of half-integral weight forms on `Gamma1(N)` with character `chi` and weight w. The modulus of `chi` should be a multiple of 4, and w a positive element of $\mathbf{Z}+1/2$.

`HalfIntegralWeightForms(G, w)`

The space of half-integral weight forms on the congruence subgroup G and weight w. Here G must be contained in `Gamma0(4)`, and w is a positive element of $\mathbf{Z}+1/2$.

138.2.2 Base Extension

If M is a space of modular forms created using one of the constructors in Section 138.2.1, then the base ring of M is $\mathbf{Z}$. Thus we can base extend M to any ring R. The examples below illustrate some simple applications of `BaseExtend`.

`BaseExtend(M, R)`

The base extension of the space M of modular forms to the ring R and the induced map from M to `BaseExtend(M,R)`. The only requirement on R is that there is a natural coercion map from the base ring of M to R. For example, when `BaseRing(M)` is the integers, any ring R is allowed.

`BaseExtend(M, phi)`

The base extension of the space M of modular forms to the ring R using the map ϕ : `BaseRing(M)` $\to R$, and the induced map from M to `BaseExtend(M,R)`

Example H138E4

We first illustrate an Eisenstein series in $M_{12}(1)$ that is congruent to 1 modulo 3.

```
> M<q> := EisensteinSubspace(ModularForms(1,12));
> E12 := M.1; E12 + O(q^4);
691 + 65520*q + 134250480*q^2 + 11606736960*q^3 + O(q^4)
> M3<q3> := BaseExtend(M,GF(3));
> Dimension(M3);
1
> M3.1+O(q3^20);
1 + O(q3^20)
```

This congruence can be proved by noting that the coefficient of q^n in the q-expansion of $E_{12}/65520$, for any $n \geq 1$, is an eigenvalue of a Hecke operator, hence an integer, and that 65520 is divisible by 3. Because E_{12} is defined over $\mathbf{Z}$ the command `"E12Q/65520"` would result in an error, so we first base extend to $\mathbf{Q}$.

```
> MQ, phi := BaseExtend(M,RationalField());
> E12Q := phi(E12);
> E12Q/65520;
691/65520 + q + 2049*q^2 + 177148*q^3 + 4196353*q^4 + 48828126*q^5 +
```

```
362976252*q^6 + 1977326744*q^7 + O(q^8)
```

It is possible to base extend to almost any silly commutative ring.

```
> M := ModularForms(11,2);
> R := PolynomialRing(GF(17),3);
> MR<q> := BaseExtend(M,R); MR;
Space of modular forms on Gamma_0(11) of weight 2 and dimension 2 over
Polynomial ring of rank 3 over GF(17)
Lexicographical Order
Variables: $.1, $.2, $.3.
> f := MR.1;  f + O(q^5);
1 + 12*q^2 + 12*q^3 + 12*q^4 + O(q^5)
> f*(R.1+3*R.2) + O(q^4);
$.1 + 3*$.2 + (12*$.1 + 2*$.2)*q^2 + (12*$.1 + 2*$.2)*q^3 + O(q^4)
```

138.2.3 Elements

`M . i`

The ith basis vector of the space of modular forms M.

`M ! f`

The coercion of f into the space of modular forms M. Here f can be a modular form, a power series with absolute precision, or something that can be coerced into `RSpace(M)`.

`ModularForm(E)`

The modular form associated to the elliptic curve E over $\mathbf{Q}$. (See Section 138.18.)

Example H138E5

```
> M := ModularForms(Gamma0(11),2);
> M.1;
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + O(q^8)
> M.2;
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
> R<q> := PowerSeriesRing(Integers());
> f := M!(1 + q + 10*q^2 + O(q^3));
> f;
1 + q + 10*q^2 + 11*q^3 + 14*q^4 + 13*q^5 + 26*q^6 + 22*q^7 + O(q^8)
> Eltseq(f);
[ 1, 1 ]
```

`Eltseq` gives f as a linear combination of `M.1` and `M.2`. Next we coerce f into $M_2(\Gamma_0(22))$.

```
> M22 := ModularForms(Gamma0(22),2);
> g := M22!f; g;
```

```
1 + q + 10*q^2 + 11*q^3 + 14*q^4 + 13*q^5 + 26*q^6 + 22*q^7 + O(q^8)
> Eltseq(g);
[ 1, 1, 10, 11, 14 ]
```

The elliptic curve E defines an element of $M_2(\Gamma_0(11))$.

```
> E := EllipticCurve([ 0, -1, 1, -10, -20 ]);
> Conductor(E);
11
> f := ModularForm(E);
> f;
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
```

A new copy of $S_2(\Gamma_0(11))$ was created as the space containing f

```
> Sf := Parent(f);
> Mf := AmbientSpace(Parent(f));
> Sf; Mf;
Space of modular forms on Gamma_0(11) of weight 2 and dimension 1 over
Integer Ring
Space of modular forms on Gamma_0(11) of weight 2 and dimension 2 over
Integer Ring
> IsIdentical(M, Mf);
false
> M eq Mf;
true
> f in Sf, f in Mf, f in M;
true false false
```

There is a canonical way to coerce f into M using its q-expansion.

```
> IsCoercible(M, f);
true
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 - 2*q^9 - 2*q^10 + q^11 + O(q^12)
```

This coercion is applied automatically for some operations.

```
> f + M.1;
1 + q + 10*q^2 + 11*q^3 + 14*q^4 + 13*q^5 + 26*q^6 + 22*q^7 + 36*q^8 + 34*q^9
     + 46*q^10 + q^11 + O(q^12)
```

138.3 Bases

Any space of modular forms that can be created in MAGMA is of the form $M_{\mathbf{Z}} \otimes_{\mathbf{Z}} R$ for some ring R and some space $M_{\mathbf{Z}}$ of modular forms defined over $\mathbf{Z}$. The basis of M is the image in M of `Basis(M_Z)`, and `Basis(M_Z)` is in Hermite normal form.

Note that in order to determine this basis over $\mathbf{Z}$, it is necessary to compute q-expansions up to a "Sturm bound" for M (see `PrecisionBound)`. The precision used internally will therefore be at least as large as this bound. If desired, however, one can compute q-expansions to lower precision by working directly with spaces of modular symbols.

`Basis(M)`

The canonical basis of the space of modular forms or half-integral weight forms M.

`Basis(M, prec)`

`qExpansionBasis(M, prec)`

A sequence containing q-expansions (to the specified precision) of the elements of `Basis(M)`.

`PrecisionBound(M : parameters)`

`Exact` BOOLELT *Default :* `false`

An integer b such that $f+O(q^b)$ determines any modular form f in the given space M of modular forms or half-integral weight forms. If the optional parameter `Exact` is set to `true`, or if a q-expansion basis has already been computed for M, then the result is best-possible, ie the *smallest* integer b such that $f + O(q^b)$ determines any modular form f in M. Otherwise it is a "Sturm bound" similar to (although in some cases sharper than) the bounds given in section 9.4 of [Ste07].

Note: In some much older versions of MAGMA the default was `Exact := true`.

`RModule(M)`

`RSpace(M)`

`VectorSpace(M)`

`Ring` RNG *Default :*

An abstract free module isomorphic to the given space of modular forms M, over the same base ring (unless `Ring` is specified). The second returned object is a map from the abstract module to/from M.

This function is needed when one wants to use linear algebra functions on M (since in Magma, a space of modular forms is not a subtype of vector space).

Example H138E6

```
> M := ModularForms(Gamma1(16),3); M;
Space of modular forms on Gamma_1(16) of weight 3 and dimension 23
over Integer Ring.
> Dimension(CuspidalSubspace(M));
9
> SetPrecision(M,19);
> Basis(NewSubspace(CuspidalSubspace(M)))[1];
q - 76*q^8 + 39*q^9 + 132*q^10 - 44*q^11 + 84*q^12 - 144*q^13 -
232*q^14 + 120*q^15 + 160*q^16 + 158*q^17 - 76*q^18 + O(q^19)
```

We can print the whole basis to less precision:

```
> SetPrecision(M,10);
> Basis(NewSubspace(CuspidalSubspace(M)));
[
    q - 76*q^8 + 39*q^9 + O(q^10),
    q^2 + q^7 - 58*q^8 + 30*q^9 + O(q^10),
    q^3 + 2*q^7 - 42*q^8 + 18*q^9 + O(q^10),
    q^4 + q^7 - 26*q^8 + 13*q^9 + O(q^10),
    q^5 + 2*q^7 - 18*q^8 + 5*q^9 + O(q^10),
    q^6 + 2*q^7 - 12*q^8 + 3*q^9 + O(q^10),
    3*q^7 - 8*q^8 + O(q^10)
]
```

Note the coefficient 3 of q^7, which emphasizes that this is not the reduced echelon form over a field, but the image of a reduced form over the integers:

```
> MQ := BaseExtend(M,RationalField());
> Basis(NewSubspace(CuspidalSubspace(MQ)))[7];
3*q^7 - 8*q^8 + O(q^10)
```

138.4 q-Expansions

The following intrinsics give the q-expansion of a modular form (about the cusp ∞).

Note that q-expansions are printed by default only to precision $O(q^12)$. This may be adjusted using `SetPrecision` (see below), which should be used to control *printing only*; to control the amount of precision computed *internally*, instead use `qExpansion` or `qExpansionBasis` and specify the desired precision.

`qExpansion(f)`

`qExpansion(f, prec)`

`PowerSeries(f)`

`PowerSeries(f, prec)`

The q-expansion (at the cusp ∞) of the modular form (or half-integral weight form) f to absolute precision `prec`. This is an element of the power series ring over the base ring of the parent of f.

`Coefficient(f, n)`

The nth coefficient of the q-expansion of the modular form f.

`Precision(M)`

`SetPrecision(M, prec)`

When an element of the space M is printed, the q-expansion is displayed to this precision. The default value is 12.

Important note: This controls **only printing.** It does not control the precision used during calculations. For instance, the precision to which q-expansions are computed is controlled by the second argument in `qExpansion` and `qExpansionBasis`.

Example H138E7

In this example, we compute the q-expansion of a modular form $f \in M_3(\Gamma_1(11))$ in several ways.

```
> M := ModularForms(Gamma1(11),3); M;
Space of modular forms on Gamma_1(11) of weight 3 and dimension 15
over Integer Ring.
> f := M.1;
> f;
1 + O(q^8)
> qExpansion(f);
1 + O(q^8)
> Coefficient(f,16);  // f is a modular form, so has infinite precision
-5457936
> qExpansion(f,17);
1 + 763774*q^15 - 5457936*q^16 + O(q^17)
> PowerSeries(f,20);   // same as qExpansion(f,20)
1 + 763774*q^15 - 5457936*q^16 + 14709156*q^17 - 12391258*q^18 -
    21614340*q^19 + O(q^20)
```

The "big-oh" notation is supported via addition of a modular form and a power series.

```
> M<q> := Parent(f);
> Parent(q);
Power series ring in q over Integer Ring
> f + O(q^17);
1 + 763774*q^15 - 5457936*q^16 + O(q^17)
```

```
> 5*q - O(q^17) + f;
1 + 5*q + 763774*q^15 - 5457936*q^16 + O(q^17)
> 5*q + f;
1 + 5*q + O(q^8)
```

Default printing precision can be set using the command `SetPrecision`.

```
> SetPrecision(M,16);
> f;
1 + 763774*q^15 + O(q^16)
```

Example H138E8

The `PrecisionBound` intrinsic is related to Weierstrass points on modular curves. Let N be a positive integer such that $S = S_2(\Gamma_0(N))$ has dimension at least 2. Then the point ∞ is a Weierstrass point on $X_0(N)$ if and only if `PrecisionBound(S : Exact := true)-1 ne Dimension(S)`.

```
> function InftyIsWP(N)
>    S := CuspidalSubspace(ModularForms(Gamma0(N),2));
>    assert Dimension(S) ge 2;
>    return (PrecisionBound(S : Exact := true)-1) ne Dimension(S);
> end function;
> [<N,InftyIsWP(N)> : N in [97..100]];
[ <97, false>, <98, true>, <99, false>, <100, true> ]
```

It is an open problem to give a simple characterization of the integers N such that ∞ is a Weierstrass point on $X_0(N)$, though Atkin and others have made significant progress on this problem (see, e.g., 1967 Annals paper [Atk67]). I verified that if $N < 3223$ is square free, then ∞ is not a Weierstrass point on $X_0(N)$, which suggests a nice conjecture.

138.5 Arithmetic

`f + g`

The sum of the modular forms f and g.

`f + g`

The sum of the modular form f and the power series g. The q-expansion of f must be coercible into the parent of g. The sum $g+f$ is also defined, as are the differences $f-g$ and $g-f$.

`f - g`

The difference of the modular forms f and g.

`a * f`

The product of the scalar a and the modular form f.

`f / a`

The product of the scalar $1/a$ and the modular form f.

`f ^ n`

The power f^n of the modular form f, where $n \geq 1$ is an integer.

`f * g`

The product of the modular forms f and g. The only condition is that the base fields of f and g be the same. The weight of `f*g` is the sum of the weights of f and g.

Example H138E9

```
> M2 := ModularForms(Gamma0(11), 2);
> f := M2.1;
> g := M2.2;
> f;
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + O(q^8)
> g;
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
> f+g;
1 + q + 10*q^2 + 11*q^3 + 14*q^4 + 13*q^5 + 26*q^6 + 22*q^7 + O(q^8)
> 2*f;
2 + 24*q^2 + 24*q^3 + 24*q^4 + 24*q^5 + 48*q^6 + 48*q^7 + O(q^8)
> MQ,phi := BaseExtend(M2, RationalField());
> phi(2*f)/2;
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + O(q^8)
> f^2;
1 + 24*q^2 + 24*q^3 + 168*q^4 + 312*q^5 + 480*q^6 + 624*q^7 + O(q^8)
> Parent($1);
Space of modular forms on Gamma_0(11) of weight 4 and dimension 4 over
Integer Ring.
> M3 := ModularForms([DirichletGroup(11).1], 3); M3;
Space of modular forms on Gamma_1(11) with character all conjugates of
[$.1], weight 3, and dimension 3 over Integer Ring.
> M3.1*f;
1 + 12*q^2 + 2*q^3 + 6*q^4 - 126*q^5 - 168*q^6 - 384*q^7 + O(q^8)
> Parent($1);
Space of modular forms on Gamma_1(11) of weight 5 and dimension 25
over Integer Ring.
```

138.6 Predicates

`IsAmbientSpace(M)`

Returns `true` if and only if M is an ambient space. Ambient spaces are those space constructed in Section 138.2.1.

`IsCuspidal(M)`

Returns `true` if M is contained in the cuspidal subspace of the ambient space.

`IsEisenstein(M)`

Returns `true` if M is contained in the Eisenstein subspace of the ambient space.

`IsEisensteinSeries(f)`

Returns `true` if f is an Eisenstein newform or was computed using the intrinsic `EisensteinSeries`. (See Section 138.10.)

`IsGamma0(M)`

Returns `true` if M is a space of modular forms for $\Gamma_0(N)$.

`IsGamma1(M)`

Returns `true` if M was created explicitly as a space of modular forms for $\Gamma_1(N)$, or if the `AmbientSpace` of M is such a space. (Note that `IsGamma1` will return `false` for any space `ModularForms(chars,k)`, even if `chars` consists of all mod N Dirichlet characters.)

`IsNew(M)`

Returns `true` if M is contained in the new subspace of its `AmbientSpace`.

`IsNewform(f)`

Returns `true` if f was created using `Newforms`. (Sometimes `true` in other cases in which f is obviously a newform. In number theory, "newform" means "normalized eigenform that lies in the new subspace".)

`IsRingOfAllModularForms(M)`

Returns `true` if and only if M is the ring of all modular forms over a given ring.

Example H138E10

We illustrate each of the above predicates with some simple computations in $M_3(\Gamma_1(11))$.

```
> M := ModularForms(Gamma1(11),3);
> f := Newform(M,1);
> IsAmbientSpace(M);
true
> IsAmbientSpace(CuspidalSubspace(M));
false
> IsCuspidal(M);
false
> IsCuspidal(CuspidalSubspace(M));
true
> IsEisenstein(CuspidalSubspace(M));
false
> IsEisenstein(EisensteinSubspace(M));
true
> IsGamma1(M);
true
> IsNew(M);
true
> IsNewform(M.1);
false
> IsNewform(f);
true
> IsRingOfAllModularForms(M);
false
> Level(f);
11
> Level(M);
11
> Weight(f);
3
> Weight(M);
3
> Weight(M.1);
3
```

138.7 Properties

`AmbientSpace(M)`

The full space of modular forms, in which the given space had been created as a subspace.

`BaseRing(M)`

`CoefficientRing(M)`

The ring over which the given space of modular forms was defined.

`Degree(f)`

The number of Galois-conjugates of the modular form f over the prime subfield of (the fraction field of) the base ring of f.

`Dimension(M)`

The dimension of the space M of modular forms or half-integral weight forms.

For spaces defined using the `ModularForms` constructors, the procedure used to obtain the dimension is to count the relevant Eisenstein series, and apply a formula giving the dimension of the relevant space of cusp forms.

`DimensionByFormula(M)`

The dimension of the given space of modular forms or half-integral weight forms (which must be either a full space or the cuspidal subspace of a full space), as given by the formulas in the paper by Cohen and Oesterle (in 'Modular Forms in One Variable, VI', Lecture Notes in Math. 627).

`DimensionByFormula(N, k)`

`DimensionByFormula(chi, k)`

`DimensionByFormula(N, chi, k)`

`Cuspidal` BoolElt *Default :* `false`

The dimension of the full space of the modular forms or half-integral weight forms with level N, character chi (taken to be trivial if not specified) and weight k, as given by the formulas in the paper by Cohen and Oesterle (in 'Modular Forms in One Variable, VI', Lecture Notes in Math. 627).

If `Cuspidal` is set to `true`, then the dimension of the space of cusp forms is returned.

`DirichletCharacters(M)`

A sequence containing exactly one representative from each Galois-conjugacy class of Dirichlet characters associated to the space of modular forms M.

`DirichletCharacter(f)`

Suppose f is a newform, created using the `Newform` command. This returns a Dirichlet character that is, up to Galois conjugacy, the Nebentypus character of f.

`Eltseq(f)`

The sequence $[a_1, \ldots, a_n]$ such that $f = a_1 g_1 + \cdots + a_n g_n$, where $g_1, \ldots, g_n$ is the basis of the parent of the modular form f.

`Level(f)`

The level of the modular form f.

`Level(M)`

The level of the space of modular forms M.

`Weight(f)`

The weight of the modular form f, if it is defined.

`Weight(M)`

The weight of the space M of modular forms.

`WeightOneHalfData(H)`

A list of tuples describing a basis of the given space of forms of weight 1/2. Each tuple is a pair $< f, t >$, where t is an integer and f is a Dirichlet character. The tuple $< f, t >$ designates the sum over all integers n of $f(n)q^{(tn^2)}$.

Example H138E11

We illustrate each of the above properties with some simple computations in $M_3(\Gamma_1(11))$.

```
> M := ModularForms(Gamma1(11),3);
> Degree(M.1);
1
> f := Newform(M,1);
> Degree(f);
4
> Dimension(M);
15
> DirichletCharacters(M);
[
    1,
    $.1,
    $.1^2,
    $.1^5
]
> Level(f);
11
> Level(M);
```

```
11
> Weight(f);
3
> Weight(M);
3
> Weight(M.1);
3
```

138.8 Subspaces

The following functions compute the cuspidal, Eisenstein, and new subspaces.

`ZeroSubspace(M)`

The trivial subspace of the space of modular forms M.

`CuspidalSubspace(M)`

The subspace of forms f in M such that the constant term of the Fourier expansion of f at every cusp is 0.

`EisensteinSubspace(M)`

The Eisenstein subspace of the space of modular forms M.

`EisensteinProjection(f)`
`CuspidalProjection(f)`

The projection of a given modular form to the `EisensteinSubspace` or to the `CuspidalSubspace`. The sum of the two projections equals the original form (after coercion).

The base ring of the given form must contain the rationals.

`NewSubspace(M)`

The new subspace of the space of modular forms M.

`DihedralSubspace(M)`

For a space M of weight 1 forms, this returns the subspace spanned by the cusp forms attached to dihedral Galois representations.

Example H138E12

We compute a basis of q-expansions for each of the above subspace of $M_2(\Gamma_0(33))$.

```
> M := ModularForms(Gamma0(33),2); M;
Space of modular forms on Gamma_0(33) of weight 2 and dimension 6 over
Integer Ring.
> Basis(M);
[
    1 + O(q^8),
    q - q^5 + 2*q^7 + O(q^8),
    q^2 + 2*q^7 + O(q^8),
    q^3 + O(q^8),
    q^4 + q^5 + O(q^8),
    q^6 + O(q^8)
]
> Basis(CuspidalSubspace(M));
[
    q - q^5 - 2*q^6 + 2*q^7 + O(q^8),
    q^2 - q^4 - q^5 - q^6 + 2*q^7 + O(q^8),
    q^3 - 2*q^6 + O(q^8)
]
> Basis(EisensteinSubspace(M));
[
    1 + O(q^8),
    q + 3*q^2 + 7*q^4 + 6*q^5 + 8*q^7 + O(q^8),
    q^3 + 3*q^6 + O(q^8)
]
> Basis(NewSubspace(M));
[
    q + q^2 - q^3 - q^4 - 2*q^5 - q^6 + 4*q^7 + O(q^8)
]
> Basis(NewSubspace(EisensteinSubspace(M)));
[]
> Basis(NewSubspace(CuspidalSubspace(M)));
[
    q + q^2 - q^3 - q^4 - 2*q^5 - q^6 + 4*q^7 + O(q^8)
]
> ZeroSubspace(M);
Space of modular forms on Gamma_0(33) of weight 2 and dimension 0 over
Integer Ring.
> MQ := BaseChange(M, Rationals()); SetPrecision(MQ, 20);
> b := Basis(MQ); b[5];
q^4 + q^5 + 2*q^8 - q^9 + 2*q^10 + 2*q^11 - q^12 + 2*q^13 +
 2*q^14 - q^15 + 3*q^16 + 2*q^17 - 2*q^18 + 2*q^19 + O(q^20)
> CuspidalProjection(b[5]);
-1/10*q - 3/10*q^2 + 1/10*q^3 + 3/10*q^4 + 2/5*q^5 + 3/10*q^6 - 4/5*q^7
+ 1/2*q^8 - 3/10*q^9 + 1/5*q^10 - 1/10*q^11 - 3/10*q^12 + 3/5*q^13 -
2/5*q^14 - 2/5*q^15 - 1/10*q^16 + 1/5*q^17 + 1/10*q^18 + O(q^20)
```

```
> EisensteinProjection(b[5]);
1/10*q + 3/10*q^2 - 1/10*q^3 + 7/10*q^4 + 3/5*q^5 - 3/10*q^6 + 4/5*q^7
+ 3/2*q^8 - 7/10*q^9 + 9/5*q^10 + 21/10*q^11 - 7/10*q^12 + 7/5*q^13 +
12/5*q^14 - 3/5*q^15 + 31/10*q^16 + 9/5*q^17 - 21/10*q^18 + 2*q^19 +
O(q^20)
> MQ! $1 + MQ! $2;  // Add the previous two answers, inside MQ
q^4 + q^5 + 2*q^8 - q^9 + 2*q^10 + 2*q^11 - q^12 + 2*q^13 +
 2*q^14 - q^15 + 3*q^16 + 2*q^17 - 2*q^18 + 2*q^19 + O(q^20)
```

The two projections sum to the original form.

138.9 Operators

Each space M of modular forms comes equipped with a commuting family $T_1, T_2, T_3, \ldots$ of linear operators acting on it called the *Hecke operators*. Unfortunately, at present, the computation of Hecke and other operators on spaces of modular forms with nontrivial character has not yet been implemented, though computation of characteristic polynomials of Hecke operators is supported.

`HeckeOperator(M, n)`

The matrix representing the nth Hecke operator T_n with respect to `Basis(M)`. (Currently M must be a space of modular forms with trivial character and integral weight ≥ 2.)

`HeckeOperator(n,f)`

The image under the Hecke operator T_n of the given modular form.

`HeckePolynomial(M, n` : *parameters*`)`

`Proof` BoolElt *Default :* `true`

The characteristic polynomial of the nth Hecke operator T_n. In some situations this is more efficient than `CharacteristicPolynomial(HeckeOperator(M,n))` or any of its variants. Note that M can be an arbitrary space of modular forms.

`AtkinLehnerOperator(M, q)`

The matrix representing the qth Atkin-Lehner involution W_q on M with respect to `Basis(M)`. (Currently M must be a cuspidal space of modular forms with trivial character and integral weight ≥ 2.)

`AtkinLehnerOperator(q,f)`

The image under the involution w_q of the given modular form.

Example H138E13

First we compute a characteristic polynomial on $S_2(\Gamma_1(13))$ over both $\mathbf{Z}$ and the finite field $\mathbf{F}_2$.

```
> R<x> := PolynomialRing(Integers());
> S := CuspForms(Gamma1(13),2);
> HeckePolynomial(S, 2);
x^2 + 3*x + 3
> S2 := BaseExtend(S, GF(2));
> R<y> := PolynomialRing(GF(2));
> Factorization(HeckePolynomial(S2,2));
[
    <y^2 + y + 1, 1>
]
```

Next we compute a Hecke operator on $M_4(\Gamma_0(14))$.

```
> M := ModularForms(Gamma0(14),4);
> T := HeckeOperator(M,2);
> T;
[  1   0   0   0   0   0   0 240]
[  0   0   0   0  18  12  50 100]
[  0   1   0   0  -2  18  12 -11]
[  0   0   0   0   1  22  25  46]
[  0   0   1   0  -1 -16 -20 -82]
[  0   0   0   0  -1  -6  -9 -38]
[  0   0   0   1   3   9  15  39]
[  0   0   0   0   0   0   0   8]
> Parent(T);
Full Matrix Algebra of degree 8 over Integer Ring
> Factorization(CharacteristicPolynomial(T));
[
    <x - 8, 2>,
    <x - 2, 1>,
    <x - 1, 2>,
    <x + 2, 1>,
    <x^2 + x + 8, 1>
]
> f := M.1;
> f*T;
1 + 240*q^7 + O(q^8)
> M.1 + 240*M.8;
1 + 240*q^7 + O(q^8)
```

This example demonstrates the Atkin-Lehner involution W_3 on $S_2(\Gamma_0(33))$.

```
> M := ModularForms(33,2);
> S := CuspidalSubspace(M);
> W3 := AtkinLehnerOperator(S, 3);
> W3;
[   1    0    0]
```

```
[ 1/3  1/3 -4/3]
[ 1/3 -2/3 -1/3]
> Factorization(CharacteristicPolynomial(W3));
[
    <x - 1, 2>,
    <x + 1, 1>
]
> f := S.2;
> f*W3;
1/3*q + 1/3*q^2 - 4/3*q^3 - 1/3*q^4 - 2/3*q^5 + 5/3*q^6
     + 4/3*q^7 + O(q^8)
```

The Atkin-Lehner and Hecke operators need not commute:

```
> T3 := HeckeOperator(S, 3);
> T3;
[ 0 -2 -1]
[ 0 -1  1]
[ 1 -2 -1]
> T3*W3 - W3*T3 eq 0;
false
```

138.10 Eisenstein Series

The intrinsics below require that the base ring of M has characteristic 0. To compute mod p eigenforms, use the `Reduction` intrinsic (see Section 138.14).

`EisensteinSeries(M)`

List of the Eisenstein series associated to the modular forms space M. By "associated to" we mean that the Eisenstein series lies in $M \otimes \mathbf{C}$.

`IsEisensteinSeries(f)`

Returns `true` if the modular form f was created using `EisensteinSeries`.

`EisensteinData(f)`

The data $< \chi, \psi, t, \chi', \psi' >$ that defines the Eisenstein series (modular form) f. Here χ is a primitive character of conductor S, ψ is primitive of conductor M, and MSt divides N, where N is the level of f. (The additional characters χ' and ψ' are equal to χ and ψ respectively, except they take values in the big field $\mathbf{Q}(\zeta_{\phi(N)})^*$ instead of $\mathbf{Q}(\zeta_n)^*$, where n is the order of χ or ψ.) The Eisenstein series associated to (χ, ψ, t) has q-expansion

$$c_0 + \sum_{m \geq 1} \left(\sum_{n|m} \psi(n) n^{k-1} \chi(m/n) \right) q^{mt},$$

where $c_0 = 0$ if $S > 1$ and $c_0 = L(1-k, \psi)/2$ if $S = 1$.

Example H138E14

We illustrate the above intrinsics by computing the Eisenstein series in $M_3(\Gamma_1(12))$.

```
> M := ModularForms(Gamma1(12),3); M;
Space of modular forms on Gamma_1(12) of weight 3 and dimension 13
over Integer Ring.
> E := EisensteinSubspace(M); E;
Space of modular forms on Gamma_1(12) of weight 3 and dimension 10
over Integer Ring.
> s := EisensteinSeries(E); s;
[*
-1/9 + q - 3*q^2 + q^3 + 13*q^4 - 24*q^5 - 3*q^6 + 50*q^7 + O(q^8),
-1/9 + q^2 - 3*q^4 + q^6 + O(q^8),
-1/9 + q^4 + O(q^8),
-1/4 + q + q^2 - 8*q^3 + q^4 + 26*q^5 - 8*q^6 - 48*q^7 + O(q^8),
-1/4 + q^3 + q^6 + O(q^8),
q + 3*q^2 + 9*q^3 + 13*q^4 + 24*q^5 + 27*q^6 + 50*q^7 + O(q^8),
q^2 + 3*q^4 + 9*q^6 + O(q^8),
q^4 + O(q^8),
q + 4*q^2 + 8*q^3 + 16*q^4 + 26*q^5 + 32*q^6 + 48*q^7 + O(q^8),
q^3 + 4*q^6 + O(q^8)
*]
> a := EisensteinData(s[1]); a;
<1, $.1, 1, 1, $.2>
> Parent(a[2]);
Group of Dirichlet characters of modulus 3 over Rational Field
> Order(a[2]);
2
> Parent(a[5]);
Group of Dirichlet characters of modulus 12 over Cyclotomic Field of
order 4 and degree 2
> Parent(s[1]);
Space of modular forms on Gamma_1(12) of weight 3 and dimension 10
over Rational Field.
> IsEisensteinSeries(s[1]);
true
```

138.11 Weight Half Forms

Modular forms of weight 1/2 are constructed directly as q-expansions following the explicit description given by Serre and Stark (see [SS77]).

`WeightOneHalfData(M)`

For a space M of modular forms of weight 1/2, this returns the basis of the space as described by Serre and Stark. A list of tuples is returned; each tuple contains a character ψ and an integer t, and the corresponding modular form is the theta series defined as the sum over all integers n of $\psi(n)q^{(tn^2)}$.

138.12 Weight One Forms

Modular forms of weight 1 can be defined using the usual constructors. For these spaces, the `Dimension`, the `CuspidalSubspace` and `EisensteinSubspace`, `EisensteinSeries`, a `qExpansionBasis`, and Hecke operators are available, as well as basic functionality such as element arithmetic.

The algorithm used to determine spaces of weight 1 forms is as follows. The Eisenstein series are constructed directly as q-expansions. The cuspidal eigenforms correspond to Galois representations; those corresponding to dihedral Galois representations are obtained explicitly (from characters on ray class groups of quadratic fields). If the dihedral forms span the full space of cusp forms, this is proved by comparing with suitable spaces of integral weight forms; if not, a q-expansion basis for the cuspidal space is obtained using the integral-weight spaces (this is the most time-consuming part of the process).

`DihedralForms(M)`

For a space of weight 1, this returns the cuspidal eigenforms in M corresponding to dihedral Galois representations, broken up according to character. A list of tuples is returned; each tuple contains an element of the `DirichletCharacters` of M, followed by a list of eigenforms.

138.13 Newforms

In this section we describe how to compute both cuspidal and Eisenstein newforms.

The intrinsics below require that the base ring of M has characteristic 0. To compute mod p eigenforms, use the `Reduction` intrinsic (see Section 138.14).

`NumberOfNewformClasses(M : parameters)`

`Proof` BoolElt *Default :* `true`

The number of Galois conjugacy-classes of newforms associate to the modular forms space M, which must have base ring $\mathbf{Z}$ or $\mathbf{Q}$. By "associated to" we mean that the newform lies in $M \otimes \mathbf{C}$.

`Newform(M, i, j : parameters)`

`Proof` BoolElt *Default :* `true`

The jth Galois-conjugate newform in the ith Galois-orbit of newforms in the space of modular forms M, which must have base ring $\mathbf{Z}$ or $\mathbf{Q}$.

`Newform(M, i : parameters)`

`Proof` BoolElt *Default :* `true`

The first Galois-conjugate newform in the ith orbit in the space of modular forms M, which must have base ring $\mathbf{Z}$ or $\mathbf{Q}$.

`Newforms(M : parameters)`

`Proof` BoolElt *Default :* `true`

Sort list of the newforms associated to the space of modular forms M divided up into Galois orbits.

`Newforms(I, M)`

Use this intrinsic to find the newforms associated to the space of modular forms M with prespecified eigenvalues. Here I is a sequence $[\langle p_1, f_1(x)\rangle, ..., \langle p_n, f_n(x)\rangle]$ of pairs. Each pair consists of a prime number that does not divide the level of M and a polynomial. This intrinsic returns the set of newforms $\sum a_n q^n$ in M such that $f_n(a_{p_n}) = 0$. (This intrinsic only works when M is cuspidal and defined over $\mathbf{Q}$ or $\mathbf{Z}$.)

Example H138E15

We compute the newforms in $M_5(\Gamma_1(8))$.

```
> M := ModularForms(Gamma1(8),5); M;
Space of modular forms on Gamma_1(8) of weight 5 and dimension 11 over
Integer Ring.
> NumberOfNewformClasses(M);
4
> Newforms(M);
[* [*
q + 4*q^2 - 14*q^3 + 16*q^4 - 56*q^6 + O(q^8)
*], [*
q + 1/24*(a - 30)*q^2 + 6*q^3 + 1/12*(-a - 162)*q^4 + 1/3*(-a + 6)*q^5
+ 1/4*(a - 30)*q^6 + 1/3*(2*a - 12)*q^7 + O(q^8),
q + 1/24*(b - 30)*q^2 + 6*q^3 + 1/12*(-b - 162)*q^4 + 1/3*(-b + 6)*q^5
+ 1/4*(b - 30)*q^6 + 1/3*(2*b - 12)*q^7 + O(q^8)
*], [*
57/2 + q + q^2 + 82*q^3 + q^4 - 624*q^5 + 82*q^6 - 2400*q^7 + O(q^8)
*], [*
q + 16*q^2 + 82*q^3 + 256*q^4 + 624*q^5 + 1312*q^6 + 2400*q^7 + O(q^8)
*] *]
> Newform(M,1);
```

```
q + 4*q^2 - 14*q^3 + 16*q^4 - 56*q^6 + O(q^8)
> Newform(M,2);
q + 1/24*(a - 30)*q^2 + 6*q^3 + 1/12*(-a - 162)*q^4 + 1/3*(-a + 6)*q^5
+ 1/4*(a - 30)*q^6 + 1/3*(2*a - 12)*q^7 + O(q^8)
> Parent(Newform(M,2));
Space of modular forms on Gamma_1(8) of weight 5 and dimension 2 over
Number Field with defining polynomial x^2 - 12*x + 8676 over the
Rational Field.
> Newform(M,2,2);
q + 1/24*(b - 30)*q^2 + 6*q^3 + 1/12*(-b - 162)*q^4 + 1/3*(-b + 6)*q^5
+ 1/4*(b - 30)*q^6 + 1/3*(2*b - 12)*q^7 + O(q^8)
> IsEisensteinSeries(Newform(M,1));
false
> IsEisensteinSeries(Newform(M,2));
false
> IsEisensteinSeries(Newform(M,3));
true
> IsEisensteinSeries(Newform(M,4));
true
```

The following example demonstrates picking out a newform in $S_2(\Gamma_0(65))$ with prespecified eigenvalues.

```
> S := CuspForms(65,2);
> R<x> := PolynomialRing(IntegerRing());
> I := [<3,x+2>];
> Newforms(I,S);
[* [*
q - q^2 - 2*q^3 - q^4 - q^5 + 2*q^6 - 4*q^7 + O(q^8)
*] *]
> Factorization(HeckePolynomial(S, 2));
[
    <x + 1, 1>,
    <x^2 - 3, 1>,
    <x^2 + 2*x - 1, 1>
]
> I := [<2,x^2-3>];
> Newforms(I,S);
[* [*
q + a*q^2 + (-a + 1)*q^3 + q^4 - q^5 + (a - 3)*q^6 + 2*q^7 + O(q^8),
q + b*q^2 + (-b + 1)*q^3 + q^4 - q^5 + (b - 3)*q^6 + 2*q^7 + O(q^8)
*] *]
```

138.13.1 Labels

It is possible to obtain the galois-conjugacy class of a newform by giving a descriptive label as an argument to `Newforms`. The format of the label is as follows:

`[G0N or G1N][Level]k[Weight][Isogeny Class].`

Some example labels are `"G0N11k2A"`, `"G0N1k12A"`, `"G1N17k2B"`, and `"G1N9k3B"`. If the string `"G0N"` or `"G1N"` is omitted, then the default is `"G0N"`. Thus the following are also valid: `"11k2A"`, `"1k12A"`, `"37k4A"`. If `k[Weight]` is omitted, then the default is weight 2, so the following are valid and all refer to weight 2 modular forms on some $\Gamma_0(N)$: `"11A"`, `"37A"`, `"65B"`. In order, possibilities for the isogeny class are as follows:

`A, B, C, ..., Y, Z, AA, BB, CC, ..., ZZ, AAA, BBB, CCC, ...`.

This is essentially the notation used in [Cre97] for isogeny classes, though sometimes for levels ≤ 450 the ordering differs from that in [Cre97].

Suppose s is a valid label, and let M be the space of modular forms that contains `ModularForm(s)`. Then `ModularForm(s)` is by definition `Newforms(M)[i]` where the isogeny class in the label s is the ith isogeny class. For example `C` corresponds to the 3rd isogeny class and `BB` corresponds to the 28th.

`Newforms(label)`

The Galois-conjugacy class(es) of newforms described by the string *label*. See the introduction for a description of the notation used for the label.

Example H138E16

We give many examples of constructing newforms using labels.

```
> Newforms("11A");
[*
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
*]
>  Newforms("G0N11k2A");
[*
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
*]
>  Newforms("G0N1k12A");
[*
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 +
O(q^8)
*]
>  Newforms("G1N17k2B");
[*
q + (-a^3 + a^2 - 1)*q^2 + (a^3 - a^2 - a - 1)*q^3 + (2*a^3 - a^2 +
2*a)*q^4 + (-a^3 - a^2)*q^5 + (-a^3 + a^2 - a + 1)*q^6 + (-a^3 + a^2 +
a - 1)*q^7 + O(q^8),
q + (-b^3 + b^2 - 1)*q^2 + (b^3 - b^2 - b - 1)*q^3 + (2*b^3 - b^2 +
2*b)*q^4 + (-b^3 - b^2)*q^5 + (-b^3 + b^2 - b + 1)*q^6 + (-b^3 + b^2 +
b - 1)*q^7 + O(q^8),
q + (-c^3 + c^2 - 1)*q^2 + (c^3 - c^2 - c - 1)*q^3 + (2*c^3 - c^2 +
```

```
2*c)*q^4 + (-c^3 - c^2)*q^5 + (-c^3 + c^2 - c + 1)*q^6 + (-c^3 + c^2 +
c - 1)*q^7 + O(q^8),
q + (-d^3 + d^2 - 1)*q^2 + (d^3 - d^2 - d - 1)*q^3 + (2*d^3 - d^2 +
2*d)*q^4 + (-d^3 - d^2)*q^5 + (-d^3 + d^2 - d + 1)*q^6 + (-d^3 + d^2 +
d - 1)*q^7 + O(q^8)
*]
> Newforms("G1N9k3B");
[*
1/3*(-5*zeta_6 - 2) + q + (4*zeta_6 + 1)*q^2 + q^3 + (20*zeta_6 -
15)*q^4 + (-25*zeta_6 + 26)*q^5 + (4*zeta_6 + 1)*q^6 + (-49*zeta_6 +
1)*q^7 + O(q^8),
1/3*(5*zeta_6 - 7) + q + (-4*zeta_6 + 5)*q^2 + q^3 + (-20*zeta_6 +
5)*q^4 + (25*zeta_6 + 1)*q^5 + (-4*zeta_6 + 5)*q^6 + (49*zeta_6 -
48)*q^7 + O(q^8)
*]
> Newforms("11k2A");
[*
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
*]
> Newforms("11A");
[*
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
*]
> Newforms("1k12A");
[*
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 +
O(q^8)
*]
> Newforms("37k4A");
[*
q + a*q^2 + 1/8*(-a^3 - 9*a^2 - 26*a - 22)*q^3 + (a^2 - 8)*q^4 +
1/8*(13*a^3 + 85*a^2 + 50*a - 186)*q^5 + 1/8*(-3*a^3 - 27*a^2 - 38*a +
6)*q^6 + 1/4*(-19*a^3 - 119*a^2 - 30*a + 170)*q^7 + O(q^8),
q + b*q^2 + 1/8*(-b^3 - 9*b^2 - 26*b - 22)*q^3 + (b^2 - 8)*q^4 +
1/8*(13*b^3 + 85*b^2 + 50*b - 186)*q^5 + 1/8*(-3*b^3 - 27*b^2 - 38*b +
6)*q^6 + 1/4*(-19*b^3 - 119*b^2 - 30*b + 170)*q^7 + O(q^8),
q + c*q^2 + 1/8*(-c^3 - 9*c^2 - 26*c - 22)*q^3 + (c^2 - 8)*q^4 +
1/8*(13*c^3 + 85*c^2 + 50*c - 186)*q^5 + 1/8*(-3*c^3 - 27*c^2 - 38*c +
6)*q^6 + 1/4*(-19*c^3 - 119*c^2 - 30*c + 170)*q^7 + O(q^8),
q + d*q^2 + 1/8*(-d^3 - 9*d^2 - 26*d - 22)*q^3 + (d^2 - 8)*q^4 +
1/8*(13*d^3 + 85*d^2 + 50*d - 186)*q^5 + 1/8*(-3*d^3 - 27*d^2 - 38*d +
6)*q^6 + 1/4*(-19*d^3 - 119*d^2 - 30*d + 170)*q^7 + O(q^8)
*]
> Newforms("37k2");
[* [*
q - 2*q^2 - 3*q^3 + 2*q^4 - 2*q^5 + 6*q^6 - q^7 + O(q^8)
*], [*
q + q^3 - 2*q^4 - q^7 + O(q^8)
```

```
*], [*
3/2 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^5 + 12*q^6 + 8*q^7 + O(q^8)
*] *]
```

138.14 Reductions and Embeddings

`Reductions(f, p)`

The mod p reductions of the modular forms f, where $p \in \mathbf{Z}$ is a prime number and f is a modular form over a number field (or the rationals or integers). Because of denominators, the list of reductions might be empty. (In some cases when f is defined over a field of large degree, the algorithm that I've implemented is definitely not close to optimal. I know a much better algorithm, but haven't implemented it yet.)

`pAdicEmbeddings(f, p)`

The p-adic embeddings of the modular form f.

`ComplexEmbeddings(f)`

The complex embeddings of the modular form f.

Example H138E17

We compute various reductions and embeddings of a degree 3 newform in $S_2(\Gamma_0(47))$.

```
> M := ModularForms(Gamma0(47),2);
> f := Newform(M,1);
> Degree(f);
4
> f;
q + a*q^2 + (a^3 - a^2 - 6*a + 4)*q^3 + (a^2 - 2)*q^4 + (-4*a^3 +
2*a^2 + 20*a - 10)*q^5 + (-a^2 - a + 1)*q^6 + (3*a^3 - a^2 - 16*a +
7)*q^7 + O(q^8)
> Parent(f);
Space of modular forms on Gamma_0(47) of weight 2 and dimension 4 over
Number Field with defining polynomial x^4 - x^3 - 5*x^2 + 5*x - 1 over
the Rational Field.
> Reductions(f,3);
[* [*
q + 2*q^2 + 2*q^3 + 2*q^4 + q^6 + q^7 + O(q^8)
*], [*
q + $.1^4*q^2 + $.1^25*q^3 + $.1^15*q^4 + $.1^20*q^5 + $.1^3*q^6 +
$.1^3*q^7 + O(q^8),
q + $.1^10*q^2 + $.1^17*q^3 + $.1^5*q^4 + $.1^24*q^5 + $.1*q^6 +
$.1*q^7 + O(q^8),
q + $.1^12*q^2 + $.1^23*q^3 + $.1^19*q^4 + $.1^8*q^5 + $.1^9*q^6 +
```

```
$.1^9*q^7 + O(q^8)
*] *]
```

The reductions are genuine modular forms, so we can compute them to higher precision later.

```
> f3 := Reductions(f,3)[1][1];
> Type(f3);
ModFrmElt
> Parent(f3);
Space of modular forms on Gamma_0(47) of weight 2 and dimension 4 over
Finite field of size 3.
> PowerSeries(f3,15);
q + 2*q^2 + 2*q^3 + 2*q^4 + q^6 + q^7 + q^9 + q^12 + 2*q^14 + O(q^15)
> f3^2;
q^2 + q^3 + 2*q^4 + q^7 + O(q^8)
> pAdicEmbeddings(f,3)[1][1];
q + (1383893738 + O(3^20))*q^2 + (288495368 + O(3^20))*q^3 +
(1448516780 + O(3^20))*q^4 + (291254407*3 + O(3^20))*q^5 + (654373882
+ O(3^20))*q^6 - (443261663 + O(3^20))*q^7 + O(q^8)
```

The p-adic precision can be increased by recreating p-adic field with higher precision.

```
> _ := pAdicField(3 : Precision := 30);
> pAdicEmbeddings(f,3)[1][1];
q + (27072777983102 + O(3^30))*q^2 - (7262683411915 + O(3^30))*q^3 +
(102359491392536 + O(3^30))*q^4 - (26022742991723*3 + O(3^30))*q^5 +
(76458862719010 + O(3^30))*q^6 + (31185356420881 + O(3^30))*q^7 +
O(q^8)
> ComplexEmbeddings(f)[1][1];
q + 0.28384129078391142728240587955711710388*q^2 +
2.23925429984775850028724717074345372615990081162*q^3 -
1.91943412164612303785432431586190778394853582709*q^4 -
4.25351411923443363456996356947846775230044737382*q^5 +
0.63559283086221161057191843630479068005905688171 2*q^6 +
2.44657723781885227971790463962077981836158867144*q^7 + O(q^8)
```

138.15 Congruences

`CongruenceGroup(M1, M2, prec)`

A group C that measures all possible congruences (to precision *prec*) between some modular form in M_1 and some modular form in M_2. The group C is defined as follows. Let W_1 be the finite-rank $\mathbf{Z}$-module q-exp$(M_1) \cap \mathbf{Z}[[q]]$ and let W_2 be q-exp$(M_2) \cap \mathbf{Z}[[q]]$. Let V be the saturation of $W_1 + W_2$ in $\mathbf{Z}[[q]]$. Then $C = V/(W_1 + W_2)$.

`CongruenceGroupAnemic(M1, M2, prec)`

Analogous to `CongruenceGroup`, but now considering congruences that hold for all q-expansion coefficients a_n with n coprime to the levels of both $M1$ and $M2$ (rather than for all q-expansion coefficients).

Example H138E18

We verify that the newform corresponding to the first elliptic curve of rank 2 is congruent modulo 5 to some Galois-conjugate newform corresponding to the winding quotient of $J_0(389)$ (there is also a congruence modulo 2 to some conjugate form).

```
> M := ModularForms(Gamma0(389),2);
> f := Newform(M,1);
> Degree(f);
1
> g := Newform(M,5);
> Degree(g);
20
> CongruenceGroup(Parent(f),Parent(g),30);
Abelian Group isomorphic to Z/20
Defined on 1 generator
Relations:
    20*$.1 = 0
```

The congruence can be seen directly by computing the reductions of f and g modulo 5:

```
> fmod5 := Reductions(f,5);
> gmod5 := Reductions(g,5);  // takes a few seconds.
> #gmod5;
7
> #fmod5;
1
> [gbar : gbar in gmod5 | #gbar eq 1];
[ [*
q + 4*q^2 + q^3 + 4*q^4 + q^5 + 4*q^6 + O(q^8)
*], [*
q + 3*q^2 + 3*q^3 + 2*q^4 + 2*q^5 + 4*q^6 + O(q^8)
*] ]
> fmod5[1][1];
q + 3*q^2 + 3*q^3 + 2*q^4 + 2*q^5 + 4*q^6 + O(q^8)
```

138.16 Overconvergent Modular Forms

These routines compute characteristic series of operators on overconvergent modular forms. While these are p-adic modular forms, the result also gives information about classical spaces: it determines the characteristic series up to a congruence (details are given below). The big advantage of this approach is that extremely large weights can be handled (the method works by indirectly computing a small part of the large space).

The algorithm has a running time which is linear in $\log(k)$, where k is the weight. The implementation for level 1 is well optimized.

The algorithm is given in Algorithms 1 and 2 of [Lau11]. Suggestions of David Loeffler and John Voight are used in generating certain spaces of classical modular forms for level $N \geq 2$.

`OverconvergentHeckeSeriesDegreeBound(p, N, k, m)`

This returns a bound on the degree of the characteristic series modulo p^m of the Atkin U_p operator on the space of overconvergent p-adic modular forms of (even) weight k and level $\Gamma_0(N)$. This bound is due to Daqing Wan and depends only on k modulo $p-1$ rather than k itself.

`OverconvergentHeckeSeries(p, N, k, m)`

`OverconvergentHeckeSeries(p, N, k, m)`

`OverconvergentHeckeSeries(p, chi, k, m)`

`OverconvergentHeckeSeries(p, chi, k, m)`

`WeightBound` RngIntElt *Default :*

This returns the characteristic series $P(t)$ modulo p^m of the Atkin U_p operator acting on the space of overconvergent p-adic modular forms of weight k and level $\Gamma_0(N)$, or with character `chi` on $\Gamma_1(N)$ when `chi` is the second argument.

The first argument must be a prime $p \geq 5$ not dividing N.

The third argument may be either a single weight or a sequence of weights, in which case the function returns a sequence containing the characteristic series for each weight. The given weights must be congruent to each other modulo $p-1$. Note that it is more efficient to compute several weights together rather than separately. (In particular, the part of the algorithm which builds up spaces from classical spaces of low weight can be done for all the weights together.)

When the second argument is a character `chi`, it must take values in a cyclotomic field. In this case the function returns a second value: a p-adic integer that defines the embedding of the cyclotomic field in the p-adic field.

When $m \leq k-1$, by Coleman's theorem, $P(t)$ is also the reverse characteristic polynomial modulo p^m of the Atkin U_p operator on the space of classical modular forms of level $\Gamma_0(Np)$ and weight k, In addition, when $m \leq (k-2)/2$, P(t) is the reverse characteristic polynomial modulo p^m of the Hecke T_p operator on the space of classical modular forms of level $\Gamma_0(N)$ and weight k.

The optional parameter `WeightBound` is a bound on the weight of the forms used to generate certain spaces of classical modular forms that are used at one point in the algorithm (for level $N \geq 2$). The default value is 6 for the case of $\Gamma_0(N)$ and for the case of even character `chi`, and 3 for odd `chi`. For most levels, the algorithm terminates more quickly with `WeightBound` 4 rather than 6. For some small levels, the algorithm may fail to terminate with these settings, however it will terminate when `WeightBound` is set to a sufficiently large value. There is some randomization in the algorithm, which may also cause variation in running time. The output is proven correct in all instances where the algorithm terminates.

Example H138E19

```
> _<t> := PolynomialRing(Integers()); // use t for printing
> time OverconvergentHeckeSeries(5,11,10000,5);
625*t^8 + 375*t^7 + 1275*t^6 - 875*t^5 - 661*t^4 + 335*t^3 + 1189*t^2 + 861*t + 1
Time: 0.300
> time OverconvergentHeckeSeries(5,11,10000,5 : WeightBound := 4);
625*t^8 + 375*t^7 + 1275*t^6 - 875*t^5 - 661*t^4 + 335*t^3 + 1189*t^2 + 861*t + 1
Time: 0.130
```

Note that the same result was obtained each time, but the weight bound reduced the computational work involved.

```
> OverconvergentHeckeSeries(7,3,[1000,1006],5);
[
    7203*t^6 - 4116*t^5 + 6713*t^4 - 700*t^3 - 4675*t^2 - 4426*t + 1,
    2401*t^6 + 8134*t^4 - 3976*t^3 + 5160*t^2 + 5087*t + 1
]
> Pseq := OverconvergentHeckeSeries(7,3,[1000,1006],5: WeightBound := 4);
> assert Pseq eq $1; // check that it is the same as the previous result
> OverconvergentHeckeSeriesDegreeBound(7,3,1000,5);
9
> OverconvergentHeckeSeries(29,1,10000,10);
    134393786323419*t^8 - 174295724342741*t^7 - 174857545225000*t^6
    - 153412311426730*t^5 + 41820892464727*t^4 - 148803482429283*t^3
    - 111232323996568*t^2 + 165679475331974*t + 1
```

138.17 Algebraic Relations

`Relations(M, d, prec)`

The relations of degree d satisfied by the q-expansions of in the space M of modular forms. The q-expansions are computed to precision *prec*. If prec is too small, this intrinsic might return relations that are not really satisfied by the modular forms. To be sure of your result, `prec` must be at least as large as `PrecisionBound(M2)`, where M_2 has the same level as M and weight d times the weight of M.

Example H138E20

We compute an equation that defines the canonical embedding of $X_0(34)$.

```
> S := CuspidalSubspace(ModularForms(Gamma0(34)));
> Relations(S, 4, 20);
[
    a^3*c - a^2*b^2 - 3*a^2*c^2 + 2*a*b^3 + 3*a*b^2*c - 3*a*b*c^2 +
        4*a*c^3 - b^4 + 4*b^3*c - 6*b^2*c^2 + 4*b*c^3 - 2*c^4
]
[
    (0  0  1 -1  0 -3  2  3 -3  4 -1  4 -6  4 -2)
]
> // a, b, and c correspond to the cusp forms S.1, S.2 and S.3:
> S.1;
q - 2*q^4 - 2*q^5 + 4*q^7 + O(q^8)
> S.2;
q^2 - q^4 + O(q^8)
> S.3;
q^3 - 2*q^4 - q^5 + q^6 + 4*q^7 + O(q^8)
```

Next we compute the canonical embedding of $X_0(75)$.

```
> S := CuspidalSubspace(ModularForms(Gamma0(75)));
> R := Relations(S, 2, 20); R;
[
    a*c - b^2 - d^2 - 4*e^2,
    a*d - b*c + b*e + d*e - 3*e^2,
    a*e - b*d - c*e
]
> // NOTE: It is much faster to compute in the power
> // series ring than the ring of modular forms!
> a, b, c, d, e := Explode([PowerSeries(f,20) : f in Basis(S)]);
> a*c - b^2 - d^2 - 4*e^2;
O(q^21)
```

The connection between the above computations and models for modular curves is discussed in Steven Galbraith's Oxford Ph.D. thesis.

138.18 Elliptic Curves

Little has been implemented so far.

`ModularForm(E)`

`Newform(E)`

The modular form associated to the elliptic curve E (which must be defined over the rationals).

`Eigenform(E, prec)`

`qEigenform(E, prec)`

The q-expansion of the newform associated to E, to the specified precision.

Note: this is exactly the same as calling `qExpansion(ModularForm(E),prec)`.

`EllipticCurve(f)`

An elliptic curve E with associated modular form f, when f is a weight 2 newform on $\Gamma_0(N)$ with rational Fourier coefficients.

The Cremona database is used to identify the isogeny class. (A routine to compute the curve from scratch is implemented, and can be called with `EllipticCurve(M : Database:=false)` where M is the relevant space of modular symbols; however this is not optimized for large level.)

Example H138E21

```
> M := ModularForms(Gamma0(389),2);
> f := Newform(M,1);
> Degree(f);
1
> E := EllipticCurve(f);
> E;
Elliptic Curve defined by y^2 + y = x^3 + x^2 - 2*x over Rational
Field
> Conductor(E);
389
> time s := PowerSeries(f,200);  // faster because it knows the elliptic curve
Time: 0.509
```

138.19 Modular Symbols

`ModularSymbols(M)`

The sequence of characteristic 0 spaces of modular symbols with given sign associated to the space of modular forms M, when this makes sense.

`ModularSymbols(M, sign)`

The sequence of characteristic 0 spaces of modular symbols with given sign associated to the space of modular forms M, when this makes sense.

Example H138E22

```
> M := ModularForms(Gamma0(389),2);
> ModularSymbols(M,+1);
[
    Full Modular symbols space of level 389, weight 2, and dimension
    33
]
> ModularSymbols(M,-1);
[
    Full Modular symbols space of level 389, weight 2, and dimension
    32
]
> M := ModularForms(Gamma1(13),2);
> ModularSymbols(M);
[
    Full Modular symbols space of level 13, weight 2, and dimension 1,
    Full Modular symbols space of level 13, weight 2, character $.1,
    and dimension 0,
    Full Modular symbols space of level 13, weight 2, character $.1,
    and dimension 4,
    Full Modular symbols space of level 13, weight 2, character $.1,
    and dimension 0,
    Full Modular symbols space of level 13, weight 2, character $.1^2,
    and dimension 2,
    Full Modular symbols space of level 13, weight 2, character $.1,
    and dimension 2
]
> Basis($1)[1];
-1/6*{-7/15, -6/13} + -1/6*{-7/18, -5/13} + -1/6*{-5/16, -4/13} +
-1/6*{-4/17, -3/13} + -1/6*{-3/19, -2/13} + -1/6*{0, oo}
```

138.20 Bibliography

[**Atk67**] A. O. L. Atkin. Weierstrass points at cusps $\Gamma_o(N)$. *Ann. of Math. (2)*, 85:42–45, 1967.

[**Cre97**] J. E. Cremona. *Algorithms for modular elliptic curves*. Cambridge University Press, Cambridge, second edition, 1997.

[**DI95**] Fred Diamond and Ju Im. Modular forms and modular curves. In *Seminar on Fermat's Last Theorem*, pages 39–133. Amer. Math. Soc., Providence, RI, 1995.

[**Lau11**] A. Lauder. Computations with classical and p-adic modular forms. *LMS J. Comput. Math.*, 14:214–231, 2011.

[**SS77**] J.-P. Serre and H. M. Stark. Modular forms of weight 1/2. In *Modular functions of one variable, VI (Proc. Second Internat. Conf., Univ. Bonn, Bonn, 1976)*, pages 27–67. Lecture Notes in Math., Vol. 627. Springer, Berlin, 1977.

[**Ste07**] William A. Stein. *Modular forms: a computational approach*, volume 79 of *Graduate Studies in Mathematics*. Amer. Math. Soc., Providence, Rhode Island, 2007.

139 MODULAR SYMBOLS

Chapter 139
MODULAR SYMBOLS

139.1 Introduction

This chapter, which was written by William Stein (wstein@gmail.com) with the help of feedback from Kevin Buzzard, describes how to compute with modular symbols using MAGMA. Modular symbols provide a presentation for certain homology groups, and as such they can be used to compute an eigenform basis for spaces of cusp forms $S_k(N, \varepsilon)$, where $k \geq 2$ is an integer and ε is an *arbitrary* Dirichlet character. Their generality makes modular symbols a natural tool in applications ranging from verification of modularity of Galois representations to elliptic curve computations.

Our implementation of modular symbols algorithms in MAGMA was deeply influenced by [Cre92, Cre97, Mer94]. The algorithms for computing arithmetic invariants of modular abelian varieties are based on [Ste00]. Those unfamiliar with modular symbols might wish to consult [Ste08] and the references contained therein and peruse [FM99].

139.1.1 Modular Symbols

The *modular group* $\mathrm{SL}_2(\mathbf{Z})$ is the group of 2×2 integer matrices with determinant 1. For each positive integer N let $\Gamma_0(N)$ denote the subgroup of $\mathrm{SL}_2(\mathbf{Z})$ of matrices that are upper triangular modulo N. As explained in the survey paper [DI95], there is an algebraic curve $X_0(N)$ over $\mathbf{Q}$ attached to $\Gamma_0(N)$. The Riemann surface attached to $X_0(N)$ is a compactified quotient of the upper half plane by the action of $\Gamma_0(N)$ via linear fractional transformations. Modular symbols provide an explicit computable presentation for certain "(co-)homology groups" attached to modular curves $X_0(N)$.

Let $\mathbf{P^1}(\mathbf{Q})$ denote the set $\mathbf{Q} \cup \{\infty\}$, and fix a field F. Let $\mathbf{M}$ denote the F-vector space generated by the formal symbols $\{a, b\}$, with $a, b \in \mathbf{P^1}(\mathbf{Q})$, modulo the relations $\{a, b\} + \{b, c\} + \{c, a\} = 0$ for all $a, b, c \in \mathbf{Q}$. (The symbol $\{a, b\}$ can be visualized as the homology class of a geodesic path from a to b in the upper half plane.) Fix a positive integer k. A weight-k symbol is a formal product $X^i Y^{k-2-i}\{a, b\}$, where $X^i Y^{k-2-i} \in F[X, Y]$. Denote by $\mathbf{M_k}$ the formal F-vector space with basis the set of all weight-k modular symbols (thus $\mathbf{M_k} \approx \mathbf{M} \otimes \mathrm{Sym}^{\mathbf{k-2}}(\mathbf{F} \times \mathbf{F})$). The group $\mathrm{GL}_2(\mathbf{Q})$ acts on the left on $\mathbf{M_k}$; the matrix $g = \begin{pmatrix} u & v \\ w & z \end{pmatrix}$ in $\mathrm{GL}_2(\mathbf{Q})$ acts by

$$g(X^i Y^{k-2-i}\{a, b\}) = (zX - vY)^i(-wX + uY)^{k-2-i} \left\{ \frac{ua + v}{wa + z}, \frac{ub + v}{wb + z} \right\}.$$

A *mod N Dirichlet character* ε is a homomorphism

$$\varepsilon : (\mathbf{Z}/N\mathbf{Z})^* \to F^*.$$

The vector space $\mathbf{M_k}(\mathbf{N}, \varepsilon; \mathbf{F})$ of *modular symbols of weight k, level N and character ε over F* is the quotient of $\mathbf{M_k}$ by the subspace generated by all $x - \varepsilon(u)g(x)$, for x in $\mathbf{M_k}$ and $g = \begin{pmatrix} u & v \\ w & z \end{pmatrix} \in \Gamma_0(N)$. We denote the equivalence class that defines a modular symbol by giving a representative element.

The space of modular symbols is a finite-dimensional vector space, and there is a natural finite presentation for it in terms of Manin symbols.

139.2 Basics

139.2.1 Verbose Output

The verbosity level for modular symbols computations can be set using the command `SetVerbose("ModularSymbols",n)`, where `n` is 0 (silent), 1 (verbose), or 2 (very verbose). (The verbose flag for modular symbols was called `ModularForms` in MAGMA version 2.7.)

139.2.2 Categories

Spaces of modular symbols belong to the category `ModSym`. The category `SetCsp` has exactly one object `Cusps()`, which is the set $\mathbf{P^1}(\mathbf{Q}) = \mathbf{Q} \cup \{\infty\}$ introduced above. The element ∞ of $\mathbf{P^1}(\mathbf{Q})$ is entered using the expression `Cusps()!Infinity()`.

Example H139E1

We compute a basis for the space of modular symbols of weight 2, level 11 and trivial character.

```
> M := ModularSymbols(11,2); M;
Full modular symbols space for Gamma_0(11) of weight 2 and dimension 3
over Rational Field
> Type(M);
ModSym
> Basis(M);
[
    {-1/7, 0},
    {-1/5, 0},
    {oo, 0}
]
> M!<1,[1/5,1]>;
{-1/5, 0}
> // the modular symbols {1/5,1} and {-1/5,0} are equal.
> Type(M!<1,[1/5,1]>);
ModSymElt
```

Using `SetVerbose`, we can see how the computation progresses.

```
> SetVerbose("ModularSymbols",2);
> M := ModularSymbols(11,2);
Computing space of modular symbols of level 11 and weight 2....
I.      Manin symbols list.
```

```
                (0 s)
II.     2-term relations.
                (0.019 s)
III.    3-term relations.
         Computing quotient by 4 relations.
                (0.009 s)
                (total time to create space = 0.029 s)
> SetVerbose("ModularSymbols",0);
```

Modular symbols can be input using `Cusps()`.

```
> M := ModularSymbols(11,2);
> P := Cusps(); P;
Set of all cusps
> Type(P);
SetCsp
> oo := P!Infinity();
> M!<1,[oo,P!0]>;       // note that 0 must be coerced into P.
{oo, 0}
> M!<1,[1/5,1]> + M!<1,[oo,P!0]>;
{-1/5, 0} + {oo, 0}
```

Modular symbols are also defined over finite fields.

```
> M := ModularSymbols(11,2,GF(7)); M;
Full modular symbols space for Gamma_0(11) of weight 2 and dimension 3
over Finite field of size 7
> BaseField(M);
Finite field of size 7
> 7*M!<1,[1/5,1]>;
0
```

139.3 Creation Functions

139.3.1 Ambient Spaces

An ambient space of modular symbols is created by specifying a base ring, character, weight, and optional sign. Note that spaces of modular symbols may be defined directly over any base ring (unlike `ModularForms`, which can only be base extensions of spaces over the integers). The most general signature is `ModularSymbols(eps, k, sign)`, where `eps` is a Dirichlet character (the level is taken to be the modulus of `eps`, and the base ring is taken to be the base ring of `eps`).

For information on Dirichlet characters, see Section 19.8.

Warning: Certain functions, such as `DualVectorSpace`, may fail when given as input a space of modular symbols over a field of positive characteristic, because the Hecke operators T_p, with p prime to the level, need not be semisimple.

`ModularSymbols(N)`

The space of modular symbols of level N, weight 2, and trivial character over the rational numbers.

`ModularSymbols(N, k)`

The space of modular symbols of level N, weight k, and trivial character over the rational numbers.

`ModularSymbols(N, k, F)`

The space of modular symbols of level N, weight k, and trivial character over the field F.

`ModularSymbols(N, k, sign)`

The space of modular symbols of level N, weight k, trivial character, and given *sign* (as an integer) over the rational numbers.

`ModularSymbols(N, k, F, sign)`

The space of modular symbols of level N, weight k, trivial character, and given *sign* (as an integer), over the field F.

`ModularSymbols(eps, k)`

The space of modular symbols of weight k and character ε (as an element of a Dirichlet group). Note that ε determines the level and the base field, so they do not need to be specified.

`ModularSymbols(eps, k, sign)`

The space of modular symbols of weight k and character ε (as an element of a Dirichlet group). The level and base field are specified as part of ε. The third argument "sign" allows for working in certain quotients. The possible values are -1, 0, and $+1$, which correspond to the -1 quotient, full space, and $+1$ quotient, respectively. The $+1$ quotient of M is $M/(*-1)M$, where $*$ is `StarInvolution(M)`.

Example H139E2

We create spaces of modular symbols in several different ways.

```
> M37 := ModularSymbols(37); M37;
Full modular symbols space for Gamma_0(37) of weight 2 and dimension 5
over Rational Field
> Basis(M37);
[
    {-1/29, 0},
    {-1/22, 0},
    {-1/12, 0},
    {-1/18, 0},
    {oo, 0}
```

```
]
```

As M37 is a space of modular symbols, it is not incorrect that its dimension is different than that of the three-dimensional space of modular forms $M_2(\Gamma_0(37))$. We have

$$\dim \mathbf{M_2}(\mathbf{\Gamma_0}(\mathbf{37})) = \mathbf{2} \times (\text{dim cusp forms}) + \mathbf{1} \times (\text{dim Eisenstein series}) = \mathbf{5}.$$

```
> MF := ModularForms(Gamma0(37),2);
> 2*Dimension(CuspidalSubspace(MF)) + Dimension(EisensteinSubspace(MF));
5
```

Next we decompose M37 with respect to the Hecke operators T_2, T_3, and T_5.

```
> D := Decomposition(M37,5); D;
[
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 1
     over Rational Field,
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field
]
```

The first factor corresponds to the standard Eisenstein series, and the second corresponds to an elliptic curve:

```
> E := EllipticCurve(D[2]); E;
Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50 over
Rational Field
> Rank(E);
0
```

We now create the space $\mathbf{M_{12}}(\mathbf{1})$ of weight 12 modular symbols of level 1.

```
> M12 := ModularSymbols(1,12); M12;
Full modular symbols space for Gamma_0(1) of weight 12 and dimension 3
over Rational Field
> Basis(M12);
[
    X^10*{0, oo},
    X^8*Y^2*{0, oo},
    X^9*Y*{0, oo}
]

> DimensionCuspFormsGamma0(1,12);
1
> R<z>:=PowerSeriesRing(Rationals());
> Delta(z)+ O(z^7);
z - 24*z^2 + 252*z^3 - 1472*z^4 + 4830*z^5 - 6048*z^6 + O(z^7)
```

As a module, cuspidal modular symbols equal cuspforms with multiplicity two.

```
> M12 := ModularSymbols(1,12);
```

```
> HeckeOperator(CuspidalSubspace(M12),2);
[-24   0]
[  0 -24]
>  qExpansionBasis(CuspidalSubspace(M12),7);
[
    q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 + O(q^7)
]
```

For efficiency purposes, since one is often interested only in q-expansion, it is possible to work in the quotient of the space of modular symbols by all relations $*x = x$ (or $*x = -x$), where $*$ is `StarInvolution(M)`. In either of these quotients (except possibly in characteristic $p > 0$) the cusp forms appear with multiplicity one instead of two.

```
> M12plus := ModularSymbols(1,12,+1);
> Basis(M12plus);
[
    X^10*{0, oo},
    X^8*Y^2*{0, oo}
]
> CuspidalSubspace(M12plus);
Modular symbols space for Gamma_0(1) of weight 12 and dimension 1 over
Rational Field
> qExpansionBasis(CuspidalSubspace(M12),7);
[
    q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 + O(q^7)
]
```

The following is an example of how to create Dirichlet characters in MAGMA, and how to create a space of modular symbols with nontrivial character. For more details, see Section 19.8.

```
> G<a,b,c> := DirichletGroup(16*7,CyclotomicField(EulerPhi(16*7)));
> Order(a);
2
> Conductor(a);
4
> Order(b);
4
> Conductor(b);
16
> Order(c);
6
> Conductor(c);
7
> eps := a*b*c;
> M := ModularSymbols(eps,2); M;
Full modular symbols space of level 112, weight 2, character a*b*c,
and dimension 32 over Cyclotomic Field of order 48 and degree 16
> BaseField(M);
Cyclotomic Field of order 48 and degree 16
```

139.3.2 Labels

It is also possible to create many spaces of modular symbols for $\Gamma_0(N)$ by passing a "descriptive label" as an argument to `ModularSymbols`. The most specific label is a string of the form `[Level]k[Weight][IsogenyClass]`, where `[Level]` is the level, `[Weight]` is the weight, `[IsogenyClass]` is a letter code: `A`, `B`, ..., `Z`, `AA`, `BB`, ..., `ZZ`, `AAA`, ..., and `k` is a place holder to separate the level and the weight. If the label is `[Level][IsogenyClass]`, then the weight k is assumed equal to 2. If the label is `[Level]k[Weight]` then the cuspidal subspace of the full ambient space of modular symbols of weight k and level N is returned. The following are valid labels: `11A`, `37B`, `3k12A`, `11k4`. The ordering used on isogenies classes is `lt`; see the documentation for `SortDecomposition`.

Note: There is currently no intrinsic that, given a space of modular symbols, returns its label.

Warning: For 146 of the levels between 56 and 450, our ordering of weight 2 rational newforms disagrees with the ordering used in [Cre97]. Fortunately, it is easy to create a space of modular symbols from one of Cremona's labels using the associated elliptic curve.

```
> E  := EllipticCurve(CremonaDatabase(),"56A");
> M1 := ModularSymbols(E);
```

Observe that Cremona's `"56A"` is different from ours.

```
> M2 := ModularSymbols("56A");
> M2 eq M1;
false
```

`ModularSymbols(s, sign)`

`ModularSymbols(s)`

The space of modular symbols described by a label, the string s, and given *sign*.

Example H139E3

The cusp form $\Delta(q)$ is related to the space of modular symbols whose label is `"1k12A"`.

```
> Del := ModularSymbols("1k12A"); Del;
Modular symbols space for Gamma_0(1) of weight 12 and dimension 2 over
Rational Field
> qEigenform(Del,5);
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
```

Next, we create the space corresponding to the first newform on $\Gamma_0(11)$ of weight 4.

```
> M := ModularSymbols("11k4A"); M;
Modular symbols space for Gamma_0(11) of weight 4 and dimension 4 over
Rational Field
> AmbientSpace(M);
Full modular symbols space for Gamma_0(11) of weight 4 and dimension 6
over Rational Field
```

```
> qEigenform(M,5);
q + a*q^2 + (-4*a + 3)*q^3 + (2*a - 6)*q^4 + O(q^5)
> Parent($1);
Power series ring in q over Univariate Quotient Polynomial Algebra in
a over Rational Field with modulus a^2 - 2*a - 2
```

We next create the $+1$ quotient of the cuspidal subspace of weight-4 modular symbols of level 37.

```
> M := ModularSymbols("37k4",+1); M;
Modular symbols space for Gamma_0(37) of weight 4 and dimension 9 over
Rational Field
> AmbientSpace(M);
Full modular symbols space for Gamma_0(37) of weight 4 and dimension
11 over Rational Field
> Factorization(CharacteristicPolynomial(HeckeOperator(M,2)));
[
    <x^4 + 6*x^3 - x^2 - 16*x + 6, 1>,
    <x^5 - 4*x^4 - 21*x^3 + 74*x^2 + 102*x - 296, 1>
]
```

139.3.3 Creation of Elements

Suppose M is a space of weight k modular symbols over a field F. A modular symbol $P(X,Y)\{\alpha,\beta\}$ is input as `M!<P(X,Y),[alpha,beta]>`, where $P(X,Y) \in F[X,Y]$ is homogeneous of degree $k-2$, and $\alpha, \beta \in \mathbf{P^1}(\mathbf{Q})$. Here is an example:

Example H139E4

First create the space $M = \mathbf{M_4}(\mathbf{\Gamma_0}(\mathbf{3}); \mathbf{F_7})$.

```
> F7 := GF(7);
> M := ModularSymbols(3,4,F7);
> R<X,Y> := PolynomialRing(F7,2);
```

Now we input $(X^2 - 2XY)\{0,1\}$.

```
> M!<X^2-2*X*Y,[Cusps()|0,1]>;
6*Y^2*{oo, 0}
```

Note that $(X^2 - 2XY)\{0,1\} = 6Y^2\{\infty, 0\}$ in M.
When $k = 2$, simply enter `M!<1,[alpha,beta]>`.

```
> M := ModularSymbols(11,2);
> M!<1,[Cusps()|0,Infinity()]>;
-1*{oo, 0}
> M![<1,[Cusps()|0,Infinity()]>, <1,[Cusps()|0,1/11]>];
-2*{oo, 0}
```

Any space M of modular symbols is finitely generated. One proof of this uses that every modular symbol is a linear combination of *Manin symbols*. Let $\mathbf{P^1}(\mathbf{Z}/N\mathbf{Z})$ be the set of pairs $(\overline{u},\overline{v}) \in \mathbf{Z}/N\mathbf{Z} \times \mathbf{Z}N\mathbf{Z}$ such that $\mathrm{GCD}(u,v,N) = 1$, where u and v are lifts of $\overline{u}$ and $\overline{v}$ to $\mathbf{Z}$. A Manin symbols $\langle P(X,Y),(\overline{u},\overline{v})\rangle$ is a pair consisting of a homogeneous polynomial $P(X,Y) \in F[X,Y]$ of degree $k-2$ and an element $(\overline{u},\overline{v}) \in \mathbf{P^1}(\mathbf{Z}/N\mathbf{Z})$. The modular symbol associated to $\langle P(X,Y),(\overline{u},\overline{v})\rangle$ is constructed as follows. Choose lifts u,v of $\overline{u},\overline{v}$ such that $\mathrm{GCD}(u,v) = 1$. Then there is a matrix $g = \begin{pmatrix} w & z \\ u & v \end{pmatrix}$ in $\mathrm{SL}_2(\mathbf{Z})$ whose lower two entries are u and v. The modular symbol is then $g(P(X,Y)\{0,\infty\})$. The intrinsic `ConvertFromManinSymbol` computes the modular symbol attached to a Manin symbol. Ever modular symbol can be written as a linear combination of Manin symbols using the intrinsic `ManinSymbol`.

Example H139E5

In this example, we convert between Manin and modular symbols representations of a few elements of a space of modular symbols of weight 4 over $\mathbf{F}_5$.

```
> F5 := GF(5);
> M := ModularSymbols(6,4,F5);
> R<X,Y> := PolynomialRing(F5,2);
> ConvertFromManinSymbol(M,<X^2+Y^2,[1,4]>);
(3*X^2 + 3*X*Y + 2*Y^2)*{-1/2, 0} + (X^2 + 4*X*Y + 4*Y^2)*{-1/3, 0} +
(X^2 + X*Y + 4*Y^2)*{1/3, 1/2}
> ManinSymbol(M.1-3*M.2);
[
    <X^2, (0 1)>,
    <2*X^2, (1 2)>
]
```

Thus the element `M.1-3*M.2` of M corresponds to the sum of Manin symbols $\langle X^2,(0,1)\rangle + 2\langle X^2,(1,2)\rangle$.

`M ! x`

The coercion of x into the space of modular symbols M. Here x can be either a modular symbol that lies in a subspace of M, a 2-tuple that describes a modular symbol, a sequence of such 2-tuples, or anything that can be coerced into `VectorSpace(M)`. If x is a valid sequence of such 2-tuples, then `M!x` is the sum of the coercions into M of the elements of the sequence x.

`ConvertFromManinSymbol(M, x)`

The modular symbol associated to the 2-tuple $x = \langle P(X,Y),[u,v]\rangle$, where $P(X,Y) \in F[X,Y]$ is homogeneous of degree $k-1$, F is the base field of the space of modular symbols M, and $[u,v]$ is a sequence of 2 integers that defines an element of $\mathbf{P^1}(\mathbf{Z}/N\mathbf{Z})$, where N is the level of M.

`ManinSymbol(x)`

An expression for the modular symbol x in terms of Manin symbols, which are represented as 2-tuples $\langle P(X,Y), [u,v] \rangle$.

Example H139E6

```
> M := ModularSymbols(14,2); M;
Full modular symbols space for Gamma_0(14) of weight 2 and dimension 5
over Rational Field
> Basis(M);
[
    {oo, 0},
    {-1/8, 0},
    {-1/10, 0},
    {-1/12, 0},
    {-1/2, -3/7}
]
> M!<1,[1,0]>;
0
> M!<1,[0,1/11]>;
{-1/10, 0} + -1*{-1/12, 0}
> M![<1,[0,1/2]>, <-1,[0,1/7]>];              // sequences are added
{-1/8, 0} + -1*{-1/12, 0} + -1*{-1/2, -3/7}
> M!<1,[0,1/2]> - M!<1,[0,1/7]>;
{-1/8, 0} + -1*{-1/12, 0} + -1*{-1/2, -3/7}
> M!<1,[Cusps()|Infinity(),0]>;               // Infinity() is in Cusps().
{oo, 0}
```

We can also coerce sequences into the underlying vector space of M.

```
> VectorSpace(M);
Full Vector space of degree 5 over Rational Field
Mapping from: Full Vector space of degree 5 over Rational Field to
ModSym: M given by a rule [no inverse]
Mapping from: ModSym: M to Full Vector space of degree 5 over Rational
Field given by a rule [no inverse]
> Eltseq(M.3);
[ 0, 0, 1, 0, 0 ]
> M![ 0, 0, 1, 0, 0 ];
{-1/10, 0}
> M.3;
{-1/10, 0}
```

The "polynomial coefficients" of the modular symbols are homogeneous polynomials in 2 variables of degree $k-2$.

```
> M := ModularSymbols(1,12);
> Basis(M);
[
```

```
    X^10*{0, oo},
    X^8*Y^2*{0, oo},
    X^9*Y*{0, oo}
]
> R<X,Y> := PolynomialRing(Rationals(),2);
> M!<X^9*Y,[Cusps()|0,Infinity()]>;
X^9*Y*{0, oo}
> M!<X^7*Y^3,[Cusps()|0,Infinity()]>;
-25/48*X^9*Y*{0, oo}
> Eltseq(M!<X*Y^9,[1/3,1/2]>);
[ -19171, -58050, -30970 ]
> M![1,2,3];
X^10*{0, oo} + 2*X^8*Y^2*{0, oo} + 3*X^9*Y*{0, oo}
> ManinSymbol(M![1,2,3]);
[
    <X^10, (0 1)>,
    <2*X^8*Y^2, (0 1)>,
    <3*X^9*Y, (0 1)>
]
```

139.4 Bases

`Basis(M)`

Basis for the space of modular symbols M.

`IntegralBasis(M)`

First suppose that the space of modular symbols M equals `AmbientSpace(M)`. Then this intrinsic returns a basis $x_1, \ldots, x_n$ for M such that $\mathbf{Z}x_1 + \cdots + \mathbf{Z}x_n$ is the $\mathbf{Z}$-submodule of M generated by all modular symbols $X^i \cdot Y^{k-2-i}\{\alpha, \beta\}$ with $i = 0, \ldots, k-2$ and $\alpha, \beta \in \mathbf{P^1}(\mathbf{Q})$. If M is not `AmbientSpace(M)`, then this intrinsic returns a $\mathbf{Z}$-basis for $M \cap (\mathbf{Z}x_1 + \cdots + \mathbf{Z}x_n)$, where $x_1, \ldots, x_n$ is an integral basis for `AmbientSpace(M)`. The base field of M must be $\mathbf{Q}$.

Example H139E7

```
> M := ModularSymbols(1,12);
> Basis(M);
[
    X^10*{0, oo},
    X^8*Y^2*{0, oo},
    X^9*Y*{0, oo}
]
> IntegralBasis(M);
[
```

```
    1/48*X^9*Y*{0, oo},
    1/14*X^8*Y^2*{0, oo},
    X^10*{0, oo}
]
```

`IntegralBasis(M)` is a basis for the **Z**-module spanned by the following symbols:

```
> R<X,Y> := PolynomialRing(Rationals(),2);
> [M!<X^i*Y^(10-i),[Cusps()|0,Infinity()]> : i in [0..10]];
[
    -X^10*{0, oo},
    X^9*Y*{0, oo},
    -X^8*Y^2*{0, oo},
    -25/48*X^9*Y*{0, oo},
    9/14*X^8*Y^2*{0, oo},
    5/12*X^9*Y*{0, oo},
    -9/14*X^8*Y^2*{0, oo},
    -25/48*X^9*Y*{0, oo},
    X^8*Y^2*{0, oo},
    X^9*Y*{0, oo},
    X^10*{0, oo}
]
```

We can also compute an integral basis of a subspace.

```
> C := CuspidalSubspace(M);
> IntegralBasis(C);
[
    1/48*X^9*Y*{0, oo},
    1/14*X^8*Y^2*{0, oo}
]
```

In Remark 3 on page 69 of [Mer94], Merel says "it would be interesting to find a basis in terms of Manin symbols" for the **Z**-module of Eisenstein symbols (see Section 139.8 for the definition of `EisensteinSubspace`). Here are the first few examples in the case of level 1:

```
> M := ModularSymbols(1,12);
> E := EisensteinSubspace(M);
> IntegralBasis(E);
[
    691*X^10*{0, oo} + 1620*X^8*Y^2*{0, oo}
]
> ManinSymbol(IntegralBasis(E)[1]);
[
    <691*X^10, (0 1)>,
    <1620*X^8*Y^2, (0 1)>
]
```

To more easily compute several examples, we define a function:

```
> function EisZ(k)
```

```
>    E := EisensteinSubspace(ModularSymbols(1,k));
>    B := IntegralBasis(E);
>    return [ManinSymbol(z) : z in B];
> end function;
> EisZ(12);
[
    [
        <691*X^10, (0 1)>,
        <1620*X^8*Y^2, (0 1)>
    ]
]
> EisZ(16);
[
    [
        <16380*X^12*Y^2, (0 1)>,
        <3617*X^14, (0 1)>
    ]
]
> EisZ(18);
[
    [
        <43867*X^16, (0 1)>,
        <270000*X^14*Y^2, (0 1)>
    ]
]
> EisZ(20);
[
    [
        <174611*X^18, (0 1)>,
        <1349460*X^16*Y^2, (0 1)>
    ]
]
> EisZ(22);
[
    [
        <748125*X^18*Y^2, (0 1)>,
        <77683*X^20, (0 1)>
    ]
]
```

Send me an email if you determine the basis in general. In each example above the coefficient of X^{k-2} is, up to sign, `Numerator(Bernoulli(k)/k)`.

139.5 Associated Vector Space

The functions `VectorSpace`, `DualVectorSpace`, and `Lattice` return the underlying vector space, dual vector space, and lattice associated to a space of modular symbols. A space of modular symbols is represented internally as a subspace of a vector space, and a subspace of the linear dual of the vector space. To carry along the subspace of the linear dual is useful in many computations; one example is efficient computation of Hecke operators. When the base field is **Q**, the lattice comes from the natural integral structure on modular symbols.

`VectorSpace(M)`

The vector space V underlying the space of modular symbols M, the map $V \to M$, and the map $M \to V$.

`DualVectorSpace(M)`

The subspace of the linear dual of `VectorSpace(AmbientSpace(M))` that is isomorphic to the space of modular symbols M as a module over the Hecke algebra.

`Lattice(M)`

The lattice generated by the integral modular symbols in the vector space representation of the space of modular symbols M. This is the lattice generated by all modular symbols $X^iY^{k-2-i}\{a, b\}$. The base field of M must be `RationalField()`.

Example H139E8

```
> M := ModularSymbols(DirichletGroup(11).1,3); M;
Full modular symbols space of level 11, weight 3, character $.1, and
dimension 4 over Rational Field
> VectorSpace(M);
Full Vector space of degree 4 over Rational Field
Mapping from: Full Vector space of degree 4 over Rational Field to
ModSym: M given by a rule [no inverse]
Mapping from: ModSym: M to Full Vector space of degree 4 over Rational
Field given by a rule [no inverse]
> Basis(VectorSpace(CuspidalSubspace(M)));
[
    ( 0  1  0 -1),
    ( 0  0  1 -1)
]
> Basis(VectorSpace(EisensteinSubspace(M)));
[
    (   1    0 -2/3 -1/3),
    ( 0  1 -5 -2)
]
> Lattice(CuspidalSubspace(M));
Lattice of rank 2 and degree 4
Basis:
```

```
( 0  1 -1  0)
( 0  1  1 -2)
Basis Denominator: 2
Mapping from: Lattice of rank 2 and degree 4 to Modular symbols space
of level 11, weight 3, character $.1, and dimension 2 over Rational
Field given by a rule [no inverse]
> Basis(Lattice(EisensteinSubspace(M)));
[
    (   0  1/2 -5/2   -1),
    (   3 -1/2  1/2    0)
]
```

139.6 Degeneracy Maps

Consider an ambient space M_1 of modular symbols of level N_1, and suppose M_2 is an ambient space of modular symbols of level a multiple N_2 of N_1 whose weight equals the weight of M_1 and whose character is induced by the character of M_1. Then for each divisor d of N_2/N_1 there are natural maps $\alpha_d : M_1 \to M_2$ and $\beta_d : M_2 \to M_1$ such that $\beta_d \circ \alpha_d$ is multiplication by $d^{k-2} \cdot [\Gamma_0(N_1) : \Gamma_0(N_2)]$, where k is the common weight of M_1 and M_2. On cuspidal parts, the map β_d is dual to the map $f(q) \to f(q^d)$ on modular forms. Use the function `DegeneracyMap` to compute the maps α_d and β_d.

Given a space M of modular symbols and a positive integer N that is a multiple of the level of M, the images of M under the degeneracy maps generate a modular symbols space of level N. The constructor `ModularSymbols(M,N)` computes this space.

Let M be a space of modular symbols of level N, and let N' be a multiple of N. The subspace

$$\sum_{d \mid \frac{N'}{N}} \alpha_d(M) \subset \mathbf{M_k}(\mathbf{N}', \varepsilon)$$

is stable under the Hecke operators. Here is how to create this subspace using MAGMA:

```
> M := ModularSymbols(11,2); M;
Full modular symbols space for Gamma_0(11) of weight 2 and dimension 3
over Rational Field
> M33 := ModularSymbols(M,33); M33;
Modular symbols space for Gamma_0(33) of weight 2 and dimension 6 over
Rational Field
```

`DegeneracyMap(M1, M2, d)`

The degeneracy map $M_1 \to M_2$ of spaces of modular symbols associated to d. Let N_i be the level of M_i for $i = 1, 2$. Suppose that d is a divisor of either the numerator or denominator of the rational number N_1/N_2, written in reduced form. If $N_1 \mid N_2$, then this intrinsic returns $\alpha_d : M_1 \to M_2$, or if $N_2 \mid N_1$, then this intrinsic returns $\beta_d : M_1 \to M_2$. It is an error if neither divisibility holds.

`DegeneracyMatrix(M1, M2, d)`

Given spaces of modular symbols $M1$ and $M2$ and an integer d, return the matrix of `DegeneracyMap(M1,M2,d)` with respect to `Basis(M1)` and `Basis(M2)`. Both `IsAmbient(M1)` and `IsAmbient(M2)` must be true.

`ModularSymbols(M, N')`

The modular symbols space of level N' associated to M. Let N be the level of M. If $N \mid N'$, then this intrinsic returns the modular symbols space

$$\sum_{d \mid \frac{N'}{N}} \alpha_d(M).$$

If $N' \mid N$, then this intrinsic returns the modular symbols space

$$\sum_{d \mid \frac{N}{N'}} \beta_d(M).$$

In this latter case, if `Conductor(DirichletCharacter(M))` does not divide N', then the 0 space is returned.

`M1 !! M2`

The modular symbols subspace of M_1 associated to M_2. Let N_1 be the level of M_1. If `ModularSymbols(M2,N1)` is defined, let M_3 be this modular symbols space, otherwise terminate with an error. If M_3 is contained in M_1, return M_3, otherwise terminate with an error.

Example H139E9

We compute degeneracy maps α_2 and β_2.

```
> M15 := ModularSymbols(15);
> M30 := ModularSymbols(30);
> alp_2 := DegeneracyMap(M15,M30,2);
> alp_2(M15.1);
2*{oo, 0} + -1*{-1/28, 0} + -1*{-1/2, -7/15}
> beta_2 := DegeneracyMap(M30,M15,2);
> beta_2(alp_2(M15.1));
3*{oo, 0}
> M15.1;
{oo, 0}
```

We can consider the space generated by the image of a space of modular symbols of level 11 in spaces of higher level.

```
> X11 := ModularSymbols("11k2A");
> qEigenform(X11,6);
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
```

```
> ModularSymbols(X11,33);
Modular symbols space for Gamma_0(33) of weight 2 and dimension 4 over
Rational Field
> X33 := ModularSymbols(X11,33);
> qExpansionBasis(X33,6);
[
    q - 2*q^2 + 2*q^4 + q^5 + O(q^6),
    q^3 + O(q^6)
]
> Factorization(CharacteristicPolynomial(HeckeOperator(X33,3)));
[
    <x^2 + x + 3, 2>
]
> ModularDegree(X33);
3
```

We can also construct the space generated by the images of `X11` at higher level using the `!!` operator.

```
> M44 := ModularSymbols(44,2);
> A := M44!!X11; A;
Modular symbols space for Gamma_0(44) of weight 2 and dimension 6 over
Rational Field
> X11!!A;       // back to the original space
Modular symbols space for Gamma_0(11) of weight 2 and dimension 2 over
Rational Field
```

139.7 Decomposition

The functions `Decomposition` and `NewformDecomposition` express a space of modular symbols as a direct sum of Hecke-stable subspaces.

In the intrinsics below, the `Proof` parameter affects the internal characteristic polynomial computations. If `Proof` is set to `false` and this causes a characteristic polynomial computation to fail, then the sum of the dimensions of the spaces returned by `Decomposition` will be less than the dimension of `M`. Thus setting `Proof` equal to `false` is usually safe.

`Decomposition(M, bound : parameters)`

`Proof` BoolElt *Default :* `true`

The decomposition of the space of modular symbols M with respect to the Hecke operators T_p with p coprime to the level of M and $p \leq$ `bound`. If `bound` is too small, the constituents of the decomposition are not guaranteed to be "irreducible", in the sense that they can not be decomposed further into kernels and images of Hecke operators T_p with p prime to the level of M. When `Decomposition` is called, the result is cached, so each successive call results in a possibly more refined decomposition.

Important Note: In some cases `NewformDecomposition` is significantly faster than `Decomposition`.

`NewformDecomposition(M : parameters)`

`Proof` BoolElt *Default :* `true`

Unsorted decomposition of the space of modular symbols M into factors corresponding to the Galois conjugacy classes of newforms of level some divisor of the level of M. We require that `IsCuspidal(M)` is `true`.

`AssociatedNewSpace(M)`

The space of modular symbols corresponding to the Galois-conjugacy class of newforms associated to the space of modular symbols M. The level of the newforms is allowed to be a proper divisor of the level of M. The space M must have been created using `NewformDecomposition`.

`SortDecomposition(D)`

Sort the sequence D of spaces of modular symbols with respect to the `lt` comparison operator.

`IsIrreducible(M)`

Returns `true` if and only if `Decomposition(M)` has cardinality 1.

`M1 lt M2`

The ordering on spaces of modular symbols is determined as follows:

(1) This rule applies only if `NewformDecomposition` was used to construct both of M_1 and M_2: If `Level(AssociatedNewSpace(M1))` is not equal to that of $M2$ then the M_i with larger associated level is first.

(2) The smaller dimension is first.

(3) The following applies when the weight is 2 and the character is trivial: Order by W_q eigenvalues, starting with the *smallest* $p \mid N$, with the eigenvalue $+1$ being less than the eigenvalue -1.

(4) Order by abs(trace(a_p)), with p not dividing the level, and with positive trace being smaller in the event that the two absolute values are equal.

Rule (3) is included so that our ordering extends the one used in (most of!) [Cre97].

Example H139E10

First, we compute the decomposition of the space of modular symbols of weight 2 and level 37.

```
> M := ModularSymbols(37,2); M;
Full modular symbols space for Gamma_0(37) of weight 2 and dimension 5
over Rational Field
> D := Decomposition(M,2); D;
[
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 1
     over Rational Field,
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field
]
> IsIrreducible(D[2]);
true
> C := CuspidalSubspace(M); C;
Modular symbols space for Gamma_0(37) of weight 2 and dimension 4 over
Rational Field
> N := NewformDecomposition(C); N;
[
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(37) of weight 2 and dimension 2
     over Rational Field
]
```

Next, we use `NewformDecomposition` to decompose a space having plentiful old subspaces.

```
> M := ModularSymbols(90,2); M;
Full modular symbols space for Gamma_0(90) of weight 2 and dimension
37 over Rational Field
> D := Decomposition(M,11); D;
[
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 11
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 4
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 4
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 8
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 6
     over Rational Field
```

```
]
> C := CuspidalSubspace(M); C;
Modular symbols space for Gamma_0(90) of weight 2 and dimension 22
over Rational Field
> N := NewformDecomposition(C); N;
[
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 4
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 4
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 8
     over Rational Field
]
```

The above decomposition uses all of the Hecke operator; it suggests that the decomposition `D` is not as fine as possible. Indeed, `D[7]` breaks up further:

```
> Decomposition(D[7],11);
[
    Modular symbols space for Gamma_0(90) of weight 2 and dimension 6
    over Rational Field
]
> Decomposition(D[7],19);
[
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 4
     over Rational Field,
     Modular symbols space for Gamma_0(90) of weight 2 and dimension 2
     over Rational Field
]
```

The function `AssociatedNewSpace` allows us to see where each of these subspace comes from. By definition they each arise by taking images under the degeneracy maps from a single Galois-conjugacy class of newforms of *some* level dividing 90.

```
> [Level(AssociatedNewSpace(A)) : A in N];
[ 90, 90, 90, 45, 30, 15 ]
> A := N[4];
> qEigenform(AssociatedNewSpace(A),7);
q + q^2 - q^4 - q^5 + O(q^7)
> qExpansionBasis(A,7);
[
    q - 2*q^4 - q^5 + O(q^7),
    q^2 + q^4 + O(q^7)
```

```
]
```

139.8 Subspaces

The following functions compute the cuspidal, Eisenstein, and new subspaces, along with the complement of a subspace.

`CuspidalSubspace(M)`

The cuspidal subspace of the space of modular symbols M. This is the kernel of `BoundaryMap(M)`.

`IsCuspidal(M)`

Returns `true` if and only if the space of modular symbols M is contained in the cuspidal subspace of the ambient space.

`EisensteinSubspace(M)`

The Eisenstein subspace of the space of modular symbols M. This is the complement in M of the cuspidal subspace of M.

`IsEisenstein(M)`

Returns `true` if and only if the space of modular symbols M is contained in the Eisenstein subspace of the ambient space.

`NewSubspace(M)`

The new subspace of the space of modular symbols M. This is the intersection of `NewSubspace(M,p)` as p varies over all prime divisors of the level of M. Note that M is required to be cuspidal.

`IsNew(M)`

Returns `true` if and only if the space of modular symbols M is contained in the new cuspidal subspace of the ambient space.

`NewSubspace(M, p)`

The p-new subspace of the space of modular symbols M. This is the kernel of the degeneracy map from M to the space of modular symbols of level equal to the level of M divided by p and character the restriction of the character of M. If the character of M does not restrict, then `NewSubspace(M,p)` is equal to M. Note that M is required to be cuspidal.

`Kernel(I, M)`

The kernel of I on the space of modular symbols M. Let T_p denote the pth Hecke operator (see Section 142.14). This is the subspace of M obtained by intersecting the kernels of the operators $f_n(T_{p_n})$, where I is a sequence $[\langle p_1, f_1(x)\rangle, ..., \langle p_n, f_n(x)\rangle]$ of pairs consisting of a prime number and a polynomial. Only primes p_i which do not divide the level of M are used.

`Complement(M)`

The space of modular symbols complementary to the space of modular symbols M in the ambient space of M. Thus the ambient space of M is equal to the direct sum of M and `Complement(M)`.

`BoundaryMap(M)`

A matrix that represents the boundary map from the space of modular symbols M to the vector space whose basis consists of the weight k cusps. (Note: At present there is no intrinsic that lists these cusps.)

Example H139E11

First we compute the cuspidal subspace of the space of modular symbols for $\Gamma_0(11)$.

```
> M := ModularSymbols(11,2); M;
Full modular symbols space for Gamma_0(11) of weight 2 and dimension 3
over Rational Field
> IsCuspidal(M);
false
> C := CuspidalSubspace(M); C;
Modular symbols space for Gamma_0(11) of weight 2 and dimension 2 over
Rational Field
> IsCuspidal(C);
true
```

Next we compute the Eisenstein subspace.

```
> IsEisenstein(C);
false
> E := EisensteinSubspace(M); E;
Modular symbols space for Gamma_0(11) of weight 2 and dimension 1 over
Rational Field
> IsEisenstein(E);
true
> E + C eq M;
true
```

The Eisenstein subspace is the complement of the cuspidal subspace, and conversely.

```
> E eq Complement(C);
true
> C eq Complement(E);
```

```
true
```

Example H139E12

```
> M := ModularSymbols("37B"); M;
Modular symbols space for Gamma_0(37) of weight 2 and dimension 2 over
Rational Field
> BoundaryMap(M);
[0 0]
[0 0]
> A := AmbientSpace(M);
> BoundaryMap(A);
[ 0  0]
[ 0  0]
[ 0  0]
[ 0  0]
[ 1 -1]
```

Observe that the Eisenstein subspace of A is not in the kernel of the boundary map.

```
> Basis(VectorSpace(EisensteinSubspace(A)));
[
    (0 0 0 1 3)
]
```

139.9 Twists

This section is about twists of newforms by Dirichlet characters. A newform is specified by giving a space of modular symbols that contains a single Galois-orbit (over $\mathbf{Q}$ or some extension of $\mathbf{Q}$) of newforms. Spaces of this kind are obtained using `NewformDecomposition`.

To prove that $f_2 = f_1^\chi$ holds for two newforms, the program compares their Hecke eigenvalues up to an appropriate Sturm bound. (For instance, a bound of this kind is given in Lemma 1.4 of [BS02]).

`IsTwist(M1, M2, p)`

Given two spaces `M1` and `M2` of modular symbols that specify newforms f_1 and f_2 as above, and a prime p, this determines whether some Galois conjugate (over $\mathbf{Q}$) of f_2 is the twist of f_1 by a nontrivial Dirichlet character of p-power conductor. If so, a character χ such that $f_2 = f_1^\chi$ is also returned.

`IsMinimalTwist(M, p)`

Given a space `M` of modular symbols that specifies a newform f as above, and a prime p, this determines whether some Galois conjugate (over $\mathbf{Q}$) of f is a twist of some newform of lower level by some Dirichlet character of p-power conductor. If so, it returns `false`, together with the newform of lower level (specified by a space of modular symbols), and the Dirichlet character.

Example H139E13

We exhibit a newform that is a twist of itself, namely the only newform of level 9 and weight 4. The newform is specified by the space of modular symbols on $\Gamma_0(9)$ of weight 4 (with sign 1).

```
> M9 := CuspidalSubspace(ModularSymbols(9, 4, 1));
> newforms := NewformDecomposition(NewSubspace(M9));
> newforms;
[
    Modular symbols space for Gamma_0(9) of weight 4 and dimension 1
    over Rational Field
]
> f := newforms[1];
> Eigenform(f, 20);
q - 8*q^4 + 20*q^7 - 70*q^13 + 64*q^16 + 56*q^19 + O(q^20)
```

Note that here the coefficients for primes congruent to 2 mod 3 are all zero.

```
> bool, chi := IsTwist(f, f, 3);
> bool;
true
> Parent(chi);
Group of Dirichlet characters of modulus 3 over Rational Field
> Conductor(chi), Order(chi);
3 2
```

However, f is not a twist of any newform with lower level:

```
> bool := IsMinimalTwist(f, 3);
> bool;
true
```

139.10 Operators

Each space $\mathbf{M}$ of modular symbols comes equipped with a commuting family $T_1, T_2, T_3, \ldots$ of linear operators acting on it called the Hecke operators.

The Hecke operators are defined recursively, as follows. First, $T_1 = 1$. When $n = p$ is prime,

$$T_p(x) = \left[\begin{pmatrix} p & 0 \\ 0 & 1 \end{pmatrix} + \sum_{r \bmod p} \begin{pmatrix} 1 & r \\ 0 & p \end{pmatrix} \right] x,$$

where the first matrix is omitted if p divides the level N of M. If m and n are coprime, then $T_{mn} = T_m T_n$. If p is a prime, $r \geq 2$ is an integer, ε is the Dirichlet character associated to M, and k is the weight of M, then

$$T_{p^r} = T_p T_{p^{r-1}} - \varepsilon(p) p^{k-1} T_{p^{r-2}}.$$

Example H139E14

In MAGMA, Hecke operators are represented as $n \times n$-matrices, acting from the right, with respect to the basis `Basis(M)`. For example

```
> M := ModularSymbols(12);
> T2 := HeckeOperator(M,2);
> M.1;
{oo, 0}
> T2;
[ 2  0 -1  0  0]
[ 2  0 -1  0  0]
[ 0  0  1 -2 -2]
[ 0 -1  1 -1 -2]
[ 0  1 -1  1  2]
> M.1*T2;
2*{oo, 0} + -1*{-1/10, 0}
```

`HeckeOperator(M, n)`

Compute a matrix representing the nth Hecke operator T_n with respect to `Basis(M)` where M is a space of modular symbols.

`HeckePolynomial(M, n)`

Compute the characteristic polynomial of the Hecke operator T_n with respect to the space of modular symbols M. When n is prime, the Deligne bound on the sizes of Hecke eigenvalues is used, so `HeckePolynomial` is frequently much faster than `CharacteristicPolynomial(HeckeOperator(M,n))`.

`IntegralHeckeOperator(M, n)`

A matrix representing the nth Hecke operator with respect to `Basis(Lattice(M))` where M is a space of modular symbols.

`DualHeckeOperator(M, n)`

Compute a matrix representing the Hecke operator T_n on the dual vector space representation of the space of modular symbols M. This function is much more efficient than `HeckeOperator(M,n)` when the dimension of M is small relative to the dimension of the `AmbientSpace(M)`. Note that `DualHeckeOperator(M,n)` is not guaranteed to be the transpose of `HeckeOperator(M,n)` because `DualHeckeOperator(M,n)` is computed with respect to `Basis(DualVectorSpace(M))`.

`AtkinLehner(M, q)`

A matrix representing the qth Atkin-Lehner involution W_q on the space of modular symbols M, when it is defined. The involution W_q is defined when M has trivial character and even weight. When possible, the Atkin-Lehner map is normalized so that it is an involution; such normalization may not be possible when $k > 2$ and the characteristic of the base field of M divides q.

To each divisor q of N such that $\gcd(q, N/q) = 1$ there is an *Atkin-Lehner involution* W_q on M, which is defined as follows. Using the Euclidean algorithm, choose integers x, y, z, w such that $qxw - (N/q)yz = 1$; let $g = \begin{pmatrix} dx & y \\ Nz & qw \end{pmatrix}$ and define

$$W_q(x) = g(x)/q^{\frac{k-2}{2}}.$$

For example, when $q = N$ we have $g = \begin{pmatrix} 0 & -1 \\ N & 0 \end{pmatrix}$.

`DualAtkinLehner(M, q)`

The action of the Atkin-Lehner involution on the dual representation of the space of modular symbols M, when it is defined.

`StarInvolution(M)`

The conjugation involution $*$ on the space of modular symbols M that sends the modular symbol $X^iY^j\{u, v\}$ to $(-1)^j X^iY^j\{-u, -v\}$.

`DualStarInvolution(M)`

The conjugation involution $*$ on the dual representation of the space of modular symbols M (see the documentation for `StarInvolution`.)

`ThetaOperator(M1, M2)`

Multiplication by $X^pY - XY^p$, which is a possible analogue of the θ-operator. (On mod p modular forms, the θ-operator is the map given by $f \mapsto q\frac{df}{dq}$.) Both M_1 and M_2 must be spaces of modular symbols over a field of positive characteristic p; they must have the same level and character, and the weight of M_2 must equal the weight of M_1 plus $p + 1$.

Example H139E15

```
> M := ModularSymbols(11,4,+1); M;
Full modular symbols space for Gamma_0(11) of weight 4 and dimension 4
over Rational Field
> HeckeOperator(M,2);
[   9    0  2/5 -2/5]
[   0    5  9/5 11/5]
[   0    5  7/5 13/5]
```

```
[   0    0 22/5 23/5]
```

The entries of T_2 are not guaranteed to be integers because `Basis(M)` is just a basis of a $\mathbf{Q}$-vector space. The entries will be integers if we compute T_2 with respect to an integral basis.

```
> IntegralHeckeOperator(M,2);
[ 0  2  0  0]
[ 1  2  0  0]
[-5  6  9  0]
[ 2  0  0  9]
```

The matrix for the Hecke operator on the dual of M is the transpose of T_2. However, the chosen basis for the cuspidal subspace of the dual of M need not satisfy any compatibility with `CuspidalSubspace(M)`.

```
> DualHeckeOperator(M,2);
[   9    0    0    0]
[   0    5    5    0]
[ 2/5  9/5  7/5 22/5]
[-2/5 11/5 13/5 23/5]
> S := CuspidalSubspace(M);
> HeckeOperator(S, 2);
[    5 -13/5]
[    5    -3]
> DualHeckeOperator(S, 2);
[-3/4  1/8]
[-1/2 11/4]
> // NOT the transpose!
```

We can also compute the Atkin-Lehner and the $*$-involution. The $*$-involution is the identity because we are working in the $+1$-quotient, which is the largest quotient of `ModularSymbols(11,4)` where $*$ acts as $+1$.

```
> AtkinLehner(S, 11);
[1 0]
[0 1]
> StarInvolution(S);
[1 0]
[0 1]
```

On the -1 quotient the Atkin-Lehner involution is the same, but $*$ acts as -1:

```
> M := ModularSymbols(11,4,-1); M;
Full modular symbols space for Gamma_0(11) of weight 4 and dimension 2
over Rational Field
> S := CuspidalSubspace(M);
> AtkinLehner(S, 11);
[1 0]
[0 1]
> StarInvolution(S);
[-1  0]
```

```
[ 0 -1]
```

Example H139E16

We compute an example of our analogue of the θ-operator on modular symbols.

```
> N := 11; p := 3;
> k1 := 2; k2 := k1 + (p+1);
> M1 := ModularSymbols(11,k1,GF(p));
> M2 := ModularSymbols(11,k2,GF(p));
> theta  := ThetaOperator(M1,M2); theta;
Mapping from: ModSym: M1 to ModSym: M2 given by a rule [no inverse]
```

Now that we have computed `theta`, we can apply it to one of the modular symbols corresponding to the newform in $S_2(\Gamma_0(11))$.

```
> D := Decomposition(M1,2);
> f := qEigenform(D[2],10); f;
q + q^2 + 2*q^3 + 2*q^4 + q^5 + 2*q^6 + q^7 + q^9 + O(q^10)
> x := D[2].1;
> y := theta(x); y;
(X^4 + X*Y^3)*{-1/7, 0} + (X^4 + X^3*Y + X*Y^3 + Y^4)*{-1/7, 0} + (X^4
+ 2*X^3*Y + 2*X*Y^3 + Y^4)*{-1/5, 0} + Y^4*{oo, 0}
```

Finally, we verify for $n < 10$ that the nth Hecke eigenvalue of $y = \theta(x)$ equals $n \cdot a_n(f)$, where f is as above.

```
> [y*HeckeOperator(M2,n) - n*Coefficient(f,n)*y : n in [1..9]];
[
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    0
]
```

139.11 The Hecke Algebra

`HeckeBound(M)`

A positive integer n such that the Hecke operators $T_1, \ldots, T_n$ generate the Hecke algebra as a $\mathbf{Z}$-module. When the character is trivial, the default bound is $(k/12) \cdot [\mathrm{SL}_2(\mathbf{Z}) : \Gamma_0(N)]$. That this suffices follows from [Stu87], as is explained in [AS02]. When the character of the space of modular symbols M is nontrivial, the default bound is twice the above bound; however, *it is not known that this bound is large enough in all cases in which the character is nontrivial*, so one may wish to increase the bound using `SetHeckeBound`.

`SetHeckeBound(M, n)`

Many computations require a bound n such that $T_1, \ldots, T_n$ generate the Hecke algebra associated to the space of modular symbols M as a $\mathbf{Z}$-module. This command allows you to set the bound that is used internally. Setting it too low can result in functions quickly producing incorrect results.

`HeckeAlgebra(M : Bound)`

The Hecke algebra associated to the space of modular symbols M. This is an algebra `TQ` over $\mathbf{Q}$, such that `Generators(TQ)` is a set that generates the ring $\mathbf{Z}[T_1, T_2, T_3, \ldots]$, as a $\mathbf{Z}$-module. If the optional integer parameter `Bound` is set, then `HeckeAlgebra` only computes the algebra generated by those T_n, with $n \leq$ `Bound`.

`DiscriminantOfHeckeAlgebra(M : Bound)`

The discriminant of the Hecke algebra associated to the space of modular symbols M. If the optional parameter `Bound` is set, then the discriminant of the algebra generated by only those T_n, with $n \leq$ `Bound`, is computed instead.

`HeckeEigenvalueRing(M : parameters)`

`Bound` RngIntElt *Default :* -1

The order generated by the Fourier coefficients of one of the q-expansions of a newform corresponding to the space of modular symbols M, along with a map from the ring containing the coefficients of `qExpansion(A)` to the order. If the optional parameter `Bound` is set, then the order generated only by those a_n, with $n \leq$ `Bound`, is computed.

`HeckeEigenvalueField(M)`

The number field generated by the Fourier coefficients of one of the q-expansions of a newform corresponding to the space of modular symbols M, along with a map from the ring containing the coefficients of `qExpansion(M)` to the number field. We require that M be defined over $\mathbf{Q}$.

Example H139E17

In this example, we compute the discriminant of the Hecke algebra of prime level 389.

```
> M := ModularSymbols(389,2,+1);
> C := CuspidalSubspace(M);
> DiscriminantOfHeckeAlgebra(C);
629670054720061882880174736321392595498204931550235108311040000000
> Factorization($1);
[ <2, 53>, <3, 4>, <5, 6>, <31, 2>, <37, 1>, <389, 1>, <3881, 1>,
<215517113148241, 1>, <477439237737571441, 1> ]
```

The prime 389 is the only prime $p < 10000$ such that p divides the discriminant of the Hecke algebra associated to $S_2(\Gamma_0(p))$. It is an open problem to decide whether or not there are any other such primes. Are there infinitely many?

139.12 The Intersection Pairing

MAGMA can compute the intersection pairing

$$H_1(X_0(N), \mathbf{Q}) \times H_1(X_0(N), \mathbf{Q}) \to \mathbf{Q}$$

on the homology of the modular curve $X_0(N)$. The algorithm that we implemented is essentially the one given in [Mer93]. (Warning: There is a typo in Proposition 4 of [Mer93]; W_i should be replaced by $W_i^{\varepsilon_i}$.)

`IntersectionPairing(x, y)`

The intersection pairing of the homology classes corresponding to the weight-2 cuspidal modular symbols x and y. The symbols x and y must have the same parent, which must have trivial character and not be a $+1$ or -1 quotient.

Example H139E18

In this example, we illustrate several basic properties of the intersection pairing on $H_1(X_0(37), \mathbf{Z})$. First, let `H37` be the space of modular symbols that corresponds to $H_1(X_0(37), \mathbf{Z})$, and compute a basis for `H37`.

```
> M37 := ModularSymbols(37,2);
> H37 := CuspidalSubspace(M37);
> Z := IntegralBasis(H37); Z;
[
    {-1/29, 0},
    {-1/22, 0},
    {-1/12, 0},
    {-1/18, 0}
]
```

Now we compute some intersection numbers.

```
> IntersectionPairing(Z[1],Z[2]);
```

```
-1
> IntersectionPairing(Z[3],Z[4]);
0
```

The intersection pairing is perfect and skew-symmetric, so the matrix that defines it is skew-symmetric and has determinant ± 1 (in fact, it has determinant $+1$).

```
> A := MatrixAlgebra(RationalField(),4);
> I := A![IntersectionPairing(x,y) : x in Z, y in Z]; I;
[ 0  1  0  1]
[-1  0  1  1]
[ 0 -1  0  0]
[-1 -1  0  0]
> I + Transpose(I) eq 0;
true
> Determinant(I);
1
```

The Hecke operators are compatible with the intersection pairing in the sense that $(T_n x, y) = (x, T_n y)$.

```
> T2 := HeckeOperator(M37,2);
> IntersectionPairing(Z[1]*T2,Z[2]);
1
> IntersectionPairing(Z[1],Z[2]*T2);
1
```

It is note the case $(T_n x, T_n y) = (x, y)$ for all n, x, and y.

```
> IntersectionPairing(Z[1]*T2,Z[2]*T2);
-2
```

The existence of the intersection pairing implies that $H_1(X_0(N), \mathbf{Z})$ is isomorphic, as a module over the Hecke algebra, to its linear dual $\mathrm{Hom}(H_1(X_0(N), \mathbf{Z}), \mathbf{Z})$.

139.13 q-Expansions

The following functions should only be called on modular symbols spaces that are cuspidal. For q-expansions of Eisenstein series, use the modular forms functions instead (see the example below).

`Eigenform(M, prec)`

`Eigenform(M)`

`qEigenform(M, prec)`

`qEigenform(M)`

`PowerSeries(M, prec)`

`PowerSeries(M)`

The q-expansion of one of the Galois-conjugate newforms associated to the irreducible cuspidal space M of modular symbols, computed to absolute precision *prec* (which defaults to the highest precision computed in previous calls to this intrinsic, or 8 if none have been computed). The coefficients of the q-expansion lie in a quotient of a polynomial extension of the base field of M. In most cases, it is necessary for M to have been defined using `NewformDecomposition`.

`qExpansionBasis(M, prec : parameters)`

`Al` MonStgElt *Default : "Newform"*

The reduced row-echelon basis of q-expansions for the space of modular forms associated to M, where K is the base field of M. The absolute precision of the q-expansions is *prec*.

The optional parameter `Al` can take the values `"Newform"` and `"Universal"`. The default is `"Newform"`, which computes a basis of q-expansions by finding a decomposition of M into subspaces corresponding to newforms, computing their q-expansions, and then taking all of their images under the degeneracy maps. If `Al := "Universal"` then the algorithm of Section 4.3 of [Mer94] is used. This latter algorithm does not require computing a newform decomposition of M, but requires computing the action of many more Hecke operators. Consequently, in practice, our implementation of Merel's algorithm is usually less efficient than our implementation of the newform algorithm.

`qIntegralBasis(M)`

`qIntegralBasis(M, prec)`

`qIntegralBasis(seq, prec)`

`Al` MonStgElt *Default : "Newform"*

The reduced integral basis of q-expansions for the space of modular forms associated to the given space M of modular symbols (or the given sequence of spaces). The q-expansions are computed to absolute precision *prec*. The base field of M must be either the rationals or a cyclotomic field.

`SystemOfEigenvalues(M, prec)`

The sequence of Hecke eigenvalues $[a_2, a_3, a_5, a_7, \ldots, a_p]$ attached to the space of modular symbols M, where p is the largest prime less than or equal to *prec*. Let K be the base field of M. Then the a_ℓ either lie in K or an extension of K (which may be constructed either as a number field or as a quotient of $K[x]$). We assume that M corresponds to a single Galois-conjugacy class of newforms.

Example H139E19

First we compute a q-basis and a representative newform for the two-dimensional space $S_2(\Gamma_0(23))$. We work in the $+1$ quotient of modular symbols since, for the purpose of computing q-expansions, nothing is lost and many algorithms are more efficient.

```
> M := CuspidalSubspace(ModularSymbols(23,2, +1));
> qExpansionBasis(M);
[
    q - q^3 - q^4 - 2*q^6 + 2*q^7 + O(q^8),
    q^2 - 2*q^3 - q^4 + 2*q^5 + q^6 + 2*q^7 + O(q^8)
]
> f := qEigenform(M,6); f;
q + a*q^2 + (-2*a - 1)*q^3 + (-a - 1)*q^4 + 2*a*q^5 + O(q^6)
> Parent(f);
Power series ring in q over Univariate Quotient Polynomial Algebra
in a over Rational Field with modulus a^2 + a - 1
> PowerSeries(M);
q + a*q^2 + (-2*a - 1)*q^3 + (-a - 1)*q^4 + 2*a*q^5 + (a - 2)*q^6 +
    (2*a + 2)*q^7 + O(q^8)
> SystemOfEigenvalues(M, 7);
[
    a,
    -2*a - 1,
    2*a,
    2*a + 2
]
```

Next we compare an integral and rational basis of q-expansions for $S_2(\Gamma_0(65))$, computed using modular symbols.

```
> S := CuspidalSubspace(ModularSymbols(65,2,+1));
> qExpansionBasis(S);
[
    q + 1/3*q^6 + 1/3*q^7 + O(q^8),
    q^2 - 1/3*q^6 + 2/3*q^7 + O(q^8),
    q^3 - 4/3*q^6 + 2/3*q^7 + O(q^8),
    q^4 - 1/3*q^6 + 5/3*q^7 + O(q^8),
    q^5 + 5/3*q^6 + 2/3*q^7 + O(q^8)
]
> qIntegralBasis(S);
[
    q + q^5 + 2*q^6 + q^7 + O(q^8),
    q^2 + 2*q^5 + 3*q^6 + 2*q^7 + O(q^8),
    q^3 + 2*q^5 + 2*q^6 + 2*q^7 + O(q^8),
    q^4 + 2*q^5 + 3*q^6 + 3*q^7 + O(q^8),
    3*q^5 + 5*q^6 + 2*q^7 + O(q^8)
```

```
]
```

If you're interested in q-expansions of Eisenstein series, see the chapter on modular forms. For example:

```
> E := EisensteinSubspace(ModularForms(65,2));
> Basis(E);
[
    1 + O(q^8),
    q + 3*q^2 + 4*q^3 + 7*q^4 + 12*q^6 + 8*q^7 + O(q^8),
    q^5 + O(q^8)
]
```

139.14 Special Values of L-functions

Let M be an irreducible space of cuspidal modular symbols defined over $\mathbf{Q}$, irreducible in the sense that M corresponds to a single Galois-conjugacy class of cuspidal newforms. Such an M can be computed using `NewformDecomposition`. Let $f^{(1)}, \ldots, f^{(d)}$ be the $\mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})$-conjugate newforms that correspond to M, and write $f^{(d)} = \sum_{n=1}^{\infty} a_n^{(d)} q^n$. By a theorem of Hecke, the Dirichlet series

$$L(f^{(i)}, s) = \sum_{n=1}^{\infty} \frac{a_n^{(i)}}{n^s}$$

extends (uniquely) to a holomorphic function on the whole complex plane. Of particular interest is the special value

$$L(M, j) = L(f^{(1)}, j) \cdots L(f^{(d)}, j),$$

for any $j \in \{1, 2, \ldots, k-1\}$.

In this section we describe how to approximate the complex numbers $L(M, j)$ in MAGMA. If you are interested in computing individual special values $L(f^{(i)}, j)$, then you should use the modular forms package instead of the modular symbols package for this.

The variable `prec` below refers to the number of terms of the q-expansion of each $f^{(i)}$ that are used in the computation, and not to the number of decimals of the answer that are correct. Thus, for example, to get a heuristic idea of the quality of an answer, you can increase `prec`, make another call to `LSeries`, and observe the difference between the two answers. If the difference is "small", then the approximation is probably "good".

`LSeries(M, j, prec)`

The special value $L(M, j)$, where j is an integer that lies in the critical strip, so $1 \leq j \leq k-1$ with k the weight of M. Here M is a space of modular symbols with sign 0, and *prec* is a positive integer which specifies the numbers of terms of q-expansions to use in the computation.

`LSeriesLeadingCoefficient(M, j, prec)`

The leading coefficient of Taylor expansion about the critical integer j and order of vanishing of $L(M,s)$ at $s=1$. Thus if the series expansion of $L(M,s)$ about $s=1$ is

$$L(M,s) = a_r(s-1)^r + a_{r+1}(s-1)^{r+1} + a_{r+2}(s-1)^{r+2} + \cdots,$$

then the leading coefficient of $L(M,s)$ is a_r and the order of vanishing is r.

`RealVolume(M, prec)`

The volume of $A_M(\mathbf{R})$, which is defined as follows. Let $S \subset \mathbf{C}[[q]]$ be the space of cusp forms associated to M. Choose a basis $f_1, \ldots, f_d$ for the free $\mathbf{Z}$-module $S \cap \mathbf{Z}[[q]]$; one can prove that $f_1, \ldots, f_d$ is also a basis for S. There is a period map Φ from integral cuspidal modular symbols H to $\mathbf{C}^d$ that sends a modular symbol $x \in H$ to the d-tuple of integrals $(\langle f_1, x\rangle, \ldots, \langle f_d, x\rangle) \in \mathbf{C}^d$. The cokernel of Φ is isomorphic to $A_M(\mathbf{C})$. Moreover, the standard measure on the Euclidean space $\mathbf{C}^d$ induces a measure on $A_M(\mathbf{R})$. It is with respect to this measure that we compute the volume. For more details, see Section 3.12.16 of [Ste00].

`MinusVolume(M, prec)`

The volume of the subgroup of $A_M(\mathbf{C})$ on which complex conjugation acts as -1.

`LRatio(M, j : parameters)`

`Bound` RngIntElt *Default :* -1

The rational number

$$\frac{L(A,j)\cdot(j-1)!}{(2\pi)^{j-1}\cdot\Omega,}$$

where j is a "critical integer", so $1 \le j \le k-1$, and Ω is `RealVolume(M)` when j is odd and `MinusVolume(M)` when j is even. If the optional parameter Bound is set, then `LRatio` is only a divisibility upper bound on the above rational number. If `Sign(M)` is not 0, then `LRatio(M,j)` is only correct up to a power of 2.

`LRatioOddPart(M, j)`

The odd part of the rational number `LRatio(M,j)`. Hopefully, computing `LRatioOddPart(M,j)` takes less time than finding the odd part of `LRatio(M,j)`.

Example H139E20

```
> M := ModularSymbols(11,2);
> C := CuspidalSubspace(M);
> LSeries(C,1,100);
0.25384186085591068433775892 33
> A := ModularSymbols("65B"); A;  // <--> dimension two abelian variety
Modular symbols space of level 65, weight 2, and dimension 4
> LSeries(A,1,100);
0.91225158869818984109351402 11 +  0.E-29*i
```

139.14.1 Winding Elements

Let $\mathbf{M_k(N)}$ be a space of modular symbols over $\mathbf{Q}$. For $i = 1, \ldots, k$, the *ith winding element*

$$\mathbf{e}_i = X^{i-1}Y^{k-2-(i-1)}\{0, \infty\} \in \mathbf{M_k(N)}$$

is of importance for the computation of special values. For any modular form $f \in S_k(N)$ and homogeneous polynomial $P(X, Y)$ of degree $k - 2$, let

$$\langle f, P(X, Y)\{0, \infty\}\rangle = -2\pi i \cdot \int_0^{i\infty} f(z)P(z, 1)\mathrm{dz}.$$

Fix a newform $f \in S_k(N)$ corresponding to a space M of modular symbols, and let j be a integer in $\{0, 1, \ldots, k - 1\}$. The winding element is significant because

$$L(f, j) = \frac{(2\pi)^{j-1}}{i^{j+1}(j-1)!} \cdot \langle f, X^{j-1}Y^{k-2-(j-1)}\{0, \infty\}\rangle.$$

Moreover, the submodule that is generated by the winding element is used in the formula for a canonical rational part of the number $L(M, j)$ (see `LRatio`, above).

`WindingElement(M)`

The winding element $Y^{k-2}\{0, \infty\}$.

`WindingElement(M, i)`

The winding element $X^{i-1}Y^{k-2-(i-1)}\{0, \infty\}$.

`TwistedWindingElement(M, i, eps)`

The element $\sum_{a\in(\mathbf{Z}/m\mathbf{Z})^*} \varepsilon(a)X^{i-1}Y^{k-2-(i-1)}\{0, \frac{a}{m}\}$.

`WindingLattice(M, j : parameters)`

`Bound` RngIntElt *Default :* -1

The image under `RationalMapping(M)` of the lattice generated by the images of the jth winding element under all Hecke operators T_n. If M is the ambient space, then the image under `RationalMapping(M)` is not taken.

`WindingSubmodule(M, j : parameters)`

`Bound` RngIntElt *Default :* -1

The image under `RationalMapping(M)` of the vector space generated by all images of `WindingElement(M,j)` under all Hecke operators T_n. If M is the ambient space, then the image under the rational period mapping is not taken.

`TwistedWindingSubmodule(M, j, eps)`

The Hecke submodule of the vector space $\Phi(M)$ generated by the image of the jth ε-twisted modular winding element, where Φ is `RationalMapping(M)`. This module is useful, for example, because in characteristic 0, if M is new of weight 2, has sign $+1$ or -1, and corresponds to a collection $\{f_i\}$ of Galois-conjugate newforms, then the dimension of the twisted winding submodule equals the cardinality of the subset of f_i such that $L(f_i, eps, 1) \neq 0$.

139.15 The Associated Complex Torus

Let M be a space of cuspidal modular symbols, which is the kernel of an ideal in the Hecke algebra. When M has weight 2 there is an abelian variety A_M attached to M; more generally, there is a complex torus $A_M(\mathbf{C})$ attached to M. The associated complex torus $A_M(\mathbf{C})$ is constructed as follows. Let S be the space of modular forms corresponding to M. The integration pairing gives rise to a natural map $M \to \mathrm{Hom}(S, \mathbf{C})$, and the cokernel of this map is $A_M(\mathbf{C})$.

`SubgroupOfTorus(M, x)`

The cyclic subgroup of the complex torus attached to the space of modular symbols M that is generated by the image under the period map of the modular symbol x.

`SubgroupOfTorus(M, s)`

An abelian group that is isomorphic to the finite group generated by the sequence of images $\pi(s[i])$ in the complex torus attached to M, where π is `PeriodMapping(M)`.

Example H139E21

The cuspidal subgroup of $J_0(N)$ is the subgroup generated by the degree 0 divisors on $X_0(N)$ of the form $(\alpha) - (\beta)$, where α and β are cusps. The following examples illustrate how to use the above functions to compute the cuspidal subgroup, as an abstract abelian group.
The modular symbols approach has the advantage that it is essentially no more complicated for N highly composite than for N prime. However, it is only applicable when the corresponding space of modular symbols can be computed in a reasonable amount of time, which at present means that N should have less than 5 decimal digits. There are other methods which may be much more efficient in special cases. For example, when p is prime Andrew Ogg showed that the cuspidal subgroup of $J_0(p)$ is cyclic of order equal to the numerator of $(p-1)/12$. More generally, he gave a simple formula for the order of the cuspidal subgroup when $N = pq$ is the product of two primes. See also papers of Ligozat.

```
> M := ModularSymbols(20); M;
Full Modular symbols space of level 20, weight 2, and dimension 7
> e := M ! <1, [Cusps()|0,Infinity()] >;   // the path from 0 to infinity
> e;
-1*{oo, 0}
> J0of20 := CuspidalSubspace(M);
```

```
> A := SubgroupOfTorus(J0of20, e); A;
Abelian Group isomorphic to Z/6
Defined on 1 generator
Relations:
    6*A.1 = 0
> // Next, the subgroup generated by all cusps
> A := SubgroupOfTorus(J0of20, IntegralBasis(M));  A;
Abelian Group isomorphic to Z/6
Defined on 1 generator
Relations:
    6*A.1 = 0
> // Let's do another example.
> M := ModularSymbols(100);
> J0of100 := CuspidalSubspace(M);
> A := SubgroupOfTorus(J0of100, IntegralBasis(M));  A;
Abelian Group isomorphic to Z/6 + Z/30 + Z/30 + Z/30 + Z/30
Defined on 5 generators
Relations:
    6*A.1 = 0
    30*A.2 = 0
    30*A.3 = 0
    30*A.4 = 0
    30*A.5 = 0
> M := ModularSymbols(77);
> J0of77 := CuspidalSubspace(M);
> A := SubgroupOfTorus(J0of77, IntegralBasis(M)); A;
Abelian Group isomorphic to Z/10 + Z/60
Defined on 2 generators
Relations:
    10*A.1 = 0
    60*A.2 = 0
> M := ModularSymbols(97);
> A := SubgroupOfTorus(CuspidalSubspace(M), IntegralBasis(M)); A;
Abelian Group isomorphic to Z/8
Defined on 1 generator
Relations:
    8*A.1 = 0
> Numerator((97-1)/12);
8
```

Example H139E22

The following code creates a file that contains a table which lists, for each integer N in some range, the abstract group structure of the subgroup of $J_0(N) = \mathrm{Jac}(X_0(N))$ generated by the cusps $(\alpha) - (\infty)$, with $\alpha \in \mathbf{Q} \cup \{\infty\}$.

```
> function CuspidalSubgroup(N)
>    M := ModularSymbols(N);
```

```
>    J := CuspidalSubspace(M);
>    G := SubgroupOfTorus(J,IntegralBasis(M));
>    return G;
> end function;
> // Test the function
> CuspidalSubgroup(65);
Abelian Group isomorphic to Z/2 + Z/84
Defined on 2 generators
Relations:
    2*$.1 = 0
    84*$.2 = 0
> procedure CuspidalTable(start, stop)
>    fname := Sprintf("cuspidal_subgroup_%o-%o.m", start, stop);
>    file  := Open(fname,"w");
>       for N in [start..stop] do
>       G := Invariants(CuspidalSubgroup(N));
>       fprintf file, "C[%o] := \t%o;\n\n", N, G;
>       printf "C[%o] := \t%o;\n\n", N, G;
>       Flush(file);
>    end for;
> end procedure;
```

`ModularKernel(M)`

The kernel of the modular isogeny. Let T be the complex torus attached to the space of modular symbols M. Then the modular isogeny is the natural map from the dual of T into T induced by autoduality of `CuspidalSubspace(AmbientSpace(M))`.

`CongruenceGroup(M : parameters)`

`Bound` RngIntElt *Default :* -1

The congruence group of the space of cusp forms corresponding to the space of cuspidal modular symbols M. Let $S = S_k(\Gamma_0(N), \mathbf{Z})$, let V be the sub $\mathbf{Z}$-module corresponding to M, and W be its orthogonal complement. Then the congruence group is $S/(V + W)$. This group encodes information about congruences between forms in V and forms in the complement of V.

The optional parameter `Bound` is a positive integer b such that the q-expansions of cusp forms are computed to absolute precision b. If the bound is too small, then `CongruenceGroup` will give only an upper bound on the correct answer. The default is `HeckeBound(M) + 1`, which gives a provably correct answer.

`IntersectionGroup(M1, M2)`

An abelian group G that encodes information about the intersection of the complex tori corresponding to the spaces of modular symbols M_1 and M_2. We require that M_1 and M_2 lie in a common ambient space. When the `IntersectionGroup(M1,M2)` is finite, it is isomorphic to $A_{M_1}(\mathbf{C}) \cap A_{M_2}(\mathbf{C})$.

`IntersectionGroup(S)`

An abelian group G that encodes information about the intersection of the collection of complex tori corresponding to the sequence S of spaces of modular symbols.

Example H139E23

In this example, we investigate a 2-dimensional abelian variety B, which is a quotient of $J_0(43)$. The purpose of this example is to show how numerical computation with modular symbols suggests interesting arithmetic questions about familiar abelian varieties. In the following example, we find that the conjecture of Birch and Swinnerton-Dyer (plus the Manin $c = 1$ conjecture) implies that the first nontrivial Shafarevich-Tate group of an (optimal) modular abelian variety has order *two*. Thus the surprising existence of an abelian varieties with non-square order could have been (but was not) hinted at long ago by somebody playing around with a modular symbols package (in fact, it was discovered by B. Poonen and M. Stoll [PS99] while they were designing and implementing algorithms for computing with Jacobians of genus-two curves).

```
> M43 := ModularSymbols(43,2);     // Level 43, weight 2.
> H1  := CuspidalSubspace(M43);    // H_1(X_0(43),Q)
> D   := NewformDecomposition(H1); // factors corresponding to newforms
> A,B := Explode(D);
> A;     // The homology of the elliptic curve "43A"
Modular symbols space of level 43, weight 2, and dimension 2
> B;     // The homology of the 2-dimensional abelian variety "43B"
Modular symbols space of level 43, weight 2, and dimension 4
> LRatio(B,1);        // L(B,1)/Omega_B
2/7
```

The Birch and Swinnerton-Dyer conjecture predicts that the Shafarevich-Tate group of B has order as given by the formula for `ShaAn` in the code below. To compute this value, it remains to compute $\#B(\mathbf{Q})$ and the Tamagawa number c_{43}.

```
> T := TorsionBound(B,11);  T; // #B(Q) divides this number
7
> // Compute the subgroup of B(Q) generated by (0)-(oo).
> C := SubgroupOfTorus(B,WindingElement(M43));  C;
Abelian Group isomorphic to Z/7
Defined on 1 generator
Relations:
    7*C.1 = 0
> TamagawaNumber(B,43);
7
> ShaAn := LRatio(B,1)*TorsionBound(B,11)^2/TamagawaNumber(B,43);
```

`ShaAn` is the Birch and Swinnerton-Dyer conjectural order of the Shafarevich-Tate group of B, under the assumption that the Manin constant of B is 1.

```
> ShaAn;
2
```

One of the Galois conjugate newforms associated to B is given below.

```
> qEigenform(B,12);
```

```
q + a*q^2 - a*q^3 + (-a + 2)*q^5 - 2*q^6 + (a - 2)*q^7 - 2*a*q^8 - q^9
    + (2*a - 2)*q^10 + (2*a - 1)*q^11 + O(q^12)
> BaseRing(Parent(qEigenform(B,12)));
Univariate Quotient Polynomial Algebra in a over Rational Field
with modulus a^2 - 2
> qIntegralBasis(B,12);
[
    q + 2*q^5 - 2*q^6 - 2*q^7 - q^9 - 2*q^10 - q^11 + O(q^12),
    q^2 - q^3 - q^5 + q^7 - 2*q^8 + 2*q^10 + 2*q^11 + O(q^12)
]
```

By integrating homology against the differentials corresponding to the two modular forms above, we obtain a lattice that defines the complex torus $A_B(\mathbf{C})$:

```
> Periods(B,97);
[
    (-0.22594995830676421187395192 -
        1.76664467629959953227333314 0*i
        0.52502811591322194337294916 48 +
        0.80660185770293072302831423 71*i),
    (0.59815631622412229864757672 20 -
        1.92008563861211949327648563 2*i
        0.82410627422619603486491720 82 -
        0.15344096225717705687483549 95*i),
    (-0.82410627453088651052152864 45 -
        0.15344096231251996100315249 20*i
        -0.29907815831297409149196804 34 -
        0.96004281996010777990314973 67*i),
    (-0.59815631622412229864757672 20 -
        1.92008563861211949327648563 2*i
        -0.82410627422619603486491720 83 -
        0.15344096225717705687483549 95*i)
]
```

Finally, it is tempting to ask whether or not the (conjectural) two-torsion element of the Shafarevich-Tate group of B suggested above is "visible" in the sense that it is "explained by a jump in the rank of the Mordell-Weil group of A" (see [CM00]). The following computation suggests, but does not prove, that this is the case.

```
> G := MordellWeilGroup(EllipticCurve(A)); G;
Abelian Group isomorphic to Z
Defined on 1 generator (free)
> IntersectionGroup(A,B);
Abelian Group isomorphic to Z/2 + Z/2
Defined on 2 generators
Relations:
    2*$.1 = 0
    2*$.2 = 0
```

139.15.1 The Period Map

Let M be a space of modular symbols the corresponds to a Galois-conjugacy class of newforms. The *period map* attached to M is a linear map

$$\text{AmbientSpace(M)} \to \mathbf{C}^d,$$

where d is the dimension of the space of modular forms associated to M. The cokernel of the period map is a complex torus $A_M(\mathbf{C})$. The terminology "period mapping" comes from the fact that there are (often?) mereomorphic functions on $\mathbf{C}^d$ whose periods are the image of the integral cuspidal modular symbols under the period mapping.

In the functions below, M must not be a $+1$ or -1 quotient and must be cuspidal.

`PeriodMapping(M, prec)`

The period mapping attached to the space of modular symbols M, computed using *prec* terms of the q-expansions of modular forms associated to M.

`Periods(M, prec)`

The complex period lattice associated to the space of modular symbols M, computed using *prec* terms of the q-expansions of modular forms associated to M.

`ClassicalPeriod(M, j, prec)`

The value

$$r_j(f) = \int_0^{i\infty} f(z) z^j dz.$$

139.15.2 Projection Mappings

Let M be a space of modular symbols over a field K. For many purposes it is useful to have a surjective map

$$\pi : \text{AmbientSpace(M)} \to V,$$

where V is a vector space over K and $\ker(\pi)$ is the same as the kernel of the period mapping.

`RationalMapping(M)`

A surjective linear map from the ambient space of the space of modular symbols M to a vector space, such that the kernel of this map is the same as the kernel of the period mapping.

`IntegralMapping(M)`

A surjective linear map from the ambient space of the space of modular symbols M to a vector space, such that the kernel of this map is the same as the kernel of the period mapping. This map is chosen in such a way that the image of

`IntegralBasis(CuspidalSubspace(AmbientSpace(M)))`

is the standard $\mathbf{Z}$-lattice. (Note that M must be defined over $\mathbf{Q}$.)

Example H139E24

```
> M := ModularSymbols(33); M;
Full Modular symbols space of level 33, weight 2, and dimension 9
> S := CuspidalSubspace(M);
> N := NewSubspace(S);
> phi := RationalMapping(N);
> [phi(x) : x in IntegralBasis(S)];
[
    ( -2 4/3),
    ( -4 2/3),
    ( -2 2/3),
    (-2  0),
    (  -2 -2/3),
    (-4  0)
]
```

Notice that the image of the basis `IntegralBasis(S)` for $H_1(X_0(33), \mathbf{Z})$ is not $\mathbf{Z} \times \mathbf{Z}$. However, `IntegralMapping(N)` is normalized so that the image is $\mathbf{Z} \times \mathbf{Z}$:

```
> int := IntegralMapping(N);
> [int(S.i) : i in [1..Dimension(S)]];
[
    ( 2 -1),
    ( 1 -2),
    ( 1 -1),
    ( 0 -2),
    ( 0 -1),
    (-1 -1)
]
```

Consider a quotient A_f of $J_0(N)$ attached to a newform $f \in S_2(\Gamma_0(N))$. Using `IntegralMapping` and the Abel-Jacobi theorem, we can see the image in $A_f(\mathbf{Q})$ of the point $(0) - (\infty) \in J_0(N)(\mathbf{Q})$. In the level 97 example below, this image has order 8, which is the numerator of $(97-1)/12$.

```
> Af := ModularSymbols("97B"); Af;
Modular symbols space of level 97, weight 2, and dimension 8
> int := IntegralMapping(Af);
> // Let x be the modular symbol {0,oo}
> x := AmbientSpace(Af)!<1,[Cusps()|0,Infinity()]>;
> int(x);
(-5/8  1/4 -1/4    0    0  1/4  3/8  1/4)
> Numerator((97-1)/12);
8
```

139.16 Modular Abelian Varieties

Let M be a space of weight 2 cuspidal modular symbols with trivial character that corresponds to a Galois-conjugacy class of newforms, and let $A_M(\mathbf{C})$ be the cokernel of the period map. G. Shimura proved that $A_M(\mathbf{C})$ is the set of complex points of an abelian variety A_M defined over $\mathbf{Q}$. Let N be the level of M and let $J_0(N)$ be the Jacobian of the modular curve $X_0(N)$. Shimura constructed A_M as a quotient of $J_0(N)$ by an abelian subvariety. More precisely, if I is the annihilator of M in the Hecke algebra, then $A_M = J_0(N)/IJ_0(N)$.

When A_M has dimension 1 it is an elliptic curve, and the theory of computing with A_M is well developed, though many interesting problems remain. In the contrary case, when A_M has dimension greater than 1, the theory of computation with A_M is still in its infancy. Fortunately, it is possible to compute a number of interesting quantities about A_M using algorithms that rely on our extensive knowledge of $J_0(N)$.

MAGMA contains functions for computing the modular degree, congruence modulus, upper and lower bounds on the order of the torsion subgroup, and the order of the component group of the closed fiber of the Néron model of A_M at primes that exactly divide the level of M.

139.16.1 Modular Degree and Torsion

`ModularDegree(M)`

The modular degree of the space of modular symbols M, which is defined as follows. Let M be a space of modular symbols of weight 2 and trivial character. The modular degree of M is the square root of `#ModularKernel(M)`. When M corresponds to an elliptic curve $E = A_M$, then the modular degree of M is the degree of induced map $X_0(N) \to E$.

`CongruenceModulus(M : parameters)`

`Bound` RNGINTELT *Default :* -1

The congruence number r of the space of modular symbols M. This is the index in $S_k(\Gamma_0(N), \mathbf{Z})$ of the sum $L + W$ of the lattice W of cusp forms L corresponding to M and the lattice of cusp forms corresponding to the complement of L in S.

`TorsionBound(M, maxp)`

The following upper bound on the order of the torsion subgroup of the abelian variety A attached to the space of modular symbols M:

$$\gcd\{\#A(\mathbf{F}_p) : 3 \leq p \leq \text{maxp},\ \ p \not| N\},$$

where N is the level of M. This bound is an isogeny invariant, so it is also a bound on the order of the torsion subgroup of the dual abelian variety $A^\vee$ of A.

To compute a lower bound, use `#SubgroupOfTorus(M,WindingElement(M))`.

Example H139E25

We compute the first example of an optimal elliptic curve over $\mathbf{Q}$ such that the congruence modulus does not equal the modular degree. (See [FM99] for further discussion of this problem. We warn the reader that the divisibility $r \mid \deg(\phi) \mid rN^i$ cited there is incorrect, as our `54B` example shows.)

```
> E := ModularSymbols("54B");
> ModularDegree(E);
2
> CongruenceModulus(E);
6
```

We next verify directly that the congruence modulus is divisible by 3.

```
> A := ModularSymbols("27A"); A;   // 27=54/2.
Modular symbols space of level 27, weight 2, and dimension 2
> A54 := ModularSymbols(A,54); A54;  // all images of A at level 54.
Modular symbols space of level 54, weight 2, and dimension 4
> qE := qIntegralBasis(E,17);
> qA54 := qIntegralBasis(A54,17);
> &+qA54 - &+qE;
-3*q^4 + 3*q^5 - 3*q^8 + 3*q^10 - 3*q^11 + 9*q^13 + 3*q^16 + O(q^17)
> IntersectionGroup(E,A54);   // however, the intersection is trivial.
Abelian Group of order 1
```

Ken Ribet proved that if E is an optimal elliptic curve quotient of $J_0(N)$, with N prime, and if f_E is the corresponding newform, then the congruence modulus of f_E equals the modular degree of E. The author is aware of no counterexamples to the following more general statement: "If E is an optimal elliptic curve of square-free conductor, then the congruence modulus of the newform f_E attached to E equals the modular degree of E." An analogous statement for abelian varieties is false, even at prime level. The first counterexample is `ModularSymbols("431F")`, which corresponds to an abelian variety of dimension 24. In this case, the modular degree is $2^{11} \cdot 6947$, whereas the congruence modulus is $2^{10} \cdot 6947$.
The following code makes a table of congruence moduli and modular degrees for the elliptic curves of conductor near 54. Notice the counterexample at level 54.

```
> for N in [53..55] do
>    C := CuspidalSubspace(ModularSymbols(N,2));
>    newforms := NewSubspace(C);
>    D := EllipticFactors(newforms,19);
>    for E in D do
>       printf "%o:\t%o,\t%o\n", N, ModularDegree(E), CongruenceModulus(E);
>    end for;
> end for;
53:     2,      2
54:     2,      6
54:     6,      6
55:     2,      2
```

`ModularKernel` makes sense even for spaces of modular symbols of weight greater than 2. As in the case of weight 2, this number gives information about congruences between modular forms.

The following example illustrates how `ModularKernel` suggest a congruence between a form of level 10 and weight 4 with a form of level 5.

```
> M := ModularSymbols(10,4);
> S := CuspidalSubspace(M);
> D := NewformDecomposition(S); D;
[
    Modular symbols space of level 10, weight 4, and dimension 2,
    Modular symbols space of level 10, weight 4, and dimension 4
]
> #ModularKernel(D[1]);
10
> f := qEigenform(D[1],8);
> g := qEigenform(D[2],8);
> g2 := Evaluate(g,Parent(g).1^2);
> f-(g+6*g2);   // a congruence modulo 10!
-10*q^3 + 20*q^4 + 10*q^5 - 20*q^6 - 10*q^7 + O(q^8)
```

139.16.2 Tamagawa Numbers and Orders of Component Groups

We provide several functions for computing the orders of component groups of optimal quotients of $J_0(N)$ at primes p that exactly divide N. Our algorithm involves Grothendieck's monodromy pairing on the character group of the toric part of the closed fiber at p of the Néron model of $J_0(N)$; the theory behind this algorithm is described in [Ste01] (or [Ste00]); see [KS00] for a computationally-oriented introduction to the algorithm. When N is prime, we use the Mestre and Oesterlé method to construct the character group of the torus, as described in [Mes86]. In general, the ideal theory of quaternion algebras is used.

Note: In the appendix to [Maz77], Mazur and Rapaport give an explicit formula for the order of the component group of $J_0(N)$ at primes $p \geq 5$ that exactly divide N. Their formula is not currently used by the `ComponentGroupOrder` function.

The `RealTamagawaNumber` function computes the order of the "component group at infinity".

`ComponentGroupOrder(M, p)`

The order of the component group at p. This is the order of the group of $\overline{\mathbf{F}}_p$-points of the component group of the reduction modulo p of the Néron model of the abelian variety attached to the space of modular symbols M. At present, it is necessary that p exactly divides the level. If `Sign(M)` is not equal to 0, then only the odd part of the order is returned.

`TamagawaNumber(M, p)`

The order of the group of $\mathbf{F}_p$-rational points of the component group of the space of modular symbols M. We require M to be associated to a single Galois-conjugacy class of newforms.

`RealTamagawaNumber(M)`

The number of connected components of $A_M(\mathbf{R})$.

`MinusTamagawaNumber(M)`

The number of connected components of the subgroup $A_M(\mathbf{C})^-$ of $A_M(\mathbf{C})$ on which complex conjugation acts as -1

Example H139E26

We compute the orders of the component groups of some abelian varieties.

```
> X11 := ModularSymbols("11A");     // corresponds to X_0(11).
> ComponentGroupOrder(X11,11);
5
> TamagawaNumber(X11,11);
5
> RealTamagawaNumber(X11);
1
> MinusTamagawaNumber(X11);
1
> J37 := ModularSymbols("37"); J37;
Modular symbols space of level 37, weight 2, and dimension 4
> ComponentGroupOrder(J37,37);
3
> A, B := Explode(NewformDecomposition(J37));
> ComponentGroupOrder(A,37);
3
> ComponentGroupOrder(B,37);
1
```

We can also compute component groups of optimal quotients whose dimension is greater than 1. The abelian varieties `B` and `C` below correspond to the Jacobians labeled $65B$ and $65A$ in [FLS$^+$01], respectively.

```
> J65 := ModularSymbols("65");
> A,B,C := Explode(SortDecomposition(NewformDecomposition(J65)));
> B;
Modular symbols space of level 65, weight 2, and dimension 4
> C;
Modular symbols space of level 65, weight 2, and dimension 4
> ComponentGroupOrder(B,5);      // not the Tamagawa number
3
> ComponentGroupOrder(B,13);
3
> ComponentGroupOrder(C,5);
7
> ComponentGroupOrder(C,13);
1
```

```
> HeckeEigenvalueField(C);
Number Field with defining polynomial x^2 + 2*x - 1 over the
Rational Field
Mapping from: Univariate Quotient Polynomial Algebra in a over
Rational Field
with modulus a^2 + 2*a - 1 to Number Field with defining
polynomial x^2 + 2*x - 1 over the Rational Field given by a rule
[no inverse]
> ComponentGroupOrder(J65,5);
42
```

When the Atkin-Lehner involution W_p acts as $+1$ on a modular abelian variety A, the order of the component group can be larger than the Tamagawa number $c_p = [A(\mathbf{Q}_p) : A_0(\mathbf{Q}_p)]$ that appears in the conjecture of Birch and Swinnerton-Dyer.

```
> AtkinLehner(B,5);
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
> ComponentGroupOrder(B,5);
3
> TamagawaNumber(B,5);
1
```

The real and minus Tamagawa numbers are defined for spaces of modular symbols of any weight over the rationals.

```
> Del := ModularSymbols("1k12A");
> Del;
Modular symbols space of level 1, weight 12, and dimension 2
```

Next we see that the period lattice associated to Δ is rectangular.

```
> RealTamagawaNumber(Del);
2
> MinusTamagawaNumber(Del);
2
> Periods(Del,40);
[
    (-0.00048533816492995160492413044 29*i),
    (0.00114073744958307 9336044545337)
]
```

139.17 Elliptic Curves

Let E be an elliptic curve. By the modularity theorem, which was recently proved by Breuil, Conrad, Diamond, Taylor, and Wiles there is a two-dimensional space M of modular symbols attached to E. Let N be the conductor of E; then M is obtained from `ModularSymbols(N,2)` by intersecting the kernels of $T_p - a_p(E)$ for sufficiently many p.

Warning: The computation of M can already be very resource intensive for elliptic curves for which `Conductor(E)` is on the order of 5000. For example, the seemingly harmless expression `ModularSymbols(EllipticCurve([0,6]))` would bring my computer to its knees.

`ModularSymbols(E)`

`ModularSymbols(E, sign)`

The space M of modular symbols associated to the elliptic curve E.

`EllipticCurve(M)`

`Database`	BoolElt	*Default :* `true`

An elliptic curve over the rational numbers that lies in the isogeny class of elliptic curves associated to M.

By default, the Cremona database is used. To compute the curve from scratch, set the optional parameter `Database` to `false`. (Note however that this is not optimized for large level.)

`pAdicLSeries(E, p)`

`CoefficientPrecision`	RngIntElt	*Default :* 3
`SeriesPrecision`	RngIntElt	*Default :* 5
`QuadraticTwist`	RngIntElt	*Default :* 1
`ProperScaling`	BoolElt	*Default :* `true`

Given an elliptic curve over the rationals and a prime of multiplicative or good ordinary reduction, compute the p-adic L-series. For technical reasons involving caching, the curve is required to be given by a (global) minimal model.

This computation can be rather time-consuming for large conductors and/or primes. The algorithm needs to compute the modular symbols (taking N^3 time or so), and typically needs to evaluate $2(p-1)p^n$ of them, where n is the desired coefficient precision. Also, the desired coefficient precision does not always apply to every term in the series.

The `QuadraticTwist` vararg computes the p-adic L-series for a twist by a fundamental discriminant D. This requires $|D|$ as many modular symbols to be computed, but in most cases is much faster than computing the modular symbols for the twisted curve. The twisting discriminant is required to be coprime to the conductor of the curve.

The `ProperScaling` vararg indicates whether to ensure that the scaling of the p-adic L-series is correct. For many applications, only the valuation of the coefficients

is needed. The computation of the proper scaling induces a bit of extra overhead, but is still typically faster than computing the modular symbols for examples of interest.

Example H139E27

We use the elliptic curve functions to numerically compute the Birch and Swinnerton-Dyer conjectural order of the Shafarevich-Tate group of the elliptic curve **389A**, which is the curve of rank 2 with smallest conductor. The Birch and Swinnerton-Dyer conjecture asserts that

$$\frac{L^{(r)}(E,1)}{r!} = \frac{\prod c_p \cdot \mathrm{Sha} \cdot \mathrm{Reg}}{|E(\mathbf{Q})_{\mathrm{tor}}|^2},$$

where r is the order of vanishing of $L(E,s)$ at $s=1$.

```
> E := EllipticCurve(CremonaDatabase(),"389A");
> M := ModularSymbols(E);
> M;
Modular symbols space of level 389, weight 2, and dimension 2
> LRatio(M,1);
0
```

Next we compute the analytic rank and the leading coefficient of the L-series at $s=1$. (If your computer is very slow, use a number smaller than 300 below.)

```
> L1, r := LSeriesLeadingCoefficient(M,1,300);
> L1;
0.75931650029224679065762600 31
> r;        // The analytic rank is 2.
2
```

Finally we check that the rank conjecture is true in this case, and compute the conjectural order of the Shafarevich-Tate group.

```
> Rank(E);  // The algebraic rank is 2.
2
> Omega := Periods(M,300)[2][1] * 2; Omega;
4.980435433609741580582713757
> Reg := Regulator(E); Reg;
0.1524601779431437875
> #TorsionSubgroup(E);
1
> TamagawaNumber(E,389);
1
> TamagawaNumber(M,389);        // entirely different algorithm
1
> Sha := L1/(Omega*Reg); Sha;
0.9999979295234896211
> f := pAdicLSeries(E,3); _<T> := Parent(f); f;
O(3^11) + O(3^3)*T - (1 + O(3^3))*T^2 + (2 + O(3^2))*T^3
 - (2 + O(3^2))*T^4 + (1 + O(3^2))*T^5 + O(T^6)
```

139.18 Dimension Formulas

`DimensionCuspFormsGamma0(N, k)`

The dimension of the space $S_k(\Gamma_0(N))$ of weight k cusp forms for $\Gamma_0(N)$.

`DimensionNewCuspFormsGamma0(N, k)`

The dimension of the new subspace of the space $S_k(\Gamma_0(N))$ of weight k cusp forms for $\Gamma_0(N)$.

`DimensionCuspFormsGamma1(N, k)`

The dimension of the space $S_k(\Gamma_1(N))$ of weight k cusp forms for $\Gamma_1(N)$.

`DimensionNewCuspFormsGamma1(N, k)`

The dimension of the new subspace of the space $S_k(\Gamma_1(N))$ of weight k cusp forms for $\Gamma_1(N)$.

`DimensionCuspForms(eps, k)`

The dimension of the space $S_k(\Gamma_1(N))(\varepsilon)$ of cusp forms of weight k and Dirichlet character `eps`. The level N is the modulus of `eps`. The dimension is computed using the formula of Cohen and Oesterlè (see [CO77]).

Example H139E28

```
> DimensionCuspFormsGamma0(11,2);
1
> DimensionCuspFormsGamma0(1,12);
1
> DimensionCuspFormsGamma0(5077,2);
422
> DimensionCuspFormsGamma1(5077,2);
1071460
> G := DirichletGroup(5*7);
> eps := G.1*G.2;
> IsOdd(eps);
true
> DimensionCuspForms(eps,2);
0
> DimensionCuspForms(eps,3);
6
```

The dimension of the space of cuspidal modular symbols is twice the dimension of the space of cusp forms.

```
> Dimension(CuspidalSubspace(ModularSymbols(eps,3)));
12
```

139.19 Bibliography

[**AS02**] Amod Agashe and William A. Stein. *Appendix to Joan-C. Lario and René Schoof:* Some computations with Hecke rings and deformation rings. *submitted*, 2002.

[**Bos00**] Wieb Bosma, editor. *ANTS IV*, volume 1838 of *LNCS*. Springer-Verlag, 2000.

[**BS02**] Kevin Buzzard and William A. Stein. A mod five approach to modularity of icosahedral Galois representations. *Pacific J. Math.*, 203(2):265–282, 2002.

[**CM00**] J. E. Cremona and Barry Mazur. Visualizing elements in the Shafarevich-Tate group. *Experiment. Math.*, 9(1):13–28, 2000.

[**CO77**] Henri Cohen and J. Oesterlé. *Dimensions des espaces de formes modulaires*, volume 627 of *Lecture Notes in Math.*, pages 69–78. Springer, Berlin, 1977.

[**Cre92**] J. E. Cremona. Modular symbols for $\Gamma_1(N)$ and elliptic curves with everywhere good reduction. *Math. Proc. Cambridge Philos. Soc.*, 111(2):199–218, 1992.

[**Cre97**] J. E. Cremona. *Algorithms for modular elliptic curves*. Cambridge University Press, Cambridge, second edition, 1997.

[**DI95**] Fred Diamond and Ju Im. Modular forms and modular curves. In *Seminar on Fermat's Last Theorem*, pages 39–133. Amer. Math. Soc., Providence, RI, 1995.

[**FLS$^+$01**] E. V. Flynn, F. Leprévost, E. F. Schaefer, W. A. Stein, M. Stoll, and J. L. Wetherell. Empirical evidence for the Birch and Swinnerton-Dyer conjectures for modular Jacobians of genus 2 curves. *Math. of Comp.*, 70(236):1675–1697, 2001.

[**FM99**] Gerhard Frey and Michael Muller. Arithmetic of modular curves and applications. In *Algorithmic algebra and number theory (Heidelberg, 1997)*, pages 11–48. Springer, Berlin, 1999.

[**KS00**] David R. Kohel and William A. Stein. Component Groups of Quotients of $J_0(N)$. In Bosma [Bos00].

[**Maz77**] Barry Mazur. Modular curves and the Eisenstein ideal. *Inst. Hautes Études Sci. Publ. Math.*, 47:33–186 (1978), 1977.

[**Mer93**] Loic Merel. Intersections sur des courbes modulaires. *Manuscripta Math.*, 80(3):283–289, 1993.

[**Mer94**] Loic Merel. Universal Fourier expansions of modular forms. In *On Artin's conjecture for odd 2-dimensional representations*, pages 59–94. Springer, 1994.

[**Mes86**] Jean-Francois Mestre. La méthode des graphes. Exemples et applications. *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata)*, pages 217–242, 1986.

[**PS99**] Bjorn Poonen and Michael Stoll. The Cassels-Tate pairing on polarized abelian varieties. *Ann. of Math. (2)*, 150(3):1109–1149, 1999.

[**Ste00**] William A. Stein. Explicit approaches to modular abelian varieties. *Ph.D. thesis, University of California, Berkeley*, 2000.

[**Ste01**] William A. Stein. Component Groups of Purely Toric Quotients of Semistable Jacobians. *submitted*, 2001.

[**Ste08**] William A. Stein. An introduction to computing modular forms using modular symbols. In *Algorithmic number theory: lattices, number fields, curves and cryptography*, volume 44 of *Math. Sci. Res. Inst. Publ.*, pages 641–652. Cambridge Univ. Press, Cambridge, 2008.

[**Stu87**] Jacob Sturm. On the congruence of modular forms. In *Number theory (New York, 1984–1985)*, pages 275–280. Springer, Berlin, 1987.

140 BRANDT MODULES

Chapter 140
BRANDT MODULES

140.1 Introduction

Brandt modules provide a representation in terms of quaternion ideals of certain cohomology subgroups associated to Shimura curves $X_0^D(N)$ which generalize the classical modular curves $X_0(N)$. The Brandt module datatype is that of a Hecke module – a free module of finite rank with the action of a ring of Hecke operators – which is equipped with a canonical basis (identified with left quaternion ideal classes) and an inner product which is adjoint with respect to the Hecke operators. The machinery of modular symbols, Brandt modules, and, in a future release, a module of singular elliptic curves, form the computational machinery underlying modular forms in MAGMA.

Brandt modules were implemented by David Kohel, motivated by the article of Mestre and Oesterlé [Mes86] on the method of graphs for supersingular elliptic curves, the article of Pizer [Piz80] on computing spaces of modular forms using quaternion arithmetic, and grew out of research in the author's thesis [Koh96] on endomorphism ring structure of elliptic curves over finite fields. The Brandt module machinery is described in the article [Koh01] and has been used, together with modular symbols, in the computation of component groups of quotients of the Jacobians $J_0(N)$ of classical modular curves [KS00].

140.2 Brandt Module Creation

We first describe the various constructors for Brandt modules and their elements.

`BrandtModule(D)`

`BrandtModule(D, m)`

`ComputeGrams` BOOLELT *Default :* `true`

Given a product D of an odd number of primes, and a integer m which has valuation at most one at each prime divisor p of D, return a Brandt module of level (D, m) over the integers. If not specified, then the conductor m is taken to be 1.

The parameter `ComputeGrams` can be set to `false` in order to not compute the $h \times h$ array, where h is the left class number of level (D, m), of reduced Gram matrices of the quaternion ideal norm forms. Instead the basis of quaternion ideals is stored and the collection of degree p ideal homomorphisms is then computed in order to find the Hecke operator T_p for each prime p.

For very large levels, setting `ComputeGrams` to `false` is more space efficient. For moderate sized levels for which one wants to compute many Hecke operators, it is preferable to compute the Gram matrices and determine the Hecke operators using theta series.

`BrandtModule(A)`

`BrandtModule(A, R)`

`ComputeGrams` BoolElt *Default :* `true`

Given a definite order A in a quaternion algebra over $\mathbf{Q}$, returns the Brandt module on the left ideals classes for A, as a module over R. If not specified, the ring R is taken to be the integers. The parameter `ComputeGrams` is as previously described.

`BaseExtend(M, R)`

Forms the Brandt module with coefficient ring base extended to R.

`BrandtModule(M, N)`

This constructor is an alternative to `BrandtModule(D, N)` above, and *uses a different algorithm* which is preferable in the case where N is not very small.

It constructs the Brandt module attached to an Eichler order of level N inside the maximal order M. The algorithm avoids explicitly working with the Eichler order.

Example H140E1

In the following example we create the Brandt module of level 101 over the field of 7 elements and decompose it into its invariant subspaces.

```
> A := QuaternionOrder(101);
> FF := FiniteField(7);
> M := BrandtModule(A,FF);
> Decomposition(M,13);
[
    Brandt module of level (101,1), dimension 1, and degree 9 over Finite
    field of size 7,
    Brandt module of level (101,1), dimension 1, and degree 9 over Finite
    field of size 7,
    Brandt module of level (101,1), dimension 1, and degree 9 over Finite
    field of size 7,
    Brandt module of level (101,1), dimension 6, and degree 9 over Finite
    field of size 7
]
```

We note that Brandt modules of non-prime discriminant can be useful for studying isogeny factors of modular curves, since it is possible to describe exactly the piece of cohomology of interest, without first computing a much larger space. In this example we see that the space of weight 2 cusp forms for $\Gamma_0(1491)$, where $1491 = 3 \cdot 7 \cdot 71$, is of dimension 189 (plus an Eisenstein space of dimension 7), while the newspace has dimension 71. The Jacobian of the Shimura curve $X_0^{1491}(1)$ is isogenous to the new factor of $J_0(1491)$, so that we can study the newspace directly via the Brandt module.

```
> DimensionCuspFormsGamma0(3*7*71,2);
189
> DimensionNewCuspFormsGamma0(3*7*71,2);
```

```
71
> BrandtModuleDimension(3*7*71,1);
72
> M := BrandtModule(3*7*71 : ComputeGrams := false);
> S := CuspidalSubspace(M);
> Dimension(S);
71
> [ Dimension(N) : N in Decomposition(S,13 : Sort := true) ];
[ 6, 6, 6, 6, 11, 12, 12, 12 ]
```

In this example by setting `ComputeGrams` equal to `false` we obtain the Brandt module much faster, but the decomposition is much more expensive. For most applications the default computation of Gram matrices is preferable.

140.2.1 Creation of Elements

`M ! x`

Given a sequence or module element x compatible with the Brandt module M, forms the corresponding element in M.

`M . i`

For a Brandt module M and integer i, returns the i-th basis element.

140.2.2 Operations on Elements

Brandt module elements support standard operations.

`a * x`
`x * a`

The scalar multiplication of a Brandt module element x by an element a in the base ring.

`x * T`

Given a Brandt module element x and an element T of the algebra of Hecke operators of degree compatible with the parent of x or of its ambient module, returns the image of x under T.

`x + y`

Returns the sum of two Brandt module elements.

`x - y`

Returns the difference of two Brandt module elements.

`x eq y`

Returns `true` if x and y are equal elements of the same Brandt module.

`Eltseq(x)`

Returns the sequence of coefficients of the Brandt module element x.

`InnerProduct(x, y)`

Returns the inner product of the Brandt module elements x and y with respect to the canonical pairing on their common parent.

`Norm(x)`

Returns the inner product of the Brandt module element x with itself.

140.2.3 Categories and Parent

Brandt modules belong to the category `ModBrdt`, with elements of type `ModBrdtElt`, involved in the type checking of arguments in MAGMA programming. The `Parent` of an element is the space to which it belongs.

`Category(M)` `Type(M)`
`Category(x)` `Type(x)`

The category, `ModBrdt` or `ModBrdtElt`, of the Brandt module M or of the Brandt module element x.

`Parent(x)`

The parent module M of a Brandt module element x.

`x in M`

Returns `true` if M is the parent of x.

140.2.4 Elementary Invariants

Here we describe the elementary invariants of the Brandt module, defined with respect to a definite quaternion order A in a quaternion algebra $\mathbf{H}$ over $\mathbf{Q}$. The *level* of M is defined to be the reduced discriminant of A, the *discriminant* is defined to be the discriminant of the algebra $\mathbf{H}$, and the *conductor* to be the index of A in any maximal order of $\mathbf{H}$ which contains it. We note that the discriminant of M is just the product of the ramified primes of $\mathbf{H}$, and the product of the conductor and discriminant of M is the reduced discriminant of A.

`Level(M)`

Returns the level of the Brandt module, which is the product of the discriminant and the conductor, and equal to the reduced discriminant of its defining quaternion order.

`Discriminant(M)`

Returns the discriminant of the quaternion algebra **H** with respect to which the Brandt module M is defined (equal to the product of the primes which ramify in **H**).

`Conductor(M)`

Returns the conductor or index of the defining quaternion order of the Brandt module M in a maximal order of its quaternion algebra.

`BaseRing(M)`

The ring over which the Brandt module M is defined.

`Basis(M)`

Returns the basis of the Brandt module M.

140.2.5 Associated Structures

The following give structures associated to Brandt modules. In particular we note the definition of the `AmbientModule`, which is the full module containing a given Brandt module whose basis corresponds to the left quaternion ideals. Elements of every submodule of the ambient module are displayed with respect to the basis of the ambient module.

`AmbientModule(M)`

The full module of level (D, m) containing a given module of this level.

`IsAmbient(M)`

Returns `true` if and only if the Brandt module M is its own ambient module.

`Dimension(M)`
`Rank(M)`

Returns the rank of the Brandt module M over its base ring.

`Degree(M)`

Returns the degree of the Brandt module M, defined to be the dimension of its ambient module.

`GramMatrix(M)`

The matrix $(\langle u_i, u_j \rangle)$ defined with respect to the basis $\{u_i\}$ of the Brandt module M.

`InnerProductMatrix(M)`

Returns the Gram matrix of the ambient module of the Brandt module M.

Example H140E2

The following example demonstrates the use of `AmbientModule` to get back to the original Brandt module.

```
> M := BrandtModule(3,17);
> S := CuspidalSubspace(M);
> M eq AmbientModule(M);
true
```

`Ideals(M)`

This constructs the quaternionic ideals which correspond to the basis of the Brandt module M. It is only implemented when M was constructed using `BrandtModule(M, N)` – the case where the new algorithm is used, which avoids constructing these ideals explicitly.

140.2.6 Verbose Output

The verbose level for Brandt modules is set with the command `SetVerbose("Brandt",n)`. Since the construction of a Brandt module requires intensive quaternion algebra machinery for ideal enumeration, the `Quaternion` verbose flag is also relevant. In both cases, the value of n can be 0 (silent), 1 (verbose), or 2 (very verbose).

Example H140E3

In the following example we show the verbose output from the quaternion ideal enumeration in the creation of the Brandt module of level $(37, 1)$.

```
> SetVerbose("Quaternion",2);
> BrandtModule(37);
Ideal number 1, right order module
Full RSpace of degree 4 over Integer Ring
Inner Product Matrix:
[ 2  0  1  1]
[ 0  4 -1  2]
[ 1 -1 10  0]
[ 1  2  0 20]
Frontier at 2-depth 1 has 3 elements.
Number of ideals = 1
Ideal number 2, new right order module
Full RSpace of degree 4 over Integer Ring
Inner Product Matrix:
[ 2 -1  0  1]
[-1  8 -1 -4]
[ 0 -1 10 -2]
[ 1 -4 -2 12]
Ideal number 3, new right order module
Full RSpace of degree 4 over Integer Ring
```

```
Inner Product Matrix:
[ 2  1  0 -1]
[ 1  8  1  3]
[ 0  1 10 -2]
[-1  3 -2 12]
Frontier at 2-depth 2 has 4 elements.
Number of ideals = 3
Brandt module of level (37,1), dimension 3, and degree 3 over Integer Ring
```

140.3 Subspaces and Decomposition

`EisensteinSubspace(M)`

Returns the Eisenstein subspace of the Brandt module M. When the level of M is square-free this will be the submodule generated by a vector of the form $(w/w_1, \ldots, w/w_n)$, if it exists in M, where w_i is the number of automorphisms of the i-th basis ideal and $w = \mathrm{LCM}(\{w_i\})$.

`CuspidalSubspace(M)`

Returns the cuspidal subspace, defined to be the orthogonal complement of the Eisenstein subspace of the Brandt module M. If the discriminant of M is coprime to the conductor, then the cuspidal subspace consists of the vectors in M of the form $(a_1, \ldots, a_n)$, where $\sum_i a_i = 0$.

`OrthogonalComplement(M)`

The Brandt module orthogonal to the given module M in the ambient module of M.

`M meet N`

Returns the intersection of the Brandt modules M and N.

`Decomposition(M, B)`

`Sort` BoolElt *Default :* `false`

Returns a decomposition of the Brandt module with respect to the Atkin–Lehner operators and Hecke operators up to the bound B. The parameter `Sort` can be set to `true` to return a sequence sorted under the operator `lt` as defined below.

`SortDecomposition(D)`

Sort the sequence D of spaces of Brandt modules with respect to the `lt` comparison operator.

Example H140E4

```
> M := BrandtModule(2*3*17);
> Decomp := Decomposition(M,11 : Sort := true);
> Decomp;
[
    Brandt module of level (102,1), dimension 1, and degree 4 over Integer Ring,
    Brandt module of level (102,1), dimension 1, and degree 4 over Integer Ring,
    Brandt module of level (102,1), dimension 1, and degree 4 over Integer Ring,
    Brandt module of level (102,1), dimension 1, and degree 4 over Integer Ring
]
> [ IsEisenstein(N) : N in Decomp ];
[ true, false, false, false ]
```

140.3.1 Boolean Tests on Subspaces

`IsEisenstein(M)`

Returns `true` if and only if the Brandt module M is contained in the Eisenstein subspace of the ambient module.

`IsCuspidal(M)`

Returns `true` if and only if the Brandt module M is contained in the cuspidal subspace of the ambient module.

`IsIndecomposable(M, B)`

Returns `true` if an only if the Brandt module M does not decompose into complementary Hecke-invariant submodules under the Atkin-Lehner operators, nor under the Hecke operators T_n, for $n \leq B$.

`M1 subset M2`

Returns `true` if and only if $M1$ is contained in the module $M2$.

`M1 lt M2`

`Bound` RngIntElt *Default :* 101

Given two indecomposable subspaces, M_1 and M_2, returns `true` if and only if $M_1 < M_2$ under the following ordering:

(1) Order by dimension, with smaller dimension being less.

(2) An Eisenstein subspace is less than a cuspidal subspace of the same dimension.

(3) Order by Atkin–Lehner eigenvalues, starting with *smallest* prime dividing the level and with '+' being less than '–'.

(4) Order by $|\operatorname{Tr}(T_{p^i}(M_j))|$, p not dividing the level, and $1 \leq i \leq g$, where g is `Dimension`(M_1), with the positive one being smaller in the event of equality.

Condition (4) differs from the similar one for modular symbols, but permits the comparison of arbitrary Brandt modules. The algorithm returns `false` if all primes up to value of the parameter `Bound` fail to differentiate the arguments.

`M1 gt M2`

`Bound` RngIntElt *Default :* 101

Returns the complement of `lt` for Brandt modules $M1$ and $M2$.

Example H140E5

```
> M := BrandtModule(7,7);
> E := EisensteinSubspace(M);
> Basis(E);
[
    (1 1 0 0),
    (0 0 1 1)
]
> S := CuspidalSubspace(M);
> Basis(S);
[
    ( 1 -1  0  0),
    ( 0  0  1 -1)
]
> PS<q> := LaurentSeriesRing(RationalField());
> qExpansionBasis(S,100);
[
    q + q^2 - q^4 - 3*q^8 - 3*q^9 + 4*q^11 - q^16 - 3*q^18 + 4*q^22 + 8*q^23
        - 5*q^25 + 2*q^29 + 5*q^32 + 3*q^36 - 6*q^37 - 12*q^43 - 4*q^44 +
        8*q^46 - 5*q^50 - 10*q^53 + 2*q^58 + 7*q^64 + 4*q^67 + 16*q^71 +
        9*q^72 - 6*q^74 + 8*q^79 + 9*q^81 - 12*q^86 - 12*q^88 - 8*q^92 -
        12*q^99 + 5*q^100 + O(q^101)
]
```

140.4 Hecke Operators

A Brandt module M is equipped with a family of linear Hecke operators which act on it. These are returned as matrices, which act on the right with respect to the basis `Basis(M)`, but the Hecke operators of the ambient module may also be applied to elements of submodules. The system of Hecke operators are computed by default using the theta series which define the classical Brandt matrices. If a module is created with the `ComputeGram` parameter set to false, the Hecke operators are determined by means of enumeration of ideals in a p-neighbouring operation analogous to the method of graphs approach of Mestre and Oesterlé [Mes86].

`HeckeOperator(M, n)`

Compute a matrix representing the nth Hecke operator T_n with respect to `Basis(M)` for the Brandt module M.

`AtkinLehnerOperator(M, p)`

Computes the Atkin–Lehner operator on the Brandt module M, where p is a prime dividing the discriminant of M.

140.5 q-Expansions

We can associate a theta series to any pair of elements of a Brandt module, which give embeddings (with respect to any fixed module element) of the Brandt module in a space of weight 2 modular forms.

`ThetaSeries(x, y, prec)`

Returns the theta series associated to the pair (x, y) of elements of a Brandt module, as an element of a power series ring.

`qExpansionBasis(M, prec)`

A sequence of power series elements, to precision *prec*, spanning the image of the theta functions associated to pairs in the Brandt module M.

140.6 Dimensions of Spaces

`BrandtModuleDimension(D, N)`

The dimension of the Brandt module of level (D, N), computed by means of standard formulas.

Example H140E6

In the following example we demonstrate the computation of the dimensions of the Brandt modules for the sequence of Eichler orders of index 3^i in a maximal order in the quaternion algebra of discriminant $2 \cdot 3 \cdot 7$.

```
> D := 2*5*7;
> for i in [0..10] do
>     BrandtModuleDimension(D,3^i);
> end for;
2
8
24
72
216
648
1944
5832
17496
52488
157464
```

140.7 Brandt Modules Over $F_q[t]$

This section concerns the case of quaternion orders whose base ring is $F_q[t]$. The definitions and constructions are similar to the case of quaternion orders over the integers. The implementation follows the new implementation over the integers (avoiding working explicitly in terms of ideals in Eichler orders), and makes use of techniques for quadratic forms over $F_q[t]$ (developed by Markus Kirschmer).

The following intrinsics are provided. Where no description is given, the arguments and return values are similar to the corresponding intrinsics over the integers.

`BrandtModuleDimension(D, N)`

`BrandtModuleDimensionOfNewSubspace(D, N)`

`BrandtModule(M, N)`

This constructs the Brandt module attached to an Eichler order of level N in the maximal order M.

`QuaternionOrder(M)` `Level(M)`

`Discriminant(M)` `Conductor(M)` `Ideals(M)`

`InnerProductMatrix(M)` `HeckeOperator(M, n)`

`HeckeEigenvectors(M)`

This returns the common eigenvectors for the Hecke operators on the Brandt module M, as elements of M.

`HeckeEigenvalue(f, p)`

For a Hecke eigenform f in a Brandt module, this returns the eigenvalue for the Hecke operator at the prime p.

140.8 Bibliography

[**Bos00**] Wieb Bosma, editor. *ANTS IV*, volume 1838 of *LNCS*. Springer-Verlag, 2000.

[**Koh96**] D. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California, Berkeley, 1996.

[**Koh01**] D. Kohel. Hecke module structure of quaternions. In K. Miyake, editor, *Class Field Theory – its Centenary and Prospect*, 2001.

[**KS00**] D. Kohel and W. Stein. Component groups of quotients of $J_0(N)$. In Bosma [Bos00].

[**Mes86**] J.-F. Mestre. Sur la méthode des graphes, Exemples et applications. In *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields*, pages 217–242. Nagoya University, 1986.

[**Piz80**] A. Pizer. An Algorithm for Computing Modular Forms on $\Gamma_0(N)$. *Journal of Algebra*, 64:340–390, 1980.

141 SUPERSINGULAR DIVISORS ON MODULAR CURVES

Chapter 141
SUPERSINGULAR DIVISORS ON MODULAR CURVES

141.1 Introduction

This chapter is about how to use MAGMA to compute with the Hecke module $D(N,p)$ of divisors on the supersingular points on $X_0(N)$ in characteristic p.

Let p be a prime. A divisor on the supersingular points of $X_0(1)$ in characteristic p is a finite formal linear combination of supersingular j-invariants $j \in \mathbf{F}_{p^2}$. More generally, suppose N is an integer that is not divisible by p. The module $D(N,p)$ of divisors on the supersingular points of $X_0(N)$ in characteristic p is the free abelian group generated by isomorphism classes of pairs (E,C) where E is a supersingular elliptic curve over $\mathbf{F}_{p^2}$ and C is a cyclic subgroup of E of order N. (We call such a pair (E,C) an elliptic curve enhanced with level N structure.) The abelian group $D(N,p)$ of divisors is equipped in a natural way with an action of Hecke operators T_n.

The module of supersingular points is a special case of the Brandt module construction (see Chapter 140) because of the following equivalence between objects:

- pairs (E,C) as above,
- isomorphism classes of left ideals of an Eichler order of level N in the quaternion algebra over $\mathbf{Q}$ ramified at, and p and ∞,
- supersingular points on $X_0(N)/\mathbf{F}_p$ over $\overline{\mathbf{F}}_p$.

The supersingular points of $X_0(N)/\mathbf{F}_p$ correspond to singularities of the special fiber of a minimal model of $X_0(Np)$ at p. This special fiber, $X_0(Np)/\mathbf{F}_p$, is isomorphic to two copies of $X_0(N)/\mathbf{F}_p$ joined at the supersingular points as simple double points. The structure of the multiplicative part of the Jacobian $J_0(Np)$ at p is captured by the behavior of the supersingular points of $X_0(N)/\mathbf{F}_p$ (see Grothendieck [Gro72] and Deligne-Rapoport [DR73]). The system of Hecke operators on the supersingular divisor group gives a representation of the p-new subspace of modular forms of level Np. We therefore refer to the *level* of the supersingular module as Np, and use the term *auxiliary level* to refer to N.

There are many reasons why one might be interested in computing with $D(N,p)$. Foremost, $D(N,p)$ is isomorphic as a module over the Hecke algebra to a subspace of the modular forms for $\Gamma_0(N)$ of weight 2 and level Np (more precisely, $D(N,p)\otimes\mathbf{C}$ is isomorphic to the p-new subspace of $M_2(\Gamma_0(Np))$). If $N = 1,2,3,5,7,13$, then MAGMA computes $D(N,p)$ using the highly-efficient method of graphs, which for small q quickly produce very sparse matrices that represent Hecke operators T_q. There are also formulas of Gross, Kudla, Merel, and others that involve the explicit representation of an eigenform in terms of a

basis of supersingular j-invariants, and Stein has a formula, which involves $D(N,p)$, for orders of component groups of modular abelian varieties (see the `ComponentGroupOrder` command in the modular symbols package).

MAGMA computes the module of supersingular divisors using either the method of graphs of Mestre–Oesterlé when $N = 1, 2, 3, 5, 7, 13$ or Brandt modules in general. When it is applicable, the method of graphs is much faster (in MAGMA) than Brandt modules, but the Brandt modules method works in general.

The modular curve approach computes correspondences on the modular curves $X_0(N)$ by means of pre-computed models for the system of covering maps $X_0(N\ell) \to X_0(N)$. Such correspondences give rise to the Hecke operators T_ℓ as the adjacency matrices of the graphs of ℓ-isogenies of the basis of $D(N,p)$. For the alternative approach through Brandt modules, the reader should consult Chapter 140 and the articles of Pizer [Piz80] and Kohel [Koh01].

This package still has some unnecessary limitations. When using Brandt modules to compute the module of supersingular divisor in MAGMA, no facility is currently provided for describing the divisors in terms of supersingular elliptic curves. Also, it is currently only possible to work with the module of supersingular divisors over the integers.

141.1.1 Categories

Modules of supersingular points belong to the category `ModSS`, and the elements of these modules belong to `ModSSElt`.

141.1.2 Verbose Output

To set the verbosity level use the command `SetVerbose("SupersingularModule",n)`, where `n` is 0 (silent), 1 (verbose), or 2 (very verbose).

141.2 Creation Functions

141.2.1 Ambient Spaces

An ambient supersingular divisors module is specified by giving an integer N (the level) and a prime p (the characteristic).

`SupersingularModule(p,N : parameters)`

`Brandt` BOOLELT *Default :* `false`

The module M of supersingular points on $X_0(N)$ over $\overline{\mathbf{F}}_p$. Equivalently, this is the free abelian group on the supersingular elliptic curves in characteristic p enhanced with level N structure. We require that N and p are coprime.

`SupersingularModule(p)`

The Hecke module M of divisors of degree 0 on the supersingular points on $X_0(1)$ over $\overline{\mathbf{F}}_p$. Equivalently, this is the free abelian group on the supersingular j-invariants in characteristic p.

Example H141E1

```
> SupersingularModule(11);
Supersingular module associated to X_0(1)/GF(11) of dimension 2
> SupersingularModule(11,3);
Supersingular module associated to X_0(3)/GF(11) of dimension 4
> SupersingularModule(3,11);
Supersingular module associated to X_0(11)/GF(3) of dimension 2
```

The optional parameter `Brandt` forces computation of the supersingular module using quaternion arithmetic, even if this will be slower. (It's not clear why anyone would want to do use this parameter except to compute the same thing using two different algorithms.)

```
> SupersingularModule(97);
Supersingular module associated to X_0(1)/GF(97) of dimension 8
> SupersingularModule(97 : Brandt := true);
Supersingular module associated to X_0(1)/GF(97) of dimension 8
```

141.2.2 Elements

`M . i`

The ith basis element of the module M.

`M ! x`

The coercion of x into the module M.

Example H141E2

First we create the supersingular module attached to $p = 11$, $N = 3$. This is the free abelian group generated by the supersingular points on $X_0(3)$ in characteristic 11, equipped with the structure of module over the Hecke algebra.

```
> X := SupersingularModule(11,3);
> P := X.1;
> P;
(5, 5)
> Eltseq(P);
[ 1, 0, 0, 0 ]
> X![ 1, 0, 0, 0 ];
(5, 5)
```

Note that the module associated to $p = 3$, $N = 11$ is computed using Brandt matrices (since $X_0(11)$ has positive genus), so elements are printed in a less informative way.

```
> Z := SupersingularModule(3,11); Z;
Supersingular module associated to X_0(11)/GF(3) of dimension 2
> P := Z.1;
> P;
```

```
[E1]
> Eltseq(P);
[ 1, 0 ]
> Z![1,0];
[E1]
```

141.2.3 Subspaces

`CuspidalSubspace(M)`

The cuspidal submodule X of M. Thus X is the submodule of divisors of degree 0 on the supersingular points. It is "cuspidal" in the sense that $X \otimes \mathbf{Q}$ is isomorphic as a Hecke module to the space $S_2(\Gamma_0(Np); \mathbf{Q})^{p\text{-new}}$ of p-new cuspforms with Fourier coefficients in $\mathbf{Q}$. Explicitly, the cuspidal subspace is the subspace of elements such that the sum of the coefficients is 0 (i.e., the subspace of divisors of degree 0).

`EisensteinSubspace(M)`

The Eisenstein submodule of M, i.e., the orthogonal complement of the cuspidal subspace of M with respect to the monodromy pairing.

`OrthogonalComplement(M)`

The orthogonal complement of the module M in the ambient space with respect to the monodromy pairing.

`Kernel(I, M)`

The kernel of I on the module M. This is the subspace of M obtained by intersecting the kernels of the operators $f_n(T_{p_n})$, where I is a sequence $[\langle p_1, f_1(x)\rangle, ..., \langle p_n, f_n(x)\rangle]$ of pairs consisting of a prime number and a polynomial.

`Decomposition(M, n)`

Decomposition of the module M with respect to the Hecke operators $T_1, T_2, \ldots, T_n$.

Example H141E3

We compute bases for the cuspidal and eisenstein subspaces when $p = 11$ and $N = 1$.

```
> M := SupersingularModule(11); Basis(M);
[
    (1, 1),
    (0, 0)
]
> S := CuspidalSubspace(M);
> E := EisensteinSubspace(M);
> Basis(S);
[
    (1, 1) - (0, 0)
```

```
]
> Basis(E);
[
    3*(1, 1) + 2*(0, 0)
]
```

Next we compute the orthogonal complement of each subspace.

```
> Basis(OrthogonalComplement(E));
[
    (1, 1) - (0, 0)
]
> Basis(OrthogonalComplement(S));
[
    3*(1, 1) + 2*(0, 0)
]
> S eq OrthogonalComplement(E);
true
```

Note that the Hecke operator T_2 acts as -2 on the cuspidal subspace. Using the `Kernel` command, we compute the subspace of M on which T_2 acts as -2, and recover the cuspidal subspace.

```
> R<x> := PolynomialRing(Integers());
> I := [<2, x + 2>];
> K := Kernel(I,M);
> Basis(K);
[
    (1, 1) - (0, 0)
]
```

We can also compute the decomposition of M into submodules for the action of the first few Hecke operators (typically a few Hecke operators are enough to give a complete decomposition with respect to all Hecke operators).

```
> Decomposition(M,5);
[
    Supersingular module associated to X_0(1)/GF(11) of dimension 1,
    Supersingular module associated to X_0(1)/GF(11) of dimension 1
]
```

141.3 Basis

`Basis(M)`

A basis for M as a $\mathbf{Z}$-module.

141.4 Properties

`AuxiliaryLevel(M)`

The level of the module M, where the auxiliary level of `SupersingularModule(N,p)` is, by definition, N.

`BaseRing(M)`

The base ring of the module M. (Currently this is always $\mathbf{Z}$.)

`Degree(P)`

The sum of the coefficients of the module element P, where P is written with respect the basis of the ambient space of the parent of M.

`Dimension(M)`

The dimension of the module M.

`Eltseq(P)`

A sequence of integers that defines the module element P.

`Level(M)`

The level of the module M, where the level of `SupersingularModule(N,p)` is, by definition, Np.

`ModularEquation(M)`

The equation of $X_0(N)$ that we use when using the Mestre method to compute with the module M of supersingular points.

`Prime(M)`

The prime of the module M, where the prime of `SupersingularModule(N,p)` is, by definition, p.

Example H141E4

```
> M := SupersingularModule(3,11);
> AuxiliaryLevel(M);
11
> BaseRing(M);
Integer Ring
> Degree(M.1+7*M.2);
8
> Dimension(M);
2
> Eltseq(M.1+7*M.2);
[ 1, 7 ]
> Level(M);
```

```
33
> Prime(M);
3
> M := SupersingularModule(11,3); M;
Supersingular module associated to X_0(3)/GF(11) of dimension 4
> ModularEquation(M);
x*y + 8
```

141.5 Associated Spaces

`BrandtModule(M)`

The Brandt module associated to the supersingular module M.

`ModularSymbols(M : parameters)`

`Proof` BoolElt *Default :* `true`

The space of modular symbols corresponding to the supersingular module M.

`ModularSymbols(M, sign : parameters)`

`Proof` BoolElt *Default :* `true`

The $+1$ or -1 quotient of the space of modular symbols corresponding to the supersingular module M.

`RSpace(M)`

The $\mathbf{Z}$-module V underlying the supersingular module M along with an invertible map $V \to M$.

Example H141E5

We compute the Brandt module and modular symbols spaces associated to the supersingular module for $p = 3$, $N = 11$, and verify that T_2 has the same characteristic polynomial on each.

```
> M := SupersingularModule(3,11);
> B := BrandtModule(M); B;
Brandt module of level (3,11), dimension 2, and degree 2 over
Integer Ring
> MS := ModularSymbols(M); MS;
Modular symbols space for Gamma_0(33) of weight 2 and dimension 4
over Rational Field
> Factorization(CharacteristicPolynomial(HeckeOperator(B, 2)));
[
    <$.1 - 3, 1>,
    <$.1 - 1, 1>
]
> Factorization(CharacteristicPolynomial(HeckeOperator(MS, 2)));
```

```
[
    <$.1 - 3, 2>,
    <$.1 - 1, 2>
]
```

There is an associated Brandt module even if the underlying computations on M are done using the Mestre-Oesterle graph method.

```
> M := SupersingularModule(11);
> UsesMestre(M);
true
> B := BrandtModule(M); B;   // takes a while
Brandt module of level (11,1), dimension 2, and degree 2 over
Integer Ring
```

141.6 Predicates

`IsAmbientSpace(M)`

Returns `true` if the supersingular module M is the full module of supersingular points and not a submodule.

`M1 eq M2`

Returns `true` if the supersingular module M_1 equals M_2.

`P eq Q`

Returns `true` if the module element P equals Q.

`M1 subset M2`

Returns `true` if the supersingular module M_1 is a subset of M_2.

`UsesBrandt(M)`

Returns `true` if the underlying computations on the supersingular module M are done using Brandt modules.

`UsesMestre(M)`

Returns `true` if the underlying computations on the supersingular module M are done using the Mestre-Oesterle method of graphs.

Example H141E6

In this example we illustrate each of the above predicates.

```
> M := SupersingularModule(11);
> S := CuspidalSubspace(M);
> IsAmbientSpace(S);
false
> IsAmbientSpace(M);
true
> S eq M;
false
> S eq S;
true
> S.1 eq S.1;
true
> S.1 eq M.1 - M.2;
true
> S.1 eq M.1;
false
> S subset M;
true
> UsesBrandt(S);
false
> UsesMestre(S);
true
> M := SupersingularModule(11 : Brandt := true);
> UsesBrandt(M);
true
> UsesMestre(M);
false
```

141.7 Arithmetic

The standard arithmetic operations + and − and left scalar multiplication are defined for elements of supersingular modules. Also one can add and intersect two submodules of an ambient supersingular module.

`P + Q` `P - Q` `a * P`

`M1 + M2`

The submodule generated by all sums of elements in the supersingular modules M_1 and M_2.

`M1 meet M2`

The intersection of the supersingular modules M_1 and M_2.

Example H141E7

First we illustrate some arithmetic on elements.

```
> M := SupersingularModule(11);
> P := M.1; P;
(1, 1)
> Q := M.2; Q;
(0, 0)
> P + Q;
(1, 1) + (0, 0)
> P - Q;
(1, 1) - (0, 0)
> 3*P;
3*(1, 1)
```

Next we illustrate some arithmetic on submodules.

```
> E := EisensteinSubspace(M);
> S := CuspidalSubspace(M);
> V := E + S;
> V;
Supersingular module associated to X_0(1)/GF(11) of dimension 2
> Basis(V);
[
    (1, 1) + 4*(0, 0),
    5*(0, 0)
]
```

The index of $E + S$ in M is of interest since it is related to congruences between Eisenstein series and cusp forms. Upon converting each of E and S to an `RSpace`, we find that the index is 5.

```
> RSpace(M)/RSpace(V);
Full Quotient RSpace of degree 1 over Integer Ring
Column moduli:
[ 5 ]
```

The intersection of E and S is the zero module.

```
> W := E meet S; W;
Supersingular module associated to X_0(1)/GF(11) of dimension 0
> Basis(W);
[]
```

141.8 Operators

`HeckeOperator(M, n)`

Compute a matrix representing the nth Hecke operator T_n with respect to `Basis(M)` for the supersingular module M.

`AtkinLehnerOperator(M, q)`

A matrix representing the Atkin-Lehner involution W_q on the supersingular module M. The number q must equal either `Prime(M)` or `AuxiliaryLevel(M)`.

Example H141E8

In this example we observe that T_2 and W_3 have the same characteristic polynomial on $S_2(\Gamma_0(33))$ as on the cuspidal subspace of the supersingular module with $p = 11$, $N = 3$.

```
> SS := CuspidalSubspace(SupersingularModule(11, 3));
> MF := CuspForms(33, 2);
> Factorization(CharacteristicPolynomial(HeckeOperator(SS, 2)));
[
    <$.1 - 1, 1>,
    <$.1 + 2, 2>
]
> Factorization(CharacteristicPolynomial(HeckeOperator(MF, 2)));
[
    <$.1 - 1, 1>,
    <$.1 + 2, 2>
]
> Factorization(CharacteristicPolynomial(AtkinLehnerOperator(SS, 3)));
[
    <$.1 - 1, 1>,
    <$.1 + 1, 2>
]
> Factorization(CharacteristicPolynomial(AtkinLehnerOperator(MF, 3)));
[
    <$.1 - 1, 2>,
    <$.1 + 1, 1>
]
```

The supersingular module with $p = 3$ and $N = 11$ is isomorphic as a module to the subspace of 3-new cuspforms in $S_2(\Gamma_0(33))$.

```
> SS := CuspidalSubspace(SupersingularModule(3, 11));
> MF := NewSubspace(CuspForms(33,2), 3);
> HeckeOperator(SS, 17);
[-2]
> HeckeOperator(MF, 17);
[-2]
> AtkinLehnerOperator(SS, 11);
[-1]
```

```
> AtkinLehnerOperator(MF, 11);
[-1]
```

141.9 The Monodromy Pairing

The monodromy pairing is a nondegenerate Hecke equivariant integer valued pairing on the module of supersingular points. (Note that it is not perfect, in general.) This pairing is simple to describe in terms of the basis for the supersingular module given by the enhanced supersingular elliptic curves. If E and F are two supersingular elliptic curve in characteristic p equipped with level N structure, then E and F pair to 0 unless $E \cong F$, in which case they pair to half the number of automorphisms of E.

`MonodromyPairing(P, Q)`

The monodromy pairing of elements P and Q of a supersingular module.

`MonodromyWeights(M)`

The diagonal entries that define the monodromy pairing on the ambient space.

Example H141E9

We compute the Brandt module and modular symbols spaces associated to the supersingular module for $p = 3$, $N = 11$, and verify that T_2 acts in a compatible way on them.

```
> M := SupersingularModule(11);
> MonodromyWeights(M);
[ 2, 3 ]
> P := Basis(CuspidalSubspace(M))[1]; P;
(1, 1) - (0, 0)
> Q := Basis(EisensteinSubspace(M))[1]; Q;
3*(1, 1) + 2*(0, 0)
> Basis(M);
[
    (1, 1),
    (0, 0)
]
> MonodromyPairing(P,Q);
0
> MonodromyPairing(P,P);
5
```

141.10 Bibliography

[**DR73**] P. Deligne and M. Rapoport. Les schémas de modules de courbes elliptiques. In *Lecture Notes in Mathematics*, volume 349, pages 143–316. Springer-Verlag, 1973.

[**Gro72**] A. Grothendieck. *Groupes de monodromie en géométrie algébrique. I.* Springer-Verlag, Berlin, 1972. Séminaire de Géométrie Algébrique du Bois-Marie 1967–1969 (SGA 7 I), Dirigé par A. Grothendieck. Avec la collaboration de M. Raynaud et D. S. Rim, Lecture Notes in Mathematics, Vol. 288.

[**Koh01**] D. Kohel. Hecke module structure of quaternions. In K. Miyake, editor, *Class Field Theory – its Centenary and Prospect*, 2001.

[**Piz80**] A. Pizer. An Algorithm for Computing Modular Forms on $\Gamma_0(N)$. *Journal of Algebra*, 64:340–390, 1980.

142 MODULAR ABELIAN VARIETIES

Chapter 142
MODULAR ABELIAN VARIETIES

142.1 Introduction

This chapter is the reference for the modular abelian varieties package in MAGMA. A *modular abelian variety* is an abelian variety that is a quotient of the modular Jacobian $J_1(N)$, for some integer N. This package provides extensive functionality for computing with such abelian varieties, including functions for enumerating and decomposing modular abelian varieties, isomorphism testing, computing exact endomorphism and homomorphism rings, doing arithmetic with finite subgroups and computing information about torsion subgroups, special values of L-functions and Tamagawa numbers.

Essentially none of the algorithms in this package use explicit defining equations for varieties, and as such work in a great degree of generality. For example, many even make sense for Grothendieck motives attached to modular forms, and we have included the corresponding functionality, when it makes sense.

MAGMA V2.11 was the first release of the modular abelian varieties package. The major drawback of the current version is that complete decomposition into simples is only implemented over the rational numbers. Thus the interesting behavior over number fields, involving extra inner twists, which leads to $\mathbf{Q}$-curves and associated questions, is not available (much of it is implemented, but there are some fundamental theoretical obstructions to overcome).

Our philosophy for representing modular abelian varieties is perhaps different than what you might expect, so we describe how we view an abelian subvariety A over $\mathbf{Q}$ contained in the modular Jacobian $J_0(N)$. By the Abel-Jacobi theorem we may view $J_0(N)$ over the complex numbers as a complex vector space V modulo the lattice $H_1(J_0(N), \mathbf{Z}) = H_1(X_0(N), \mathbf{Z})$. An abelian subvariety $A \subset J_0(N)$ and the map $i : A \to J_0(N)$ is completely determined by giving the image of $H_1(A, \mathbf{Q})$ in the vector space $H_1(X_0(N), \mathbf{Q})$. At this point, it might appear that we have to compute lots of floating point numbers and approximate lattices in the complex numbers, but this is not the case. Instead, we use modular symbols to compute $H_1(X_0(N), \mathbf{Z})$ as an abstract abelian group, and use everything we can from the extensive theory of modular forms to compute things about the abelian varieties determined by subgroups of $H_1(X_0(N), \mathbf{Z})$ and other related abelian varieties. Note that even though we work with homology, which is associated to complex tori, the abelian variety A over $\mathbf{Q}$ is still determined by our defining data (a certain subgroup of $H_1(X_0(N), \mathbf{Z})$), and our algorithms can often take advantage of this.

142.1.1 Categories

Modular abelian varieties belong to the category `ModAbVar`, and the elements of modular abelian varieties belong to `ModAbVarElt`. The category `MapModAbVar` consists of homomorphisms between modular abelian varieties (sometimes only up to isogeny, i.e., with a denominator). Spaces of homomorphisms between modular abelian varieties form the category `HomModAbVar`. Finitely generated subgroups of modular abelian varieties form the category `ModAbVarSubGrp`. Homology of a modular abelian variety is in the category `ModAbVarHomol`. The L-series of modular abelian varieties are in `ModAbVarLSer`.

Example H142E1

We create an object of each category.

```
> A := JZero(11);
> Type(A);
ModAbVar
> Type(A!0);
ModAbVarElt
> Type(nIsogeny(A,2));
MapModAbVar
> Type(nTorsionSubgroup(A,2));
ModAbVarSubGrp
> Type(End(A));
HomModAbVar
> Type(Homology(A));
ModAbVarHomol
> Type(LSeries(A));
ModAbVarLSer
```

142.1.2 Verbose Output

The verbosity level is set using the command `SetVerbose("ModAbVar",n)`, where `n` is 0 (silent), 1 (verbose), or 2 (very verbose). The default verbose level is 0.

Two additional verbose levels are included in this package. Level 3 is exactly like level 1, except instead of displaying to the screen, verbose output is appended to the file `ModAbVar-verbose.log` in the directory that MAGMA was run from. Verbose level 4 is exactly like level 2, except verbose output is appended to `ModAbVar-verbose.log`. On a UNIX-like system, use the shell command `tail -f ModAbVar-verbose.log` to watch the verbose log in another terminal.

Example H142E2

Using `SetVerbose`, we get some information about what is happening during computations.

```
> SetVerbose("ModAbVar",1);   // some verbose output
> SetVerbose("ModAbVar",2);   // tons of verbose output
> SetVerbose("ModAbVar",3);   // some verbose output to ModAbVar-verbose.log
> SetVerbose("ModAbVar",4);   // tons of verbose output to ModAbVar-verbose.log
```

142.2 Creation and Basic Functions

The functions described below create modular abelian varieties, combine them together in various ways, and obtain simple information about them.

Modular abelian varieties are much less restricted than spaces of modular symbols as one can take arbitrary finite direct sums.

142.2.1 Creating the Modular Jacobian $J_0(N)$

The `JZero` command will create the Jacobian $J_0(N)$ of the modular curve $X_0(N)$ (which parameterizes isomorphism classes of pairs of elliptic curves and cyclic subgroups of order N). Higher weight motivic analogues of this Jacobian can be created. Computations can be carried out in the $+1$ or -1 quotient of homology for efficiency, though certain results will be off by factors of 2.

`JZero(N : parameters)`

`JZero(N, k : parameters)`

`Sign` RngIntElt *Default :* 0

Create the modular abelian variety $J_0(N)$ of level N and weight 2 or weight $k \geq 2$, i.e., the Jacobian of the modular curve $X_0(N)$. The parameter `Sign` determines whether computations will be carried out in the $+1$ or -1 quotient of homology.

Example H142E3

```
> JZero(23);
Modular abelian variety JZero(23) of dimension 2 and level 23 over Q
> JZero(23 : Sign := +1);
Modular abelian variety JZero(23) of dimension 2 and level 23 over Q
with sign 1
> JZero(23,4);
Modular motive JZero(23,4) of dimension 5 and level 23 over Q
> JZero(23,4 : Sign := -1);
Modular motive JZero(23,4) of dimension 5 and level 23 over Q with
sign -1
> JZero(389,2 : Sign := +1);
Modular abelian variety JZero(389) of dimension 32 and level 389
over Q with sign 1
```

142.2.2 Creating the Modular Jacobians $J_1(N)$ and $J_H(N)$

The `JOne` command creates the Jacobian of the modular curve $X_1(N)$ (which parameterizes isomorphism classes of pairs of elliptic curves and cyclic subgroups of order N). The command `Js` creates an abelian variety isogenous to $J_1(N)$; more precisely it is the product of abelian varieties $J_\varepsilon(N)$, where $J_\varepsilon(N)$ is the abelian variety attached to all modular forms whose character is a Galois conjugate of ε. Creating $J_s(N)$ is much faster than creating $J_1(N)$, since less time is spent finding the integral structure on homology.

The `JH` command creates the Jacobian $J_H(N)$ of the curve $X_H(N)$, which is the quotient of $X_1(N)$ by the subgroup H of the integers modulo N.

`JOne(N : parameters)`

`JOne(N, k : parameters)`

`Sign` RngIntElt *Default :* 0

Create the modular abelian variety $J_1(N)$ of level N and weight k, if given, or 2, i.e., the Jacobian of the modular curve $X_1(N)$. Note that creating and finding the integral structure on $J_s(N)$, which is isogenous to $J_1(N)$, is much faster. Computing with $J_1(N)$ may be expensive.

`Js(N : parameters)`

`Js(N, k : parameters)`

`Sign` RngIntElt *Default :* 0

A modular abelian variety that is $\mathbf{Q}$-isogenous to the weight k (or 2) version of $J_1(N)$. More precisely, the direct sum of the modular abelian varieties attached to modular symbols spaces with Nebentypus.

`JH(N, d : parameters)`

`JH(N, k, d : parameters)`

`Sign` RngIntElt *Default :* 0

Suppose H is some (cyclic) subgroup of $G = (\mathbf{Z}/N\mathbf{Z})^*$ such that G/H has order d. Create the modular abelian variety $J_H(N)$ of level N and weight k (or 2 if k not given), where $J_H(N)$ is *isogenous* to the Jacobian of the modular curve $X_H(N)$ associated to the subgroup of $\mathrm{SL}_2(\mathbf{Z})$ of matrices $[a, b; c, d]$ with c divisible by N and a in H modulo N. It is the product of modular symbols varieties $J(\epsilon)$ for all Dirichlet characters ϵ that are trivial on H.

The parameter `Sign` determines whether computations will be carried out in the $+1$ or -1 quotient of homology.

JH(N, gens : *parameters*)

JH(N, k, gens : *parameters*)

Sign RngIntElt *Default :* 0

Let H be the subgroup of $(\mathbf{Z}/N\mathbf{Z})^*$ generated by the sequence of integers, *gens*. Create the modular abelian variety $J_H(N)$ of level N and weight k (or 2 if k not given), where $J_H(N)$ is isogenous to the Jacobian of the modular curve $X_H(N)$ associated to the subgroup of $\mathrm{SL}_2(\mathbf{Z})$ of matrices $[a, b; c, d]$ with c divisible by N and a in H modulo N. It is the product of modular symbols variety $J(\epsilon)$ for all Dirichlet characters ϵ that are trivial on H.

The parameter Sign determines whether computations will be carried out in the $+1$ or -1 quotient of homology.

Example H142E4

```
> JOne(13);
Modular abelian variety JOne(13) of dimension 2 and level 13 over Q
> JOne(13,4);
Modular motive JOne(13,4) of dimension 15 and level 13 over Q
> JOne(13,4 : Sign := 1);
Modular motive JOne(13,4) of dimension 15 and level 13 over Q
> JH(13,6);
Modular abelian variety J_H(13) of dimension 2 and level 13 over Q
> JH(13,3);
Modular abelian variety J_H(13) of dimension 0 and level 13 over Q
> JH(13,[-1]);
Modular abelian variety J_H(13) of dimension 2 and level 13 over Q
> JOne(17);
Modular abelian variety JOne(17) of dimension 5 and level 17 over Q
> Js(17);
Modular abelian variety Js(17) of dimension 5 and level 17 over Q
> IsIsogenous(JOne(17),Js(17));
true
> Degree(NaturalMap(JOne(17),Js(17)));
16
> JH(17,2);
Modular abelian variety J_H(17) of dimension 1 and level 17 over Q
> JH(17,4);
Modular abelian variety J_H(17) of dimension 1 and level 17 over Q
> JH(17,8);
Modular abelian variety J_H(17) of dimension 5 and level 17 over Q
```

142.2.3 Abelian Varieties Attached to Modular Forms

The following commands create abelian varieties attached to spaces of modular forms, sequences of spaces of forms, newforms, and characters. If an input space of modular forms is not cuspidal, MAGMA automatically replaces it with its cuspidal subspace.

`ModularAbelianVariety(M : parameters)`

`Sign` RNGINTELT *Default :* 0

The abelian variety attached to the modular forms space M.

`ModularAbelianVariety(X : parameters)`

`Sign` RNGINTELT *Default :* 0

The abelian variety attached to the sequence X of modular forms spaces. This is the direct sum of the spaces attached to each element of the sequence.

`ModularAbelianVariety(eps : parameters)`

`ModularAbelianVariety(eps, k : parameters)`

`Sign` RNGINTELT *Default :* 0

The abelian variety associated to the dirichlet character ϵ. This corresponds to the space of modular forms of weight k and character any Galois conjugate of ϵ. We include all Galois conjugates in order to obtain an abelian variety that is defined over $\mathbf{Q}$.

`ModularAbelianVariety(f)`

The abelian variety attached to the newform f.

`Newform(A)`

A newform f such that the modular abelian variety A is isogenous to the newform abelian variety A_f. An error occurs if A is not attached to a newform.

Example H142E5

We first create the modular abelian variety attached to the spaces $S_2(\Gamma_0(11))$ and $S_2(\Gamma_1(13))$. This is the direct sum of $J_0(11)$ with $J_1(13)$.

```
> X := [ModularForms(11,2), ModularForms(Gamma1(13),2)];
> A := ModularAbelianVariety(X); A;
Modular abelian variety of dimension 3 and level 11*13 over Q
> IsIsomorphic(A, JZero(11)*JOne(13));
true Homomorphism N(1) from modular abelian variety of dimension
3 to JZero(11) x JOne(13) (not printing 6x6 matrix)
```

Next we create the modular abelian variety A attached to $S_2(\Gamma_1(17))$ along with $J_1(17)$ and $J_s(17)$. We then note that A is isomorphic to $J_1(17)$, but there is no reason that A should be isomorphic to $J_s(17)$ (they are probably only isogenous). This example illustrates the fact that the abelian variety computed by MAGMA attached to $S_2(\Gamma_1(17))$ is $J_1(17)$ rather than $J_s(17)$.

(Recall that $J_s(N)$ is a product of copies of abelian varieties corresponding to conjugacy classes of characters.)

```
> A := ModularAbelianVariety(ModularForms(Gamma1(17),2)); A;
Modular abelian variety of dimension 5 and level 17 over Q
> B := JOne(17); B;
Modular abelian variety JOne(17) of dimension 5 and level 17 over Q
> C := Js(17); C;
Modular abelian variety Js(17) of dimension 5 and level 17 over Q
> IsIsomorphic(A,B);
true Homomorphism from modular abelian variety of dimension 5 to
JOne(17) (not printing 10x10 matrix)
> Degree(NaturalMap(A,C));
16
```

Example H142E6

If ε is a Dirichlet character and $k \geq 2$ is an integer, let S be the space of modular forms with weight k and character a Galois conjugate of ε. The command `ModularAbelianVariety(eps,k)` computes the modular abelian variety attached to S.

```
> G<eps> := DirichletGroup(13,CyclotomicField(12));
> Order(eps^2);
6
> ModularAbelianVariety(eps^2);
Modular abelian variety of dimension 2 and level 13 over Q
> ModularAbelianVariety(eps,3);
Modular motive of dimension 4 and level 13 over Q
```

Next we compute the modular abelian variety attached to a newform in $S_2(\Gamma_1(25))$.

```
> S := CuspForms(Gamma1(25),2);
> N := Newforms(S);
> #N;
2
> f := N[1][1];
> PowerSeries(f,4);
q + a*q^2 + 1/1355*(941*a^7 + 4820*a^6 + 11150*a^5 + 11522*a^4 +
    3582*a^3 + 10041*a^2 + 24432*a - 5718)*q^3 + O(q^4)
> A_f := ModularAbelianVariety(f);
> A_f;
Modular abelian variety Af of dimension 8 and level 5^2 over Q
```

The abelian variety A_f also determines the newform:

```
> PowerSeries(Newform(A_f),4);
q + a*q^2 + 1/1355*(941*a^7 + 4820*a^6 + 11150*a^5 + 11522*a^4 +
    3582*a^3 + 10041*a^2 + 24432*a - 5718)*q^3 + O(q^4)
```

Example H142E7

The `Newform` command works even if A wasn't explicitly created using a newform.

```
> A := Decomposition(JZero(37))[1];
> Newform(A);
q - 2*q^2 - 3*q^3 + 2*q^4 - 2*q^5 + 6*q^6 - q^7 + O(q^8)
```

142.2.4 Abelian Varieties Attached to Modular Symbols

The commands below associate modular abelian varieties to spaces of modular symbols and to sequences of spaces of modular symbols. Conversely, they associate spaces of modular symbols to modular abelian varieties. If an input space of modular symbols is not cuspidal, it is replaced by its cuspidal subspace.

`ModularAbelianVariety(M)`

The abelian variety attached to the modular symbols space M.

`ModularAbelianVariety(X)`

The abelian variety attached to the sequence X of modular symbols spaces.

`ModularSymbols(A)`

A sequence of spaces of modular symbols associated to the abelian variety A.

Example H142E8

We create modular abelian varieties attached to several spaces of modular symbols.

```
> M := ModularSymbols(37,2);
> ModularAbelianVariety(M);
Modular abelian variety of dimension 2 and level 37 over Q
> M := ModularSymbols(Gamma1(17));
> ModularAbelianVariety(M);
Modular abelian variety of dimension 5 and level 17 over Q
```

Note that the sign of the space of modular symbols determines the sign of the corresponding abelian variety.

```
> M := ModularSymbols(Gamma1(17),2,+1);
> A := ModularAbelianVariety(M); A;
Modular abelian variety of dimension 5 and level 17 over Q with sign 1
```

We can also create an abelian variety attached to any sequence of modular symbols spaces.

```
> ModularAbelianVariety([ModularSymbols(11), ModularSymbols(Gamma1(13))]);
```

```
Modular abelian variety of dimension 3 and level 11*13 over Q
```

Conversely, there is a sequence of modular symbols spaces associated to any abelian variety defined over $\mathbf{Q}$. These need not be the same as the spaces used to define the modular abelian variety; instead they are what is used internally in computations on that abelian variety.

```
> ModularSymbols(JOne(13));
[
    Modular symbols space of level 13, weight 2, character $.1,
    and dimension 2 over Cyclotomic Field of order 6 and degree 2
]
> ModularSymbols(JZero(37));
[
    Modular symbols space for Gamma_0(37) of weight 2 and
    dimension 4 over Rational Field
]
> A := JOne(17);
> ModularSymbols(A);
[
    Modular symbols space for Gamma_0(17) of weight 2 and
    dimension 2 over Rational Field,
    Modular symbols space of level 17, weight 2, character $.1,
    and dimension 2 over Cyclotomic Field of order 8 and degree 4
]
```

142.2.5 Creation of Abelian Subvarieties

Suppose A is an abelian variety and V is a vector subspace of the rational homology $H_1(A, \mathbf{Q})$. Then it can be determined whether or not V is the rational homology of an abelian subvariety of A, and if so that abelian subvariety can be computed.

The other commands below are used to create the zero-dimensional abelian variety.

`DefinesAbelianSubvariety(A, V)`

Return `true` if and only if the subspace V of rational homology defines an abelian subvariety of the variety A. If so, also returns the abelian subvariety.

This intrinsic relies on knowing a complete decomposition of A as a product of simple abelian varieties (so it is currently restricted to abelian varieties for which such a decomposition can be computed in Magma, e.g., modular abelian varieties over $\mathbf{Q}$).

`ZeroModularAbelianVariety()`

The zero-dimensional abelian variety.

`ZeroModularAbelianVariety(k)`

The zero-dimensional abelian variety of weight k.

`ZeroSubvariety(A)`

Returns the zero subvariety of the abelian variety A.

Example H142E9

We define two subspaces of the rational homology of $J_0(33)$; one defines an abelian subvariety and the other does not.

```
> A := JZero(33);
> w3 := AtkinLehnerOperator(A,3);
> W := Kernel(Matrix(w3)+1);
> DefinesAbelianSubvariety(A,W);
true Modular abelian variety of dimension 1 and level 3*11 over Q
> V := RationalHomology(A);
> DefinesAbelianSubvariety(A,W + sub<V|[V.1]>);
false
```

We create several zero-dimensional abelian varieties.

```
> ZeroModularAbelianVariety();
Modular abelian variety ZERO of dimension 0 and level 1 over Q
> ZeroModularAbelianVariety(2);
Modular abelian variety ZERO of dimension 0 and level 1 over Q
> ZeroSubvariety(JZero(11));
Modular abelian variety ZERO of dimension 0 and level 11 over Q
```

142.2.6 Creation Using a Label

As a useful shorthand, it is sometimes possible to create modular abelian varieties by giving a short string. If the string contains a single integer N, e.g., 37, then the corresponding abelian variety is $J_0(N)$. If it is of the form `"<level>k<weight>"`, then it is the possibly motivic $J_0(N)$ of weight k. If it is of the form `"<level>k<weight><isogeny code>"`, where `<isogeny code>` is one of `"A"`, `"B"`, ... `"Z"`, `"AA"`, `"BB"`, ... `"ZZ"`, `"AAA"`, `"BBB"`, ... then the corresponding abelian variety is `JZero(N,k)(iso)`, where iso is a positive integer, and `"A"` corresponds to iso=1, `"Z"` to iso=26, `"AA"` to iso=27, `"ZZ"` to iso=52, `"AAA"` to iso=53, etc.

This labeling convention is the same as the one used for modular symbols, and extends the one used for Cremona's database of elliptic curves, except that Cremona's database contains some random scrambling for levels between 56 and 450. If the weight part of the label is omitted, the weight is assumed to be 2. To get the optimal quotient of $J_0(N)$ with Cremona label s, set the optional parameter `Cremona` equal to true.

`ModularAbelianVariety(s : parameters)`

`Sign`	RngIntElt	*Default :* 0
`Cremona`	BoolElt	*Default :* `false`

The abelian variety defined by the string s. The parameter `Sign` determines whether computations will be carried out in the $+1$ or -1 quotient of homology. If the parameter `Cremona` is set to `true`, the optimal quotient of $J_0(N)$ with Cremona label s will be returned.

Example H142E10

```
> ModularAbelianVariety("37");
Modular abelian variety 37 of dimension 2 and level 37 over Q
> ModularAbelianVariety("37A");
Modular abelian variety 37A of dimension 1 and level 37 over Q
> ModularAbelianVariety("11k4A");
Modular motive 11k4A of dimension 2 and level 11 over Q
> ModularAbelianVariety("65C");
Modular abelian variety 65C of dimension 2 and level 5*13 over Q
> ModularDegree(ModularAbelianVariety("56A"));
4
> ModularDegree(ModularAbelianVariety("56A" : Cremona := true));
2
```

142.2.7 Invariants

Commands are available which can retrieve the base ring, dimension, character of the defining modular form, a field of definition, the level, the sign, the weights, and a short name of a modular abelian variety.

`BaseRing(A)`

The ring that the modular abelian variety A is defined over.

`Dimension(A)`

The dimension of the modular abelian variety A.

`DirichletCharacter(A)`

If the modular abelian variety $A = A_f$ is attached to a newform, returns the Nebentypus character of f. Since f is only well-defined up to $\mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})$ conjugacy, the character is also only well-defined up to $\mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q})$ conjugacy.

For information on Dirichlet characters, see 19.8.

`DirichletCharacters(A)`

List of all Dirichlet characters of spaces of modular symbols associated with the modular symbols abelian variety which parameterizes A.

`FieldOfDefinition(A)`

The best known field of definition of the modular abelian variety A.

`Level(A)`

An integer N such that the modular abelian variety A is a quotient of a power of $J_1(N)$. Note that N need not be minimal. It is determined by how A is explicitly represented as a quotient of modular Jacobians.

`Sign(A)`

The sign of the modular abelian variety A, which is either 0, -1, or $+1$. If the sign is $+1$ or -1, then only the corresponding complex-conjugation eigenspace of the homology of A is computed, so various computations will be off by a factor of 2.

`Weights(A)`

The set of weights of the modular abelian variety A. The weight need not be unique since direct sums of modular symbols spaces of different weights are allowed.

Example H142E11

We illustrate all the commands for $J_0(23)$.

```
> A := JZero(23);
> BaseRing(A);
Rational Field
> Dimension(A);
2
> DirichletCharacter(A);
1
> FieldOfDefinition(A);
Rational Field
> Level(A);
23
> Sign(A);
0
> Weights(A);
{ 2 }
```

Example H142E12

This is an example of a nontrivial Dirichlet character.

```
> eps := DirichletCharacter(JOne(23)(2)); eps;
$.1^2
> Order(eps);
11
```

Example H142E13

We illustrate the `Weights` command in several cases.

```
> Weights(JZero(11));
{ 2 }
> Weights(JZero(11,4));
{ 4 }
> Weights(JOne(13,3));
{ 3 }
> Weights(DirectSum(JZero(11),JOne(13,3)));
{ 2, 3 }
> Weights(DirectSum(JZero(11),JZero(13,3)));
{ 2 }
```

Example H142E14

We display a few fields of definition.

```
> FieldOfDefinition(JOne(13));
Rational Field
> FieldOfDefinition(BaseExtend(JZero(11),QuadraticField(7)));
Rational Field
> FieldOfDefinition(ChangeRing(JZero(11),GF(7)));
Finite field of size 7
```

Example H142E15

In the following example we quotient $J_0(11)$ out by a 5-torsion point. The resulting abelian variety might not be defined over $\mathbf{Q}$, and the `FieldOfDefinition` command currently plays it safe and returns $\overline{\mathbf{Q}}$.

```
> A := JZero(11);
> G := nTorsionSubgroup(A,5);
> H := Subgroup([G.1]);
> H;
Finitely generated subgroup of abelian variety with invariants [ 5 ]
> FieldOfDefinition(A/H);
Algebraically closed field with no variables
```

142.2.8 Conductor

Let A be a modular abelian variety over $\mathbf{Q}$. The conductor command computes the conductor of A by factoring A into newform abelian varieties A_f whose conductor is N^d, where N is the level of f and d is the dimension of A_f.

`Conductor(A)`

The conductor of the abelian variety A. We require that A is defined over $\mathbf{Q}$. When $A = A_f$ is attached to a newform of level N, then the conductor of A is N^d, where d is the dimension of A.

Example H142E16

```
> Factorization(Conductor(JZero(33)));
[ <3, 1>, <11, 3> ]
> Factorization(Conductor(JZero(11)^5));
[ <11, 5> ]
> Factorization(Conductor(OldSubvariety(JZero(46))));
[ <23, 4> ]
> Factorization(Conductor(JOne(25)));
[ <5, 24> ]
```

142.2.9 Number of Points

Given an abelian variety A over a field K a divisor and a multiple of $\#A(K)$ can be computed. When finite, the multiple of the number of rational points is computed using reduction mod primes up to 100. Currently the lower bound is nontrivial only when A is a quotient of $J_0(N)$.

`NumberOfRationalPoints(A)`

A divisor and a multiple of the cardinality of $A(K)$, where A is a modular abelian variety defined over a field K. If K is an abelian number field, then we assume the Birch and Swinnerton-Dyer conjecture.

`#A`

The cardinality of $A(K)$ where A is a modular abelian variety defined over the field K if an exact value for this cardinality is known.

Example H142E17

```
> #JZero(11);
5
> #JZero(23);
11
> NumberOfRationalPoints(JZero(37));
Infinity Infinity
> NumberOfRationalPoints(JOne(13));
1 19
> NumberOfRationalPoints(JOne(23));
1 408991
> Factorization(408991);
[ <11, 1>, <37181, 1> ]
> NumberOfRationalPoints(ModularAbelianVariety("43B"));
7 7
```

142.2.10 Inner Twists and Complex Multiplication

If f is a newform then an *inner twist* of f is a Dirichlet character χ such that the twist of f by χ equals a Galois conjugate of f, at least at Fourier coefficients whose subscript is coprime to some fixed integer. A *CM twist* is a nontrivial character χ such that f twisted by χ equals f, at least at Fourier coefficients whose subscript is coprime to some fixed integer. The commands below find the CM and inner twists of the newform corresponding to a newform abelian variety.

The optional parameter `Proof` to each command is by default `false`. If `true`, it uses a huge number of terms of q-expansions. If `false`, it uses far less (and is hence very quick), and in practice this should be fine. The computation of inner and CM twists is not provably correct, even if the `Proof := true` option is set. It's very likely to be correct when `Proof := true`, but not provably correct. (More precisely, MAGMA only checks that each twist is a twist to precision 10^{-5}, but does not prove that this precision is sufficient.)

`CMTwists(A : parameters)`

`Proof` BOOLELT *Default :* `false`

A sequence of the CM inner twist characters of the newform abelian variety $A = A_f$ that are defined over the base ring of A. To gain all CM twists, base extend to `AlgebraicClosure(RationalField())` first.

`InnerTwists(A : parameters)`

`Proof` BOOLELT *Default :* `false`

A sequence of the inner twist characters of the newform abelian variety $A = A_f$ that are defined over the base ring of A. To gain all inner twists, first base extend to `AlgebraicClosure(RationalField())`.

Example H142E18

We compute the inner twists for $J_1(13)$.

```
> A := JOne(13); A;
Modular abelian variety JOne(13) of dimension 2 and level 13 over Q
> CMTwists(A);
[]
> A2 := BaseExtend(A,AlgebraicClosure(RationalField()));
> CMTwists(A2);
[]
> InnerTwists(A2);
[
    1,
    $.1^5
]
> Parent($1[2]);
Group of Dirichlet characters of modulus 13 over Cyclotomic Field
of order 6 and degree 2
```

We compute the inner twists for the second newform factor of $J_1(23)$.

```
> A := Decomposition(JOne(23))[2]; A;
Modular abelian variety image(23A[2]) of dimension 10, level 23
and conductor 23^10 over Q
> InnerTwists(BaseExtend(A,AlgebraicClosure(RationalField())));
[
    1,
    $.1^20
]
```

The CM elliptic curve $J_0(32)$ has a nontrivial CM inner twist.

```
> A := JZero(32);
> InnerTwists(BaseExtend(A,AlgebraicClosure(RationalField())));
[
    1,
    $.1
]
> CMTwists(BaseExtend(A,AlgebraicClosure(RationalField())));
[
    $.1
]
```

Quotients of $J_0(N)$ can also have nontrivial inner twists, which are not CM twists.

```
> J := JZero(81);
> A := Decomposition(J)[1];
> InnerTwists(BaseExtend(A,AlgebraicClosure(RationalField())));
[
    1,
```

```
    $.1
]
> CMTwists(BaseExtend(A,AlgebraicClosure(RationalField())));
[]
> Newform(A);
q + a*q^2 + q^4 - a*q^5 + 2*q^7 + O(q^8)
```

Example H142E19

The following is an example of a 4-dimensional abelian variety $A = A_f$ in $J_0(512)$ that has four inner twists, none of which are CM twists. One can use this fact to prove that if a_p is a prime-indexed Fourier coefficient of f, then $a_p^2 \in \mathbf{Z}$. Thus no single a_p generates the degree 4 field generated by all a_n.

```
> J := JZero(512, 2, +1);
> A := Decomposition(J)[7]; A;
Modular abelian variety 512G of dimension 4, level 2^9 and
conductor 2^36 over Q with sign 1
> f := Newform(A); f;
q + 1/12*(a^3 - 30*a)*q^3 + 1/12*(-a^3 + 42*a)*q^5 +
     1/6*(-a^2 + 18)*q^7 + O(q^8)
> Coefficient(f,3)^2;
6
> Coefficient(f,5)^2;
12
> Coefficient(f,7)^2;
8
> Abar := BaseExtend(JZero(512,2,+1)(7),AlgebraicClosure(RationalField()));
> InnerTwists(Abar);
[
    1,
    $.1,
    $.2,
    $.1*$.2
]
> CMTwists(Abar);
[]
```

142.2.11 Predicates

Most of the predicates below work in full generality. The ones whose domain of applicability is somewhat limited are `IsIsomorphic`, `IsQuaternionic`, and `IsSelfDual`. In theory, it is possible to determine isomorphism for more general classes of modular abelian varieties, but this has not yet been implemented.

`CanDetermineIsomorphism(A, B)`

Return `true` if we can determine whether or not the modular abelian varieties A and B are isomorphic. If we can determine isomorphism, also returns `true` if A and B are isomorphic and an explicit isomorphism, or `false` if they are not isomorphic. If we can not determine isomorphism, also returns the reason why we cannot as a string. If A and B are simple and defined over $\mathbf{Q}$, then there is an algorithm to determine whether or not A and B are isomorphic; it has been implemented in most (but not all) cases in MAGMA. If one of A or B has simple factors of multiplicity one, then in principal it is possible, but the algorithm has not been programmed.

`HasMultiplicityOne(A)`

Return `true` if the simple factors of the modular abelian variety A appear with multiplicity one.

`IsAbelianVariety(A)`

Return `true` if A is an abelian variety, i.e., defined over a ring of characteristic 0 in which the conductor is invertible, or a finite field whose characteristic does not divide the conductor of A. For example, if A has positive dimension and is defined over $\mathbf{Z}$, then Raynaud's theorem implies that A is not an abelian variety.

`IsAttachedToModularSymbols(A)`

Return `true` if the underlying homology of the modular abelian variety A is being computed using a space of modular symbols. For example, this will be true for $J_0(N)$ and for newform abelian varieties.

`IsAttachedToNewform(A)`

Return `true` if the modular abelian variety A is isogenous to a newform abelian variety A_f. This intrinsic also returns the abelian variety A_f and an explicit isogeny from A_f to A.

`IsIsogenous(A, B)`

Return `true` if the modular abelian varieties A and B are isogenous. If this can *not* be determined, an error occurs. It is always possible to determine whether or not A and B are isogenous when both are defined over $\mathbf{Q}$.

`IsIsomorphic(A, B)`

Return `true` if the modular abelian varieties A and B are isomorphic. If so, also returns an explicit isomorphism. This command will work if A and B are defined over $\mathbf{Q}$ and the simple factors occur with multiplicity one, and may work otherwise, but an error may occur in the general case. Use the command `CanDetermineIsomorphism` to avoid getting an error.

`IsOnlyMotivic(A)`

Return `true` if any of the modular forms attached to the modular abelian variety A have weight bigger than 2.

`IsQuaternionic(A)`

Return `true` if and only if some simple factor of the modular abelian variety A over the base ring has quaternionic multiplication.

`IsSelfDual(A)`

Return `true` if the modular abelian variety A is known to be isomorphic to its dual. An error occurs if MAGMA is unable to decide.

`IsSimple(A)`

Return `true` if and only if the modular abelian variety A has no proper abelian subvarieties over `BaseRing(A)`.

Example H142E20

We test whether a few objects are actually abelian varieties.

```
> A := JZero(11);
> A11 := ChangeRing(A,GF(11));
> IsAbelianVariety(A11);
false
> AZ := ChangeRing(A,Integers());
> IsAbelianVariety(AZ);
false
> A3 := ChangeRing(A,pAdicRing(3));
> IsAbelianVariety(A3);
true
```

Example H142E21

A modular motive is sometimes also an abelian variety, but only over the complex numbers.

```
> A := JZero(11,4); A;
Modular motive JZero(11,4) of dimension 2 and level 11 over Q
> IsAbelianVariety(A);
false
> IsOnlyMotivic(A);
true
> IsAbelianVariety(BaseExtend(A,ComplexField()));
true
```

Example H142E22

Abelian varieties $J_0(N)$ and $J_s(N)$ are attached to modular symbols, as are newform abelian varieties.

```
> J := JZero(37);
> IsAttachedToModularSymbols(J);
true
> A := Decomposition(J)[1];
> IsAttachedToModularSymbols(A);
false
> t, Af := IsAttachedToNewform(A);
> IsAttachedToModularSymbols(Af);
true
> IsIsomorphic(A,Af);
true Homomorphism from 37A to 37A given on integral homology by:
[1 0]
[0 1]
> IsAttachedToModularSymbols(Js(17));
true
> IsAttachedToModularSymbols(JOne(17));
false
```

Example H142E23

We test isogeny between a few abelian varieties.

```
> IsIsogenous(JZero(11),JOne(11));
true
> IsIsogenous(JZero(11)*JZero(11),JZero(22));
true
> IsIsogenous(JZero(11)*JZero(11),JZero(33));
false
> IsIsogenous(JZero(11),JZero(14));
false
> IsIsogenous(JZero(11)^2,JZero(22));
```

```
true
> A := JZero(37)(2); B := JOne(13);
> IsIsogenous(A*B*A, A*A*B);
true
> A := JZero(43);
> G := RationalCuspidalSubgroup(A);
> IsIsogenous(A,A/G);
true
```

Next we test isomorphism between some abelian varieties.

```
> A := JZero(43);
> G := RationalCuspidalSubgroup(A);
> B := A/G;
> CanDetermineIsomorphism(A,B);
true false
> IsIsomorphic(A,B);
false
> IsIsomorphic(JZero(11),JOne(11));
false
> IsIsomorphic(JZero(13),JOne(13));
false
> IsIsomorphic(Js(13),JOne(13));
true Homomorphism from Js(13) to JOne(13) given on integral
homology by:
[ 0  0 -1  0]
[ 0  0  0 -1]
[ 1  0 -1  0]
[ 0  1  0 -1]
> CanDetermineIsomorphism(Js(17),JOne(17));
false All tests failed to decide whether A and B are isomorphic.
```

Example H142E24

We test whether certain Jacobians have simple factors with multiplicity one.

```
> HasMultiplicityOne(JZero(43));
true
> HasMultiplicityOne(JZero(33));
false
> Decomposition(JZero(33));
[
    Modular abelian variety 33A of dimension 1, level 3*11 and
    conductor 3*11 over Q,
    Modular abelian variety N(11,33,1)(11A) of dimension 1, level
    3*11 and conductor 11 over Q,
    Modular abelian variety N(11,33,3)(11A) of dimension 1, level
    3*11 and conductor 11 over Q
```

```
]
```

Example H142E25

We give an example of a non-quaternionic surface.

```
> IsQuaternionic(JOne(13));
false
```

Example H142E26

Jacobians are self dual, and there is a surface of level 43 that is self dual and a surface of level 69 that is not.

```
> IsSelfDual(JOne(13));
true
> IsSelfDual(JZero(69)(2));
false
```

The surface A below is isomorphic to its dual. The natural polarization has kernel that is the kernel of multiplication by 2.

```
> A  := JZero(43)(2);
> A;
Modular abelian variety 43B of dimension 2, level 43 and
conductor 43^2 over Q
> IsSelfDual(A);
true
> phi := ModularPolarization(A);
> Invariants(Kernel(phi));
[ 2, 2, 2, 2 ]
```

Example H142E27

We test a few abelian varieties for simplicity.

```
> IsSimple(JOne(25));
false
> IsSimple(JOne(13));
true
> IsSimple(JZero(11)^10);
false
> IsSimple(NewSubvariety(JZero(100)));
true
```

142.2.12 Equality and Inclusion Testing

These functions test whether two abelian varieties are exactly equal or if one is a subset of another.

`A eq B`

Return `true` if the modular abelian varieties A and B are equal.

`A subset B`

Return `true` if the modular abelian variety A is a subset of the modular abelian variety B.

Example H142E28

This example illustrates that taking the direct product of abelian varieties is not commutative, as measured by equality (though it is as measured by isomorphism).

```
> A := JZero(11);
> B := JZero(14);
> A*B eq A*B;
true
> A*B eq B*A;
false
> IsIsomorphic(A*B, B*A);
true Homomorphism N(1) from JZero(11) x JZero(14) to JZero(14) x JZero(11)
given on integral homology by:
[0 0 1 0]
[0 0 0 1]
[1 0 0 0]
[0 1 0 0]
```

Example H142E29

The first inclusion below is as expected, but the second non-inclusion might be surprising. We do not consider $J_0(11)$ as a subset of $J_0(22)$, even though there is an injective map from one to the other, since to be a subset is much stronger than just the existence of an inclusion map.

```
> JZero(37) subset JZero(37);
true
> JZero(11) subset JZero(22);
false
> IsInjective(NaturalMap(JZero(11),JZero(22)));
true
```

142.2.13 Modular Embedding and Parameterization

Every modular abelian variety A is equipped with a modular parameterization and a modular embedding. The modular parameterization is a surjective homomorphism from a modular symbols abelian variety, such as $J_0(N)$. The modular embedding is a homomorphism to a modular symbols abelian variety, which is only guaranteed to be *injective in the category of abelian varieties up to isogeny*, i.e., to have finite kernel as a morphism of abelian varieties. The structure of these two homomorphisms is extremely important as they completely define A.

`CommonModularStructure(X)`

This intrinsic finds modular abelian varieties J_e and J_p associated to modular symbols and returns a list of finite-kernel maps from the abelian varieties in the sequence X to J_e and a list of modular parameterizations from J_p to the abelian varieties in X.

`ModularEmbedding(A)`

A morphism with finite kernel from the modular abelian variety A to a modular abelian variety attached to modular symbols. This is only guaranteed to be an embedding in the category of abelian varieties up to isogeny.

`ModularParameterization(A)`

A surjective morphism to the modular abelian variety A from an abelian variety attached to modular symbols.

Example H142E30

```
> X := [JZero(11),ModularAbelianVariety("37B")];
> CommonModularStructure(X);
[*
Homomorphism from JZero(11) to JZero(11) x JZero(37) given on integral
homology by:
[1 0 0 0 0 0]
[0 1 0 0 0 0],
Homomorphism from 37B to JZero(11) x JZero(37) given on integral
homology by:
[0 0 1 1 1 0]
[0 0 0 0 0 1]
*] [*
Homomorphism from JZero(11) x JZero(37) to JZero(11) (not printing 6x2
matrix),
Homomorphism from JZero(11) x JZero(37) to 37B (not printing 6x2
matrix)
*]
```

Example H142E31

This example illustrates that the modular "embedding" need only be an embedding in the category of abelian varieties up to isogeny.

```
> A := JZero(37)(1);
> x := A![1/2,1];
> B := A/Subgroup([x]);
> e := ModularEmbedding(B);
> e;
Homomorphism from modular abelian variety of dimension 1 to
JZero(37)_Qbar given on integral homology by:
[ 1 -1  1  0]
[ 2 -2 -2  2]
> IsInjective(e);
false
```

Moreover, the modular parameterization is surjective, but it need be optimal (have connected kernel).

```
> pi := ModularParameterization(B);
> IsSurjective(pi);
true
> ComponentGroupOfKernel(pi);
Finitely generated subgroup of abelian variety with invariants [ 2 ]
> IsOptimal(pi);
false
```

142.2.14 Coercion

Coercion can be used to create points on modular abelian varieties from vectors on a basis of integral homology, from other elements of modular abelian varieties, or from modular symbols. See the examples below, especially the last one, to understand some of the subtleties of coercion that arise because we view an abelian variety as a vector space modulo a lattice, and that lattice can be embedded in any way in $\mathbf{Q}^n$.

`A ! x`

Coerce x into the modular abelian variety A. The argument x can be an element of a modular abelian variety, the integer 0, a sequence like that obtained by from `Eltseq` of element of A (i.e., a linear combination of *integral* homology), a vector on the basis for *rational* homology, or a tuple of the form $\langle P(X,Y), [u,v]\rangle$ defining a modular symbol.

Example H142E32

If you coerce a *sequence* of rationals or reals into an abelian variety A, then MAGMA computes the corresponding linear combination of a basis of integral homology and returns the point it defines. The sequence must have length the rank of the integral homology.

```
> JZero(11)![1/2,1/5];
Element of abelian variety defined by [1/2 1/5] modulo homology
```

If you coerce exactly two cusps (or extended reals) into A, then MAGMA computes the point corresponding to that modular symbol.

```
> JZero(11)![Cusps()|1/2,1/5];
0
> JZero(11)![Sqrt(2),0];
Element of abelian variety defined by
[1.41421356237309504880168872419 8 0] modulo homology
> JZero(11)![Cusps()|0,Infinity()];     // cusps
Element of abelian variety defined by [0 1/5] modulo homology
> JZero(11)![0,Infinity()];             // extended reals
Element of abelian variety defined by [0 1/5] modulo homology
```

Coercion of modular symbols also works for higher weight.

```
> JZero(11,4)![0,Infinity()];
Element of abelian variety defined by [-4/61 5/61 1/61 -1/61]
modulo homology
> R<x,y> := PolynomialRing(RationalField(),2);
> JZero(11,4)!<x^2,[0,Infinity()]>;
Element of abelian variety defined by [-4/61 5/61 1/61 -1/61]
modulo homology
> JZero(11,4)!<y^2,[0,Infinity()]>;
Element of abelian variety defined by [44/61 -55/61 -11/61 11/61]
modulo homology
```

You can also coerce elements from abelian subvarieties into an ambient abelian variety.

```
> J := JZero(37); A := Decomposition(J)[1];
> x := A![1/5,0];
> Parent(x);
Modular abelian variety 37A of dimension 1, level 37 and
conductor 37 over Q
> x in J;
false
> y := J!x; y;
Element of abelian variety defined by [1/5 -1/5 1/5 0] modulo
homology
> y in J;
true
> Parent(y);
```

```
Modular abelian variety JZero(37) of dimension 2 and level 37 over Q
```

Coercion also provides an easy way to create the 0 element.

```
> JZero(37)!0;
0
```

Example H142E33

The following example illustrates the subtlety of coercion when the element being coerced in is a vector instead of a sequence. We create the quotient of $J_0(11)$ by a cyclic subgroup of order 10. The lattice that defines $J_0(11)$ is $\mathbf{Z} \times \mathbf{Z}$, but the lattice that defines this quotient is $(1/10)\mathbf{Z} \times \mathbf{Z}$. Thus the natural quotient map on vector spaces is defined by the identity matrix. On the other hand, the matrix of the quotient with respect to a basis for integral homology has determinant 10.

```
> A := JZero(11);
> x := A![1/10,0]; x;
Element of abelian variety defined by [1/10 0] modulo homology
> Order(x);
10
> B,pi := A/Subgroup([x]);
> B;
Modular abelian variety of dimension 1 and level 11 over Qbar
> pi;
Homomorphism from JZero(11)_Qbar to modular abelian variety of
dimension 1 given on integral homology by:
[10  0]
[ 0  1]
> Matrix(pi);
[1 0]
[0 1]
> IntegralMatrix(pi);
[10  0]
[ 0  1]
> base := Basis(IntegralHomology(B)); base;
[
    (1/10    0),
    (0 1)
]
```

If we coerce in the sequence [1/10,0] we get the point in B that is represented by 1/10th of the first generator for homology. If we coerce in the vector $(1/10, 0)$, we instead get the element of B represented by that element of the rational homology, which is 0, since the lattice that defines B is embedded in such a way that it contains $(1/10, 0)$.

```
> y := B![1/10,0]; y;
Element of abelian variety defined by [1/10 0] modulo homology
> Order(y);
10
```

```
> z := B!base[1]; z;
0
```

142.2.15 Modular Symbols to Homology

Modular symbols determine elements of the rational homology of $J_0(N)$, $J_1(N)$, etc., and hence of arbitrary modular abelian varieties, by using the modular parameterization. The commands below convert from modular symbols, which are represented in various ways, to vectors on the basis for rational or integral homology.

`ModularSymbolToIntegralHomology(A, x)`

The element of integral homology of the abelian variety A, naturally associated to the (formal) modular symbol $x = P(X,Y)\{\alpha,\beta\}$, where α, β are in $P^1(\mathbf{Q})$ and P is a homogeneous polynomial of degree 2. The returned vector is written with respect to the basis of integral homology. The argument x may be given as a sequence $[\alpha,\beta]$ or as a tuple $\langle P(X,Y),[\alpha,\beta]\rangle$ where α and β are in `Cusps()`.

`ModularSymbolToRationalHomology(A, x)`

The element of rational homology of the abelian variety A naturally associated to the (formal) modular symbol $x = P(X,Y)\{\alpha,\beta\}$, where α, β are in $P^1(\mathbf{Q})$ and P is a homogeneous polynomial of degree 2. The returned vector is written with respect to the basis of rational homology. The argument x may be given as a modular symbol, a sequence $[\alpha,\beta]$ or as a tuple $\langle P(X,Y),[\alpha,\beta]\rangle$ where α and β are in `Cusps()`.

Example H142E34

```
> A := JZero(11);
> x := ModularSymbolToIntegralHomology(A,[0,Infinity()]); x;
(  0 1/5)
> z := A!x; z;
Element of abelian variety defined by [0 1/5] modulo homology
> Order(z);
5
> A := JZero(47);
> x := ModularSymbolToIntegralHomology(A,[0,Infinity()]); x;
(-1/23  3/23 -4/23  4/23 -3/23  1/23  2/23  8/23)
> z := A!x;
> Order(z);
23
```

Example H142E35

```
> J := JZero(11,4);
> IntegralHomology(J);
Lattice of rank 4 and degree 4
Basis:
( 3  1 -1 -1)
( 1 -1 -3  1)
( 2 -2  2  2)
( 2 -2  2 -2)
Basis Denominator: 8
> ModularSymbolToIntegralHomology(J,[0,Infinity()]);
(-4/61  5/61  1/61 -1/61)
```

Notice that for weight greater than 2 the homogeneous polynomial part of the modular symbol may be omitted. If so, it defaults to x^{k-2}.

```
> R<x,y> := PolynomialRing(RationalField(),2);
> ModularSymbolToIntegralHomology(J,<x^2,[0,Infinity()]>);
(-4/61  5/61  1/61 -1/61)
> ModularSymbolToIntegralHomology(J,<y^2,[0,Infinity()]>);
( 44/61 -55/61 -11/61  11/61)
```

The result of coercion to rational homology is different because it is written in terms of the basis for rational homology instead of the basis for integral homology, and in this example the two basis differ.

```
> ModularSymbolToRationalHomology(J,[0,Infinity()]);
( -7/488  -9/488 -11/488  13/488)
```

Coercion is also a way to define torsion points on abelian varieties.

```
> JZero(37)![1/5,0,0,0];
Element of abelian variety defined by [1/5 0 0 0] modulo homology
```

142.2.16 Embeddings

The `Embeddings` command contains a list of embeddings (up to isogeny) from A to other abelian varieties. The `AssertEmbedding` command allows you to add an embedding to the beginning of the list. The embeddings are used for computing intersections, sums, etc., with the embedding at the front of the list having highest priority.

`Embeddings(A)`

A list of morphisms from the modular abelian variety A into abelian varieties, which are used in making sense of intersections, sums, etc. The embeddings at the beginning of the list take precedence over those that occur later. Note that these maps might not really be injective; e.g., the modular embedding, which need only be injective on homology, is at the end of this list.

`AssertEmbedding(~A, phi)`

Place the homomorphism ϕ of abelian varieties at the beginning of `Embeddings(A)` where A is a modular abelian variety. The morphism ϕ must have finite kernel.

Example H142E36

Every modular abelian variety comes equipped with at least one embedding, the "modular embedding".

```
> Embeddings(JZero(11));
[*
Homomorphism from JZero(11) to JZero(11) given on integral homology by:
[1 0]
[0 1]
*]
> A := JZero(37)(1);
> Embeddings(A);
[*
Homomorphism from 37A to JZero(37) given on integral homology by:
[ 1 -1  1  0]
[ 1 -1 -1  1]
*]
```

We add another embedding to the list of embeddings for A.

```
> phi := NaturalMap(A,JZero(37*2));
> AssertEmbedding(~A,phi);
> Embeddings(A);
[*
Homomorphism N(1) from 37A to JZero(74) given on integral homology
by:
[-1  1 -1  0  2 -1  1  0 -2  1 -2  2 -1  2 -1  1]
[-1  0  0  0  2 -1  1 -2  0  0 -1  1 -1  2  1 -1],
Homomorphism from 37A to JZero(37) given on integral homology by:
[ 1 -1  1  0]
[ 1 -1 -1  1]
*]
```

The following intersection would not make sense if we hadn't chosen an embedding of A into $J_0(74)$.

```
> B := Codomain(phi)(1); B;
Modular abelian variety 74A of dimension 2, level 2*37 and
conductor 2^2*37^2 over Q
> #(A meet B);
1
```

142.2.17 Base Change

The `BaseExtend` and `ChangeRing` commands allow you to change the base ring of a modular abelian variety. The `BaseExtend` command is the same `ChangeRing`, but is more restrictive. For example, if A is over $\mathbf{Q}$ then `BaseExtend(A,GF(2))` is not allowed, but `ChangeRing(A,GF(2))` is.

Abelian varieties can have their base ring set to a finite field, but there is very little that is implemented for abelian varieties over finite fields. Computing the number of points is implemented, but creation of actual points or homomorphisms is not.

`CanChangeRing(A, R)`

Return `true` if it is possible to change the base ring of the modular abelian variety A to the ring R, and the modular abelian variety over R when possible.

`ChangeRing(A, R)`

Return the modular abelian variety obtained from the modular abelian variety A having base ring R.

`BaseExtend(A, R)`

Extend the base ring of the modular abelian variety A to the ring R, if possible. This is a more restrictive version of `ChangeRing`.

Example H142E37

We consider $J_1(13)$ over many fields and rings.

```
> A := JOne(13);
> BaseExtend(A,CyclotomicField(7));
Modular abelian variety JOne(13) of dimension 2 and level 13 over
Q(zeta_7)
> BaseExtend(A,AlgebraicClosure(RationalField()));
Modular abelian variety JOne(13)_Qbar of dimension 2 and level 13
over Qbar
> BaseExtend(A,RealField());
Modular abelian variety JOne(13)_R of dimension 2 and level 13 over R
> BaseExtend(A,ComplexField());
Modular abelian variety JOne(13)_C of dimension 2 and level 13 over C
> ChangeRing(A,GF(3));
Modular abelian variety JOne(13)_GF(3) of dimension 2 and level 13
over GF(3)
> #ChangeRing(A,GF(3));
19 19
> B := ChangeRing(A,GF(13)); B;
Modular abelian variety JOne(13)_GF(13) of dimension 2 and level 13
over GF(13)
> IsAbelianVariety(B);
false
> ChangeRing(A,Integers());
```

```
Modular abelian variety JOne(13)_Z of dimension 2 and level 13 over
Z
> ChangeRing(A,PolynomialRing(RationalField(),10));
Modular abelian variety JOne(13) of dimension 2 and level 13 over
Polynomial ring of rank 10 over Rational Field
Lexicographical Order
Variables: $.1, $.2, $.3, $.4, $.5, $.6, $.7, $.8, $.9, $.10
```

142.2.18 Additional Examples

Example H142E38

The simplest abelian variety is an abelian variety of dimension 0, i.e., a point.

```
> A := ZeroModularAbelianVariety(); A;
Modular abelian variety ZERO of dimension 0 and level 1 over Q
```

We create the Jacobian $J_0(22)$ of the modular curve $X_0(22)$ as follows:

```
> J := JZero(22); J;
Modular abelian variety JZero(22) of dimension 2 and level 2*11 over Q
```

Notice that A is a subset of $J_0(22)$.

```
> A subset J;
true
```

We can also create the higher weight analogues $J_0(N, k)$ of $J_0(N)$, which are motives defined over $\mathbf{Q}$. Many computations that make sense for $J_0(N)$ also make sense for these higher weight analogues.

```
> J4 := JZero(22,4); J4;
Modular motive JZero(22,4) of dimension 7 and level 2*11 over Q
> IsOnlyMotivic(J4);
true
```

One can also create $J_1(N)$:

```
> JOne(22);
Modular abelian variety JOne(22) of dimension 6 and level 2*11 over Q
```

For efficiency purposes, it is often much quicker to do computations working only with the $+1$ quotient of $H_1(J_0(N), \mathbf{Z})$, or using only the -1 quotient. These computations will be off by powers of 2, or subgroups will be halved and off by a power of 2. Nonetheless, if you know what you are doing, such computations can be very useful. Create $J_0(N)$, but working only with the $+1$ quotient of homology as follows:

```
> Jplus := JZero(22,2,+1); Jplus;
Modular abelian variety JZero(22) of dimension 2 and level 2*11 over Q
with sign 1
> Sign(Jplus);
1
```

```
> Sign(JZero(22));
0
```

Notice that the sign is printed out when it is 1 or -1. Also, sign 0 is shorthand for "no sign", i.e., working with the full homology.

Example H142E39

Let $\varepsilon : (\mathbf{Z}/N\mathbf{Z})^* \to \mathbf{Q}(\zeta_n)^*$ be Dirichlet character. The command `ModularAbelianVariety(eps)` creates the modular abelian variety *over* $\mathbf{Q}$ corresponding to the cusp forms with Dirichlet character any Galois conjugate of ε.

```
> G<eps> := DirichletGroup(22,CyclotomicField(EulerPhi(22)));
> Order(eps);
10
> Conductor(eps);
11
> A := ModularAbelianVariety(eps); A;
Modular abelian variety of dimension 0 and level 2*11 over Q
> A := ModularAbelianVariety(eps^2); A;
Modular abelian variety of dimension 4 and level 2*11 over Q
```

Example H142E40

Let H be a subgroup of $G = (\mathbf{Z}/N\mathbf{Z})^*$. The group $(\mathbf{Z}/N\mathbf{Z})^*$ acts by diamond bracket operators as a group of automorphisms on $X_1(N)$, and the modular curve $X_H(N)$ is the quotient of $X_1(N)$ by the action of H. Thus if $H = 1$, then $X_H(N) = X_1(N)$, and if $H = G$, then $X_H(N) = X_0(N)$. The command `JH(N,d)` creates an abelian variety *isogenous to* the Jacobian of $X_H(N)$, where d is the index of H in G (thus $d = 1$ corresponds to $J_0(N)$). A weight k and sign (either 0 or ± 1) can be provided. More precisely, `JH(N,d)` is the product of $J(\varepsilon)$, where ε varies over Dirichlet characters such that $\varepsilon(H) = \{1\}$.

```
> JH(22,1);
Modular abelian variety JZero(22) of dimension 2 and level 2*11 over Q
> JH(22,10);
Modular abelian variety Js(22) of dimension 6 and level 2*11 over Q
> JH(22,2);
Modular abelian variety J_H(22) of dimension 2 and level 2*11 over Q
```

As a shortcut, the command `Js(N)` creates a modular abelian variety that is isogenous to $J_1(N)$. Thus `Js(N)` is the same as `JH(N,d)`, where d is the order of $(\mathbf{Z}/N\mathbf{Z})^*$.

```
> Js(22);
Modular abelian variety Js(22) of dimension 6 and level 2*11 over Q
> JH(22,10) eq Js(22);
true
```

Example H142E41

We can also create modular abelian varieties attached to spaces of modular forms and spaces of modular symbols:

```
> ModularAbelianVariety(ModularForms(22));
Modular abelian variety of dimension 2 and level 2*11 over Q
> ModularAbelianVariety(ModularSymbols(22));
Modular abelian variety of dimension 2 and level 2*11 over Q
```

Example H142E42

Here is another example, in which the space of modular forms is on $\Gamma_1(25)$.

```
> M := ModularForms(Gamma1(25)); M;
Space of modular forms on Gamma_1(25) of weight 2 and dimension
39 over Integer Ring.
> S := CuspidalSubspace(M); S;
Space of modular forms on Gamma_1(25) of weight 2 and dimension
12 over Integer Ring.
> A := ModularAbelianVariety(S); A;
Modular abelian variety of dimension 12 and level 5^2 over Q
```

Example H142E43

We can also construct abelian varieties attached to newforms.

```
> S := CuspForms(43);
> N := Newforms(S); N;
[* [*
q - 2*q^2 - 2*q^3 + 2*q^4 - 4*q^5 + 4*q^6 + O(q^8)
*], [*
q + a*q^2 - a*q^3 + (-a + 2)*q^5 - 2*q^6 + (a - 2)*q^7 + O(q^8),
q + b*q^2 - b*q^3 + (-b + 2)*q^5 - 2*q^6 + (b - 2)*q^7 + O(q^8)
*] *]
> f := N[2][1]; f;
q + a*q^2 - a*q^3 + (-a + 2)*q^5 - 2*q^6 + (a - 2)*q^7 + O(q^8)
> A := ModularAbelianVariety(f); A;
Modular abelian variety Af of dimension 2 and level 43 over Q
> Newform(A);
q + a*q^2 - a*q^3 + (-a + 2)*q^5 - 2*q^6 + (a - 2)*q^7 + O(q^8)
```

When possible, we can also obtain a newform that gives rise to an abelian variety.

```
> J := JZero(43);
> D := Decomposition(J); D;
[
    Modular abelian variety 43A of dimension 1, level 43 and
    conductor 43 over Q,
    Modular abelian variety 43B of dimension 2, level 43 and
```

```
    conductor 43^2 over Q
]
> Newform(D[2]);
q + a*q^2 - a*q^3 + (-a + 2)*q^5 - 2*q^6 + (a - 2)*q^7 + O(q^8)
```

We demonstrate here how modular abelian varieties may be described by a label, using the string "43B". Continuing the previous code, we have:

```
> A := ModularAbelianVariety("43B");
> A eq D[2];
true
```

142.3 Homology

The homology $H_1(A, R)$ of an abelian variety A with coefficients in a ring R is a free R-module of rank equal to twice the dimension of A. For many purposes, we view abelian varieties as complex tori V/Λ, so there is a canonical isomorphism $\Lambda \cong H_1(A, \mathbf{Z})$. (If the sign of A is ± 1, then the homology command below gives a $\mathbf{Z}$-module of rank $\dim A$.)

142.3.1 Creation

The `Homology` command creates the first homology of a modular abelian variety, which is of type `ModAbVarHomol`. This is the only command for creating homology.

`Homology(A)`

The first integral homology of the modular abelian variety A. (If the sign of A is 1 or -1, then this is $\mathbf{Z}$-module of rank equal to the dimension of A.)

Example H142E44

The homology of the elliptic curve $J_0(14)$ is of dimension 2.

```
> A := JZero(14); A;
Modular abelian variety JZero(14) of dimension 1 and level 2*7 over Q
> Homology(A);
Modular abelian variety homology space of dimension 2
```

If, for efficiency purposes, we work in the $+1$ quotient, then we are only working with half the homology, so the dimension of the homology is 1.

```
> Homology(JZero(14,2 : Sign := +1));
Modular abelian variety homology space of dimension 1
```

142.3.2 Invariants

The only invariant of homology is its dimension. If A is an abelian variety, and H is its first homology, then H has dimension equal to twice the dimension of A. (Except if, for efficiency purposes, we are working with sign $+1$ or -1, in which case the dimension of H is equal to the dimension of A.)

`Dimension(H)`

The dimension of the space H of homology.

Example H142E45

```
> Dimension(Homology(JZero(100)));
14
> Dimension(Homology(JZero(100,2 : Sign := +1)));
7
> Dimension(Homology(JZero(100,2 : Sign := -1)));
7
```

142.3.3 Functors to Categories of Lattices and Vector Spaces

The following commands provide convenient functors from the categories of homology and modular abelian varieties to lattices and vector spaces. Mathematically, these functors are defined using the first homology of the complex manifold underlying the complex points of the abelian variety.

`IntegralHomology(A)`

The lattice underlying the homology of the modular abelian variety A.

`Lattice(H)`

The underlying lattice of the homology space H. This is a free $\mathbf{Z}$-module of rank equal to the dimension of H.

`RationalHomology(A)`

A $\mathbf{Q}$-vector space obtained by tensoring the homology of the modular abelian variety A with $\mathbf{Q}$.

`RealHomology(A)`

A vector space over $\mathbf{R}$ obtained by tensoring the homology of the modular abelian variety A with $\mathbf{R}$.

`RealVectorSpace(H)`

The $\mathbf{R}$-vector space whose dimension is the dimension of the homology space H.

`VectorSpace(H)`

The $\mathbf{Q}$-vector space whose dimension is the dimension of the homology space H.

Example H142E46

The integral, rational and real homology of a modular abelian variety A is $H_1(A, R)$, where $R = \mathbf{Z}, \mathbf{R}, \mathbf{Q}$, respectively. This homology is just a free module over R of dimension twice $\dim(A)$ (unless the sign of A is ± 1, in which case the homology is of dimension $\dim(A)$).

```
> J := JZero(22); J;
Modular abelian variety JZero(22) of dimension 2 and level 2*11 over Q
> IntegralHomology(J);
Standard Lattice of rank 4 and degree 4
> RationalHomology(J);
Full Vector space of degree 4 over Rational Field
> RealHomology(J);
Full Vector space of degree 4 over Real Field
> J := JZero(22, 2, +1); J;
Modular abelian variety JZero(22) of dimension 2 and level 2*11 over Q
with sign 1
> RationalHomology(J);
Full Vector space of degree 2 over Rational Field
```

Example H142E47

The following code demonstrates the above commands applied to the abelian surface attached to the space of cusp forms of level 37 with quadratic character.

```
> G<eps> := DirichletGroup(37);
> Order(eps);
2
> A := ModularAbelianVariety(eps); A;
Modular abelian variety of dimension 2 and level 37 over Q
> Decomposition(A);
[
    Modular abelian variety image(37A[18]) of dimension 2, level
    37 and conductor 37^2 over Q
]
> IntegralHomology(A);
Standard Lattice of rank 4 and degree 4
> H := Homology(A); H;
Modular abelian variety homology space of dimension 4
> Lattice(H);
Standard Lattice of rank 4 and degree 4
> RationalHomology(A);
Full Vector space of degree 4 over Rational Field
> RealHomology(A);
Full Vector space of degree 4 over Real Field
> RealVectorSpace(H);
Full Vector space of degree 4 over Real Field
> VectorSpace(H);
```

```
Full Vector space of degree 4 over Rational Field
```

Example H142E48

The code below illustrates that the lattice need not just be the free lattice on $\dim(H)$ generators. For efficiency reasons many internal computations only require knowing $H_1(A, \mathbf{Q})$, which is sometimes all that gets computed, and the lattice returned by this command is $H_1(A, \mathbf{Z})$ written with respect to a basis for $H_1(A, \mathbf{Q})$. Thus `Lattice` should be viewed as giving an integral structure of $H_1(A, \mathbf{Q})$.

```
> J := JZero(37);
> A := J(1);
> Lattice(Homology(A));
Lattice of rank 2 and degree 2
Basis:
( 1  1)
( 1 -1)
> IntegralHomology(JOne(17));
Lattice of rank 10 and degree 10
Basis:
( 0  0  1  0  0  0  1  0  0  0)
( 0  0  1  0  0  0  0  0 -1  0)
( 0  0  0  1  0  0  0  1  0  0)
( 0  0  0  1  0  0  0 -1  0  0)
( 0  0  0  0  1  0  0  0 -1  0)
( 0  0  0  0  0  1  0  0  0  1)
( 0  0  0  0  0  1  0  0  0 -1)
( 0  0  0  0  0  0  1  0 -1  0)
( 0  2  1  0  0  0  0  0  0  0)
( 2 -1  0  0  0  0  1 -1  0  1)
```

142.3.4 Modular Structure

If H is the homology of an abelian variety that is attached to a space of modular symbols, then H remembers that space of modular symbols. The following two functions decide if H is attached to modular symbols, and if so provide the corresponding spaces of modular symbols. The reason that `ModularSymbols` returns a sequence instead of a single modular symbols space is that arbitrary finite products of abelian varieties are allowed, so corresponding direct sums of modular symbols spaces must be allowed, which are represented as sequences because MAGMA currently doesn't have a facility for taking direct sums of modular symbols spaces.

`IsAttachedToModularSymbols(H)`

Return `true` if the homology space H is presented as being attached to a sequence of spaces of modular symbols.

`ModularSymbols(H)`

If the space of homology H is attached to a sequence of spaces of modular symbols, then this is that sequence. Otherwise an error occurs.

Example H142E49

In this example, we create the product $J_0(23) \times J_0(11)$, and find that its homology is attached modular symbols, since both $J_0(23)$ and $J_0(11)$ are attached to modular symbols. We then display the corresponding spaces of modular symbols.

```
> A := JZero(23) * JZero(11);
> H := Homology(A); H;
Modular abelian variety homology space of dimension 6
> IsAttachedToModularSymbols(H);
true
> ModularSymbols(H);
[
    Modular symbols space for Gamma_0(23) of weight 2 and
    dimension 4 over Rational Field,
    Modular symbols space for Gamma_0(11) of weight 2 and
    dimension 2 over Rational Field
]
```

142.3.5 Additional Examples

Example H142E50

The following example illustrates each of the above intrinsics for the homology of $J_0(23)$.

```
> A := JZero(23); A;
Modular abelian variety JZero(23) of dimension 2 and level 23 over Q
> H := Homology(A); H;
Modular abelian variety homology space of dimension 4
```

Notice that homology has its own type:

```
> Type(H);
ModAbVarHomol
```

In this case, the homology is attached to a space of modular symbols, since $J_0(23)$.

```
> IsAttachedToModularSymbols(H);
true
> ModularSymbols(H);
[
    Modular symbols space for Gamma_0(23) of weight 2 and
    dimension 4 over Rational Field
]
> Dimension(H);
```

```
4
> Lattice(H);
Standard Lattice of rank 4 and degree 4
> RealVectorSpace(H);
Full Vector space of degree 4 over Real Field
> VectorSpace(H);
Full Vector space of degree 4 over Rational Field
```

We can also obtain various homologies of A more directly.

```
> IntegralHomology(A);
Standard Lattice of rank 4 and degree 4
> RationalHomology(A);
Full Vector space of degree 4 over Rational Field
> RealHomology(A);
Full Vector space of degree 4 over Real Field
```

Example H142E51

The following example illustrates that a factor of $J_0(37)$ is not presented as something associated to a space of modular symbols. Also notice that the two lattices are different.

```
> J := JZero(37);
> D := Decomposition(J);
> H1 := Homology(D[1]);  H2 := Homology(D[2]);
> IsAttachedToModularSymbols(H1);
false
> Lattice(H1);
Lattice of rank 2 and degree 2
Basis:
( 1  1)
( 1 -1)
> Lattice(H2);
Standard Lattice of rank 2 and degree 2
```

142.4 Homomorphisms

142.4.1 Creation

The commands below create the multiplication by n map on an abelian variety, for any n. Other ways to create homomorphisms are described in other sections of this chapter. These include computing Hecke operators, Atkin-Lehner operators, and computing the full endomorphism and homomorphism rings, and choosing elements in them.

`IdentityMap(A)`

The identity homomorphism from the modular abelian variety A to A.

`ZeroMap(A)`

The zero homomorphism from the modular abelian variety A to A.

`nIsogeny(A, n)`

The multiplication by n isogeny on the modular abelian variety A, where n is a rational number or an integer.

Example H142E52

```
> A := JZero(23);
> IdentityMap(A);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
> ZeroMap(A);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[0 0 0 0]
[0 0 0 0]
[0 0 0 0]
[0 0 0 0]
> nIsogeny(A,3);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[3 0 0 0]
[0 3 0 0]
[0 0 3 0]
[0 0 0 3]
> nIsogeny(A,1/3);
Homomorphism from JZero(23) to JZero(23) (up to isogeny) on integral
homology by:
[1/3   0   0   0]
[  0 1/3   0   0]
[  0   0 1/3   0]
[  0   0   0 1/3]
```

142.4.2 Restriction, Evaluation, and Other Manipulations

Suppose A is an abelian variety, B is an abelian subvariety of A, and ϕ is a homomorphism from A. Then the `Restriction` command computes the restriction of ϕ to B. If, moreover, ϕ is an endomorphism of A (i.e., also has codomain A) and $\phi(B)$ is contained in B, then `RestrictEndomorphism` computes the endomorphism of B induced by ϕ.

If f is a polynomial over the integers or rational numbers and ϕ is an endomorphism of an abelian variety, then the `Evaluate` command computes the endomorphism $f(\phi)$ (if f has denominators, then $f(\phi)$ might only be a morphism on abelian varieties up to isogeny). Together with the `Kernel` command, `Evaluate` is useful for cutting out subvarieties of an abelian variety.

The `SurjectivePart`, `DivideOutIntegers`, and `UniversalPropertyOfCokernel` commands can also all be useful in various contexts for constructing homomorphisms.

`Restriction(phi, B)`

The restriction of the map of abelian varieties ϕ to a morphism from the abelian variety B to the codomain of ϕ, if this obviously makes sense.

`RestrictEndomorphism(phi, B)`

The restriction of the map of abelian varieties ϕ to an endomorphism of the abelian variety B, if this obviously makes sense. If B is not left invariant by ϕ, an error may occur.

`RestrictEndomorphism(phi, i)`

Suppose ϕ is an endomorphism of an abelian variety A and $i : B \to A$ is an injective morphism of abelian varieties such that $i(B)$ is invariant under ϕ. This intrinsic computes the endomorphism ψ of B induced by ϕ. If $i(B)$ is not left invariant by ϕ, an error may occur.

`RestrictionToImage(phi, i)`

Suppose $i : A \to D$ and $\phi : D \to B$ are morphisms of abelian varieties. This intrinsic computes the restriction of ϕ to the image of i. The resulting map is an endomorphism of the image of i.

`Evaluate(f, phi)`

The endomorphism $f(\phi)$ of A, where f is a univariate polynomial and ϕ is an endomorphism of a modular abelian variety A.

`DivideOutIntegers(phi)`

If $\phi : A \to B$ is a homomorphism of abelian varieties, find the largest integer n such that $\psi = (1/n) * \phi$ is also a homomorphism from A to B and return ψ and n.

`SurjectivePart(phi)`

Let $\phi : A \to B$ be a homomorphism. This intrinsic returns the *surjective* homomorphism $\pi : A \to \phi(A)$ induced by ϕ.

`UniversalPropertyOfCokernel(pi, f)`

Uses the universal property of the cokernel to find the unique morphism with a certain property. More precisely, suppose $\pi : B \to C$ is the cokernel of a morphism, so π is surjective with kernel K. Suppose $f : B \to D$ is a morphism whose kernel contains K. By definition of cokernel, there exists a unique morphism $\psi : C \to D$ such that $\pi * \psi = f$. This intrinsic returns ψ. If we only have that the identity component of $\ker(\pi)$ is contained in `ker(f)`, then ψ will only be a homomorphism in the category of abelian varieties up to isogeny, i.e., it will have a nontrivial denominator.

Example H142E53

We use the `Evaluate` and `Kernel` commands to cut out a 2-dimensional abelian subvariety of $J_0(65)$.

```
> J := JZero(65);
> R<x> := PolynomialRing(RationalField());
> T2 := HeckeOperator(J,2);
> Factorization(CharacteristicPolynomial(T2));
[
    <x + 1, 2>,
    <x^2 - 3, 2>,
    <x^2 + 2*x - 1, 2>
]
> phi := Evaluate(x^2-3,T2);
> _,A := Kernel(phi);
> A;
Modular abelian variety of dimension 2 and level 5*13 over Q
```

Example H142E54

This example illustrates the universal property of the cokernel. We decompose $J = J_0(65)$ as a product of simples A, B, and C, of dimensions 1, 2, and 2, respectively. Then we let π be a homomorphism from J onto $B + C$, and f a homomorphism from J onto B. The universal property supplies a morphism ψ from $B + C$ to B such that $\pi * \psi = f$ (i.e., $f(x) = \psi(\pi(x))$ for all x).

```
> J := JZero(65);
> A,B,C := Explode(Decomposition(J));
> pi := NaturalMap(J,B+C);
> IsSurjective(pi);
true
> f := NaturalMap(J,B);
```

```
> psi := UniversalPropertyOfCokernel(pi,f); psi;
Homomorphism from modular abelian variety of dimension 4 to 65B
(up to isogeny) on integral homology by:
 (not printing 8x4 matrix)
> Matrix(psi);
[ 1/2    0    0 -1/2]
[ 1/2    0 -1/2  1/2]
[   0 -1/2  1/2    0]
[   0    0 -1/2    1]
[ 1/2 -1/2 -1/2  1/2]
[ 1/2    0 -1/2  1/2]
[   0  1/2    0    0]
[   0    0  1/2    0]
> pi*psi eq f;    // apply pi then psi is the same as applying f.
true
> Denominator(psi);
2
```

Since ψ has a denominator of 2, it is only a morphism in the category of abelian varieties up to isogeny. This is because only the connected component of the kernel of π is contained in the kernel of f.

```
> G := Kernel(pi); G;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2, 2, 2 ]
> H, K := Kernel(f);
> H;
{ 0 }: finitely generated subgroup of abelian variety with
invariants []
> G subset K;
false
```

Example H142E55

Next we illustrate dividing out by integers, by dividing the 10 out of $10T_3$ acting on $J_0(23)$. Note that this division occurs in the full endomorphism ring, not just the Hecke algebra.

```
> phi := 10*HeckeOperator(JZero(23),3); phi;
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[-10 -20  20   0]
[  0 -30  20 -20]
[ 20 -40  30 -20]
[ 20 -20   0  10]
> DivideOutIntegers(phi);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[-1 -2  2  0]
[ 0 -3  2 -2]
[ 2 -4  3 -2]
[ 2 -2  0  1]
```

```
10
```

Example H142E56

Next we illustrate the restriction commands using factors of $J_0(65)$.

```
> J := JZero(65);
> A := Decomposition(J)[2]; A;
Modular abelian variety 65B of dimension 2, level 5*13 and
conductor 5^2*13^2 over Q
> T := HeckeOperator(J,3);
> Factorization(CharacteristicPolynomial(T));
[
    <x + 2, 2>,
    <x^2 - 2*x - 2, 2>,
    <x^2 - 2, 2>
]
> Restriction(T,A);
Homomorphism from modular abelian variety of dimension 2 to
JZero(65) given on integral homology by:
[-1  0  0  1  0 -1  0  0  0  0]
[-1  3 -1  1 -3 -1  1  2 -1 -2]
[-1  1 -1  3 -1 -1 -1  0  1 -2]
[-1  0  0  3  0 -1 -2  0  2 -2]
> TonA := RestrictEndomorphism(T,A); TonA;
Homomorphism from 65B to 65B given on integral homology by:
[-1  0  0  1]
[-1  3 -1  1]
[-1  1 -1  3]
[-1  0  0  3]
> Factorization(CharacteristicPolynomial(TonA));
[
    <x^2 - 2*x - 2, 2>
]
```

Finally, we illustrate the `SurjectivePart` command on the non-surjective morphism $T_3 + 2$ of $J_0(65)$.

```
> phi := T+2;
> IsSurjective(phi);
false
> pi := SurjectivePart(phi);
> IsSurjective(pi);
true
> Codomain(pi);
Modular abelian variety of dimension 4 and level 5*13 over Q
```

142.4.3 Kernels

The category of abelian varieties is not an abelian category because kernels need not exist in this category. More precisely, if ϕ is a homomorphism $A \to B$ of abelian varieties, then the kernel of ϕ is usually not an abelian variety because it is rarely connected. Instead, the kernel fits into an exact sequence

$$0 \to C \to \ker(\phi) \to G \to 0,$$

where C is an abelian variety and G is a finite group, both defined over the same field as ϕ.

The `ConnectedKernel` command returns C.

The `ComponentGroupOfKernel` command returns G as a subgroup of A/C.

The `Kernel` command returns a finite subgroup of A that maps to G, the abelian variety C, and a map from C into A.

Note that if $C \neq 0$ then the finite group that the `Kernel` command returns is *not canonical*, since it is just some finite group that maps onto G via quotienting out by C. For the canonical component group, use the `ComponentGroupOfKernel` command, which gives a group defined over the same base field as ϕ, whose main drawback is that the component group is not contained in A.

`ComponentGroupOfKernel(phi)`

Component group of $\ker(\phi)$ where ϕ is a homomorphism of abelian varieties.

`ConnectedKernel(phi)`

The connected component C of $\ker(\phi)$ and a morphism from C to the domain of ϕ where ϕ is a homomorphism of abelian varieties.

`Kernel(phi)`

Let ϕ be a homomorphism of abelian varieties. This intrinsic returns a finite subgroup G of A (the domain of ϕ) as a subgroup of an abelian variety, an abelian variety C such that $\ker(\phi)$ equals $f(C) + G$, and an injective map f from C to A. If $C = 0$, then G has the same field of definition as ϕ; otherwise, G is only known to be defined over the algebraic closure.

Example H142E57

The kernel of $T_2 - 3$ on $J_0(65)$ is an extension of an abelian surface by a finite group isomorphic to $(\mathbf{Z}/2\mathbf{Z})^4$.

```
> J := JZero(65);
> T := HeckeOperator(J,2);
> phi := T^2-3;
> ConnectedKernel(phi);
Modular abelian variety of dimension 2 and level 5*13 over Q
Homomorphism from modular abelian variety of dimension 2 to
JZero(65) given on integral homology by:
```

```
[ 1  0  0  0  0  1 -1  0  1 -1]
[ 0  1  0  0 -1  0  0  1  0 -1]
[ 0  0  1  0  0  0 -1  1  1 -1]
[ 0  0  0  1  0  0 -1  0  1 -1]
> Kernel(phi);
Finitely generated subgroup of abelian variety with
invariants [ 2, 2, 2, 2 ]
Modular abelian variety of dimension 2 and level 5*13 over Q
Homomorphism from modular abelian variety of dimension 2 to
JZero(65) given on integral homology by:
[ 1  0  0  0  0  1 -1  0  1 -1]
[ 0  1  0  0 -1  0  0  1  0 -1]
[ 0  0  1  0  0  0 -1  1  1 -1]
[ 0  0  0  1  0  0 -1  0  1 -1]
> G := ComponentGroupOfKernel(phi); G;
Finitely generated subgroup of abelian variety with
invariants [ 2, 2, 2, 2 ]
> FieldOfDefinition(G);
Rational Field
```

Though G is defined over $\mathbf{Q}$, the ambient variety of G is the quotient of $J_0(65)$ by the two dimensional connected component of the kernel of $T_2 - 3$.

```
> AmbientVariety(G);
Modular abelian variety of dimension 3 and level 5*13 over Q
```

On the other hand, the group returned by the `Kernel` command is a subgroup of $J_0(65)$.

```
> H, C := Kernel(phi);
> H;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2 ]
> FieldOfDefinition(H);
Algebraically closed field with no variables
> AmbientVariety(H);
Modular abelian variety JZero(65) of dimension 5 and level 5*13
over Q
```

142.4.4 Images

These commands compute the image of a modular abelian variety or a finite group under a homomorphism ϕ of modular abelian varieties. Using `@@`, one can also compute a group lifting a given group. Note that this lift is not canonical unless ϕ has finite kernel, in which case `G@@phi` is the full inverse image of G.

`A @ phi`

`phi(A)`

The image of the abelian variety A under the map ϕ of abelian varieties, if this makes sense, i.e., if A is the domain of ϕ, or A has dimension 0, or one of the embeddings of A has codomain equal to the domain of ϕ.

`G @ phi`

`phi(G)`

The image of the subgroup G of an abelian variety under the map ϕ of abelian varieties, if this makes sense. If A is the ambient variety of G, then this makes sense if A is the domain of ϕ, or A has dimension 0, or one of the embeddings of A has codomain equal to the domain of ϕ.

`Image(phi)`

The image C of the morphism ϕ of abelian varieties, which is a modular abelian subvariety contained in the codomain of ϕ, a morphism from C to the codomain of ϕ, and a surjective morphism from the domain of ϕ to C.

`G @@ phi`

A finite group whose image under ϕ, a morphism of abelian varieties, is equal to the subgroup G of a modular abelian variety, if possible. If ϕ has finite kernel, then this is the exact inverse image of G under ϕ. If not, then this is a group generated by a choice of torsion inverse image for each generator of G, which may not be canonical.

Example H142E58

The image of $J_0(37)$ under T_2 is an elliptic curve.

```
> J := JZero(37);
> phi := HeckeOperator(J,2);
> phi(J);
Modular abelian variety of dimension 1 and level 37 over Q
> Image(phi);
Modular abelian variety of dimension 1 and level 37 over Q
Homomorphism from modular abelian variety of dimension 1 to
JZero(37) given on integral homology by:
[ 1 -1  1  0]
[ 1 -1 -1  1]
Homomorphism from JZero(37) to modular abelian variety of dimension
1 given on integral homology by:
[ 0 -1]
[ 1  0]
[-1  1]
```

```
[ 0  0]
```

The Hecke operator T_2 maps the 2-torsion subgroup of $J_0(37)$ maps onto the 2-torsion subgroup of the image of T_2.

```
> G := nTorsionSubgroup(J,2); G;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2 ]
> phi(G);
Finitely generated subgroup of abelian variety with invariants
[ 2, 2 ]
```

The homomorphism $T_2 - 1$ is surjective, and the inverse image of the 2-torsion is a subgroup isomorphic to $(\mathbf{Z}/2\mathbf{Z})^2 \times (\mathbf{Z}/6\mathbf{Z})^2$.

```
> IsSurjective(phi-1);
true
> psi := phi-1;
> H := G@@(psi); H;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 6, 6 ]
> psi(H);
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2 ]
> H := G@@psi; H;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 6, 6 ]
> psi(H);
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2 ]
```

142.4.5 Cokernels

`Cokernel(phi)`

The cokernel of the morphism ϕ of abelian varieties and a morphism from the codomain of ϕ to the cokernel.

Example H142E59

We compute a 2-dimensional quotient of the 3-dimensional abelian variety $J_0(33)$ using the Hecke operator T_2 and the `Cokernel` command.

```
> J := JZero(33);
> T := HeckeOperator(J,2);
> Factorization(CharacteristicPolynomial(T));
[
    <x - 1, 2>,
    <x + 2, 4>
```

```
]
> phi := T + 2;
> A, pi := Cokernel(phi);
> A;
Modular abelian variety of dimension 2 and level 3*11 over Q
> pi;
Homomorphism from JZero(33) to modular abelian variety of dimension
2 (not printing 6x4 matrix)
```

142.4.6 Matrix Structure

Homomorphisms are stored and worked with internally using the linear maps they induce on homology. The commands below provide access to those matrices.

`Matrix(phi)`

The matrix on the chosen basis of rational homology that defines the morphism ϕ of abelian varieties.

`Eltseq(phi)`

The `Eltseq` of the underlying matrix that defines the morphism ϕ of abelian varieties. This is a sequence of integers or rational numbers.

`Ncols(phi)`

The number of columns of the matrix which defines the morphism ϕ of abelian varieties. This is also the dimension of the homology of the codomain of ϕ.

`Nrows(phi)`

The number of rows of the matrix that defines the morphism ϕ of abelian varieties. This is also the dimension of the homology of the domain of ϕ.

`Rows(phi)`

The sequence rows of the matrix that defines the morphism ϕ of abelian varieties.

`IntegralMatrix(phi)`

The matrix which defines the morphism ϕ of abelian varieties, written with respect to integral homology.

`IntegralMatrixOverQ(phi)`

The matrix which defines the morphism ϕ of abelian varieties, written with respect to integral homology.

`RealMatrix(phi)`

The matrix which defines the morphism ϕ of abelian varieties, written with respect to real homology.

Example H142E60

The following example demonstrates each of the commands for the Hecke operator T_2 on $J_0(23)$.

```
> phi := HeckeOperator(JZero(23),2); phi;
Homomorphism T2 from JZero(23) to JZero(23) given on integral homology by:
[ 0  1 -1  0]
[ 0  1 -1  1]
[-1  2 -2  1]
[-1  1  0 -1]
> Eltseq(phi);
[ 0, 1, -1, 0, 0, 1, -1, 1, -1, 2, -2, 1, -1, 1, 0, -1 ]
> IntegralMatrix(phi);
[ 0  1 -1  0]
[ 0  1 -1  1]
[-1  2 -2  1]
[-1  1  0 -1]
> Parent($1);
Full RMatrixSpace of 4 by 4 matrices over Integer Ring
> IntegralMatrixOverQ(phi);
[ 0  1 -1  0]
[ 0  1 -1  1]
[-1  2 -2  1]
[-1  1  0 -1]
> Parent($1);
Full Matrix Algebra of degree 4 over Rational Field
> Matrix(phi);
[ 0  1 -1  0]
[ 0  1 -1  1]
[-1  2 -2  1]
[-1  1  0 -1]
> Ncols(phi);
4
> Nrows(phi);
4
> RealMatrix(phi);
[   0    1   -1    0]
[   0    1   -1    1]
[  -1  2/1 -2/1    1]
[  -1    1    0   -1]
> Parent($1);
Full KMatrixSpace of 4 by 4 matrices over Real Field
> Rows(phi);
[
    ( 0  1 -1  0),
    ( 0  1 -1  1),
    (-1  2 -2  1),
    (-1  1  0 -1)
```

```
]
```

142.4.7 Arithmetic

MAGMA supports many standard arithmetic operations with homomorphisms of abelian varieties, including composition, addition, subtraction, and exponentiation.

`Inverse(phi)`

The inverse of ϕ and an integer d such that $d*\phi^{-1}$ is a morphism of abelian varieties. More precisely, if ϕ is an isogeny, then the inverse of ϕ is a morphism in the category of abelian varieties up to isogeny. This intrinsic returns such a morphism or the actual inverse of ϕ if ϕ has degree 1.

`phi * psi`

The composition of the homomorphisms ϕ and ψ of abelian varieties.

`a * phi`

The product of the rational number or integer a and the homomorphism ϕ of abelian varieties. The result might only be a homomorphism, up to isogeny.

`phi * psi`

The product of the matrix that defines the morphism ϕ of abelian varieties and the matrix ψ.

`psi * phi`

The product of the matrix ψ and the matrix that defines the morphism ϕ of abelian varieties.

`phi ^ n`

The n-fold composition ϕ^n of the endomorphism ϕ of an abelian variety with itself. If $n = -1$, then this is the inverse of ϕ, in which case ϕ must be an isogeny or an error occurs.

`phi + psi`

The sum of the homomorphisms ϕ and ψ

`n + phi`

The sum of the morphism multiplication by the rational number or integer n and the endomorphism ϕ of an abelian variety.

`phi + n`

The sum of the endomorphism ϕ of an abelian variety and the endomorphism multiplication-by-n where n is an integer.

`phi + psi`

The sum of the matrix that defines the morphism ϕ of abelian varieties and the matrix ψ.

`psi + phi`

The sum of the matrix ψ and the matrix which defines the morphism ϕ of abelian varieties.

`phi - psi`

The difference between the homomorphisms ϕ and ψ of abelian varieties.

`n - phi`

The difference between the morphism multiplication-by-n, where n is a rational number or an integer, and the endomorphism ϕ of an abelian variety.

`phi - n`

The difference between the endomorphism ϕ of an abelian variety and the endomorphism multiplication-by-n where n is a rational number or an integer.

`phi - psi`

The difference between the matrix that defines the morphism ϕ of abelian varieties and the matrix ψ.

`psi - phi`

The difference between the matrix ψ and the matrix which defines the morphism ϕ of abelian varieties.

Example H142E61

We illustrate several arithmetic operations use the Hecke operators T_2 and T_3 on $J_0(23)$.

```
> J := JZero(23);
> phi := HeckeOperator(J,2);
> psi := HeckeOperator(J,3);
> Matrix(phi)*Matrix(psi);
[-2  1 -1  0]
[ 0 -1 -1  1]
[-1  2 -4  1]
[-1  1  0 -3]
> phi*psi;
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[-2  1 -1  0]
[ 0 -1 -1  1]
[-1  2 -4  1]
[-1  1  0 -3]
> Inverse(phi);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
```

```
[ 1  1 -1  0]
[ 0  2 -1  1]
[-1  2 -1  1]
[-1  1  0  0]
1
> 1/3*phi;
Homomorphism from JZero(23) to JZero(23) (up to isogeny) on integral
homology by:
[   0  1/3 -1/3    0]
[   0  1/3 -1/3  1/3]
[-1/3  2/3 -2/3  1/3]
[-1/3  1/3    0 -1/3]
> 2+phi;
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[ 2  1 -1  0]
[ 0  3 -1  1]
[-1  2  0  1]
[-1  1  0  1]
> phi+psi;
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[-1 -1  1  0]
[ 0 -2  1 -1]
[ 1 -2  1 -1]
[ 1 -1  0  0]
> phi^4;
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[ 2 -3  3  0]
[ 0 -1  3 -3]
[ 3 -6  8 -3]
[ 3 -3  0  5]
1
> phi^(-4);
Homomorphism from JZero(23) to JZero(23) given on integral homology by:
[ 5  3 -3  0]
[ 0  8 -3  3]
[-3  6 -1  3]
[-3  3  0  2]
1
```

142.4.8 Polynomials

The polynomial commands compute the characteristic and minimal polynomials of endomorphisms of modular abelian varieties. Each command requires that the input homomorphism be a genuine homomorphism (not just a morphism up to isogeny). The same computation could be done by first extracting the matrix of the endomorphism and using the analogous command on the matrix. However, in some special cases there may be special techniques which can be used to do the computation more quickly, so there can be some advantage to using the commands below.

`CharacteristicPolynomial(phi)`

The characteristic polynomial of the endomorphism ϕ of an abelian variety. We can sometimes use extra information about ϕ (e.g., how it was constructed, Deligne bounds) to gain an answer faster.

`FactoredCharacteristicPolynomial(phi)`

The factorization of the characteristic polynomial of the endomorphism ϕ of an abelian variety. We can sometimes use extra information about ϕ (e.g., how it was constructed, Deligne bounds), to gain an answer faster. These factored polynomials are cached.

`MinimalPolynomial(phi)`

The minimal polynomial of the endomorphism ϕ of an abelian variety. We can sometimes use extra information about ϕ (e.g., how it was constructed, Deligne bounds), to gain an answer faster.

Example H142E62

This example illustrates each of the polynomial commands for the Hecke operator T_2 on $J_0(66)$.

```
> J := JZero(66);
> phi := HeckeOperator(J,2);
> R<x> := PolynomialRing(RationalField());
> CharacteristicPolynomial(phi);
x^18 + 4*x^17 + 8*x^16 + 8*x^15 + 6*x^14 - 20*x^12 - 56*x^11 -
    55*x^10 - 4*x^9 + 36*x^8 + 48*x^7 + 104*x^6 + 128*x^5 -
    32*x^4 - 192*x^3 - 112*x^2 + 64*x + 64
> CharacteristicPolynomial(Matrix(phi));
x^18 + 4*x^17 + 8*x^16 + 8*x^15 + 6*x^14 - 20*x^12 - 56*x^11 -
    55*x^10 - 4*x^9 + 36*x^8 + 48*x^7 + 104*x^6 + 128*x^5 -
    32*x^4 - 192*x^3 - 112*x^2 + 64*x + 64
> FactoredCharacteristicPolynomial(phi);
[
    <x - 1, 4>,
    <x + 1, 2>,
    <x^2 - x + 2, 2>,
    <x^2 + 2*x + 2, 4>
```

```
]
> MinimalPolynomial(phi);
x^6 + x^5 + x^4 + x^3 + 2*x^2 - 2*x - 4
> Factorization(MinimalPolynomial(phi));
[
    <x - 1, 1>,
    <x + 1, 1>,
    <x^2 - x + 2, 1>,
    <x^2 + 2*x + 2, 1>
]
```

142.4.9 Invariants

It is possible to retrieve the domain and codomain of homomorphisms between abelian varieties. The degrees of such homomorphisms can be computed. Denominators of homomorphisms can be determined and cleared. A field over which a homomorphism is defined is available. The rank, nullity and trace of a homomorphism can also be computed.

`Domain(phi)`

The domain of the morphism ϕ of abelian varieties.

`Codomain(phi)`

The codomain of the morphism ϕ of abelian varieties.

`Degree(phi)`

The degree of the morphism ϕ of abelian varieties, i.e. the cardinality of the kernel of ϕ. If the kernel of ϕ is not finite, then the degree is defined to be 0.

`Denominator(phi)`

Given a morphism ϕ of abelian varieties return the smallest positive integer n such that $n * \phi$ is a homomorphism. This is also the denominator of the matrix that defines ϕ.

`ClearDenominator(phi)`

Return the morphism $n * \phi$ where ϕ is a morphism of abelian varieties and n is positive and the smallest such that $n * \phi$ is a genuine homomorphism.

`FieldOfDefinition(phi)`

A field of definition of the morphism ϕ of abelian varieties. This is only some field over which ϕ is defined, it is not guaranteed to be minimal.

`Nullity(phi)`

The dimension of the kernel of the morphism ϕ of abelian varieties.

`Rank(phi)`

The dimension of the kernel of the morphism ϕ of abelian varieties.

`Trace(phi)`

The trace of any matrix which represents the action of the morphism ϕ of abelian varieties on integral homology. For example, the trace of multiplication by n on A is $2n\dim(A)$.

Example H142E63

```
> phi := NaturalMap(JZero(11),JZero(33));
> Codomain(phi);
Modular abelian variety JZero(33) of dimension 3 and level 3*11 over Q
> Domain(phi);
Modular abelian variety JZero(11) of dimension 1 and level 11 over Q
> Degree(phi);
1
> Denominator(1/5*phi);
5
> FieldOfDefinition(phi);
Rational Field
> Nullity(phi);
0
> Rank(phi);
1
> Trace(HeckeOperator(JZero(33),2));
-6
> Trace(nIsogeny(JZero(33),5));
30
```

142.4.10 Predicates

A range of predicates are provided allowing a morphism to be checked for whether it is an actual morphism or not, whether its kernel is finite, and whether it is injective or surjective or an isogeny. There are also commands for testing equality of homomorphisms and inclusion in a list.

`IsMorphism(phi)`

Return `true` if and only if the map ϕ between abelian varieties is a morphism in the category of abelian varieties (not just in the category of abelian varieties up to isogeny).

`OnlyUpToIsogeny(phi)`

Return `true` if the map ϕ between abelian varieties is not a homomorphism, but $n*\phi$ is a homomorphism for some positive integer n, i.e., ϕ is only a homomorphism in the category of abelian varieties up to isogeny.

`HasFiniteKernel(phi)`

Return `true` if the kernel of the homomorphism ϕ of abelian varieties is finite.

`IsInjective(phi)`

Return `true` if the map ϕ between abelian varieties is an injective homomorphism.

`IsSurjective(phi)`

Return `true` if the homomorphism ϕ between abelian varieties is surjective.

`IsEndomorphism(phi)`

Return `true` if the morphism ϕ between abelian varieties is an endomorphism, i.e., the domain and codomain of ϕ are equal.

`IsInteger(phi)`

Return `true` if the morphism ϕ between abelian varieties is multiplication by n for some integer n. If so, returns that n as well, otherwise, returns `false`.

`IsIsogeny(phi)`

Return `true` if the morphism ϕ between abelian varieties is a surjective homomorphism with finite kernel. Note that this definition differs from the one in Silverman's books on elliptic curves, agrees with the one in Milne's articles, and is an equivalence relation.

`IsIsomorphism(phi)`

Return `true` if the morphism ϕ between abelian varieties is an isomorphism of abelian varieties.

`IsOptimal(phi)`

Return `true` if the morphism ϕ between abelian varieties is an optimal quotient map, i.e., ϕ is surjective and has connected kernel.

`IsHeckeOperator(phi)`

Returns `true` if the morphism ϕ between abelian varieties was computed using the `HeckeOperator` command, and the argument n passed to the `HeckeOperator` command when ϕ was created, otherwise `false`.

`IsZero(phi)`

Return `true` if the morphism ϕ of abelian varieties is the zero morphism.

`phi eq psi`

Return `true` if the two homomorphisms of abelian varieties ϕ and ψ are equal.

`n eq phi`

`phi eq n`

Return `true` if the morphism ϕ between abelian varieties is equal to multiplication by the integer n.

`phi in X`

Return `true` if the morphism ϕ between abelian varieties is one of the homomorphisms in the list X of homomorphisms.

Example H142E64

```
> phi := HeckeOperator(JZero(65),2)-1;
> HasFiniteKernel(phi);
true
> IsEndomorphism(phi);
true
> IsHeckeOperator(phi);
false 0
> IsInjective(phi);
false
> IsInteger(phi);
false
> IsIsogeny(phi);
true
> IsIsomorphism(phi);
false
> IsMorphism(phi);
true
> IsMorphism(1/2*phi);
false
> IsSurjective(phi);
true
> IsZero(phi);
false
> OnlyUpToIsogeny(phi);
false
> 2 eq phi;
false
> phi eq 2;
false
> 2 eq nIsogeny(JZero(65),2);
true
> phi eq nIsogeny(JZero(65),2);
```

```
false
> phi in [* phi, nIsogeny(JZero(65),2) *];
true
> IsIsomorphism(NaturalMap(JZero(11),JOne(11)));
false
> IsIsomorphism(NaturalMap(JZero(11)^2,JZero(22)));
false
> IsIsomorphism(NaturalMap(JZero(11),JZero(11)));
true
```

142.5 Endomorphism Algebras and Hom Spaces

142.5.1 Creation

Let A and B be modular abelian varieties. Then one can create the finite-rank free abelian group of homomorphisms from A to B or the vector space of homomorphisms in the category of abelian varieties up to isogeny. It is also possible to create the endomorphism algebra of an abelian variety. Such creations do non-trivial computations until further information is requested of the structures, such as rank or basis. In particular, such creations do not compute $\mathrm{Hom}(A, B)$.

`Hom(A, B)`

`Hom(A, B, oQ)`

Return the group of homomorphisms from the abelian variety A to the abelian variety B. If the argument oQ is given and is `true`, return the vector space generated by such homomorphisms.

`End(A)`

`End(A, oQ)`

The endomorphism ring of the abelian variety A. If oQ is given and is `true`, return the endomorphism algebra.

`BaseExtend(H, R)`

The space $H \otimes R$, where H is a group of homomorphisms of a modular abelian variety and R is the rational numbers or integers. When $R = \mathbf{Q}$, this is the space of homomorphisms in the category of abelian varieties up to isogeny.

`HeckeAlgebra(A)`

The Hecke algebra associated to the abelian variety A, which is a commutative subring of $\mathrm{End}(A)$ generated by Hecke operators. For an abelian variety attached to modular symbols, this is the algebra induced by Hecke operators acting on modular symbols (homology). For a general abelian variety, let $\pi : J \to A$ and $e : A \to J$ be the modular parameterization and embedding of A. Then this is the ring of endomorphisms obtained by pulling back the Hecke algebra on J to A using π and e, i.e., it is $e * T * \pi$, where T is the Hecke ring of J. For example, since $J_1(N)$ is represented as a quotient of $J_0(N)$, the Hecke algebra is not what you might expect but may differ by some finite index.

Example H142E65

```
> A := JZero(11); B := JZero(33);
> Hom(A,B);
Group of homomorphisms from JZero(11) to JZero(33)
> Hom(A,B,true);
Group of homomorphisms from JZero(11) to JZero(33) in the category of
abelian varieties up to isogeny
> End(A);
Group of homomorphisms from JZero(11) to JZero(11)
> End(A,true);
Group of homomorphisms from JZero(11) to JZero(11) in the category of
abelian varieties up to isogeny
> BaseExtend(Hom(A,B),RationalField());
_Q: Group of homomorphisms from JZero(11) to JZero(33) in the category
of abelian varieties up to isogeny
> HeckeAlgebra(A);
HeckeAlg(JZero(11)): Group of homomorphisms from JZero(11) to JZero(11)
```

142.5.2 Subgroups and Subrings

Subgroups of $\mathrm{Hom}(A, B)$ can also be formed as well as subrings of $\mathrm{End}(A, B)$. Magma provides the computation of saturations of subgroups of $\mathrm{Hom}(A, B)$ where A and B are abelian varieties.

`Subgroup(X)`

The group of homomorphisms from the abelian variety A to the abelian variety B generated by the homomorphisms of abelian varieties in the sequence X.

`Subgroup(X, oQ : parameters)`

`IsBasis` BoolElt *Default :* `false`

The group of homomorphisms generated by homomorphisms of abelian varieties in the sequence X. If the parameter `IsBasis` is `true`, then it is assumed that the elements of X are a basis for their span. If the argument oQ is `true`, return the vector space generated by such homomorphisms.

`Subring(X)`

`Subring(X, oQ)`

The ring of endomorphisms of a modular abelian variety generated by the endomorphisms in the sequence X. It does not need to contain unity. If the argument oQ is given and is `true`, return the algebra generated by such endomorphisms.

`Subring(phi)`

The ring of endomorphisms generated by the endomorphism ϕ of an abelian variety. It does not need to contain unity.

`Saturation(H)`

The saturation of the group H of homomorphisms of abelian varieties in all homomorphisms. Suppose A and B are abelian varieties and H is a subgroup of $\mathrm{Hom}(A, B)$. Then $\mathrm{Hom}(A, B)$ is a free $\mathbf{Z}$-module, and the saturation of H in $\mathrm{Hom}(A, B)$ is a group H' that contains H with finite index such that the quotient of `Hom(A,B)` by H' is torsion free.

`RingGeneratedBy(H)`

The ring of endomorphisms generated by the endomorphisms in the group H of homomorphisms between abelian varieties.

Example H142E66

In this example we use the `Saturation` command to find the integers N up to 60 such that the Hecke algebra of $J_0(N)$ is not saturated in the full ring of endomorphisms.

```
> function ind(N)
>    H := HeckeAlgebra(JZero(N));
>    return Index(Saturation(H),H);
> end function;
> for N in [2..60] do
>    i := ind(N);
>    if i gt 1 then N, i; end if;
> end for;
44 2
46 2
54 3
56 2
```

```
60 2
```

Note that multiplicity one fails at 3 for $J_0(54)$. It might be interesting to find a precise relationship between failure of multiplicity one and the index of the Hecke algebra T in its saturation.

Example H142E67

This example illustrates constructing a subgroup.

```
> J := JZero(33);
> E := End(J); E;
Group of homomorphisms from JZero(33) to JZero(33)
> H := Subgroup([E.1, E.3]); H;
Group of homomorphisms from JZero(33) to JZero(33)
> Rank(H);
2
```

Example H142E68

We compute the subring generated by T_2 on $J_0(100)$.

```
> T2 := HeckeOperator(JZero(100),2);
> R := Subring(T2); R;
Group of homomorphisms from JZero(100) to JZero(100)
> Rank(R);
3
```

Example H142E69

The Hecke operator T_2 and the main Atkin-Lehner involution together generate a commutative ring of rank 10 over $\mathbf{Z}$.

```
> J := JZero(100);
> T2 := HeckeOperator(J,2);W := AtkinLehnerOperator(J,100);
> R := Subring([End(J) | T2,W]);
> Dimension(R);
10
> Dimension(End(J));
13
> IsRing(R);
true
> IsCommutative(R);
false
```

142.5.3 Pullback and Pushforward of Hom Spaces

A homomorphism of abelian varieties induces a map from one space of homomorphisms into another. The three commands below compute the image of such maps.

`Pullback(H, phi)`

Given a space of homomorphisms H in $\mathrm{Hom}(A, B)$ and a morphism $\phi : B \to C$, compute the image of H in $\mathrm{Hom}(A, C)$ via the map that sends f to $f * \phi$.

`Pullback(phi, H)`

Given a space of homomorphisms H in $\mathrm{Hom}(A, B)$ and a morphism $\phi : C \to A$, compute the image of H in $\mathrm{Hom}(C, B)$ via the map that sends f to $\phi * f$.

`Pullback(phi, H, psi)`

Suppose H is a space of homomorphisms $A \to B$ and $\phi : C \to A$ and $\psi : B \to D$. Then this intrinsic computes and returns the ring of homomorphisms of A of the form $\phi * f * \psi$, where f is in H.

Example H142E70

```
> H := Hom(JZero(11),JZero(22));
> phi := NaturalMap(JZero(22),JZero(33));
> psi := NaturalMap(JZero(33),JZero(11));
> Pullback(H,phi);
Group of homomorphisms from JZero(11) to JZero(33)
> Pullback(psi,H);
Group of homomorphisms from JZero(33) to JZero(22)
> Pullback(psi,H,phi);
Group of homomorphisms from JZero(33) to JZero(33)
```

142.5.4 Arithmetic

The + and `meet` command compute the sum and intersection of two subgroups of $\mathrm{Hom}(A, B)$.

`H1 + H2`

The subgroup generated by H_1 and H_2, where H_1 and H_2 are assumed to both be subgroups of $\mathrm{Hom}(A, B)$, for abelian varieties A and B.

`H1 meet H2`

The intersection of H_1 and H_2, where H_1 and H_2 are assumed to both be subgroups of $\mathrm{Hom}(A, B)$, for abelian varieties A and B.

Example H142E71

```
> J := JZero(33);
> E := End(J);
> H1 := HeckeAlgebra(J); H1;
HeckeAlg(JZero(33)): Group of homomorphisms from JZero(33) to JZero(33)
> H2 := Subgroup([E.1,E.2]); H2;
Group of homomorphisms from JZero(33) to JZero(33)
> Dimension(E);
5
> Dimension(H1);
3
> Dimension(H2);
2
> Dimension(H1 meet H2);
1
> Dimension(H1 + H2);
4
```

142.5.5 Quotients

If H_1 and H_2 are subspaces of $\mathrm{Hom}(A, B)$ then the index of H_1 in H_2 can be computed. The quotient H_2/H_1 can also be taken.

`Index(H2, H1)`

The index of H_1 in H_2, where H_1 and H_2 are both subgroups of $\mathrm{Hom}(A, B)$, for abelian varieties A and B. If H_1 is contained in H_2, this is just the cardinality of H_2/H_1, or 0 if this cardinality is infinite. If H_1 is not contained in H_2, then the index is by definition the generalized lattice index $[H_1 + H_2 : H_1]/[H_1 + H_2 : H_2]$, assuming the denominator is nonzero, i.e., that H_2 has finite index in $H_1 + H_2$ (an error occurs if H_2 does not have finite index in $H_1 + H_2$).

`Quotient(H2, H1)`

`H2 / H1`

The abelian group quotient H_2/H_1, a map from H_2 to this quotient, and a lifting map from this quotient to H_2, where H_1 and H_2 are subgroups of $\mathrm{Hom}(A, B)$ and A and B are abelian varieties.

Example H142E72

We define the subgroup of $\mathrm{End}(J_0(54))$ generated by T_1, T_2, T_3, and T_4, find that it has infinite index in the full Hecke algebra, and compute the quotient, which is $\mathbf{Z}$.

```
> J := JZero(54);
> T := HeckeAlgebra(J);
> Dimension(T);
4
> S := Subgroup([HeckeOperator(J,n) : n in [1..4]]);
> Dimension(S);
3
> Index(T,S);
0
> Quotient(T,S);
Abelian Group isomorphic to Z
Defined on 1 generator (free)
Mapping from: HomModAbVar: T to Abelian Group isomorphic to Z
Defined on 1 generator (free) given by a rule [no inverse]
Mapping from: Abelian Group isomorphic to Z
Defined on 1 generator (free) to HomModAbVar: T given by a rule
[no inverse]
> G := T/S; G;
Abelian Group isomorphic to Z
Defined on 1 generator (free)
```

We compute the subgroup generated by T_3, T_4, T_5, and T_{10}, and find that it has index 6 in its saturation.

```
> S := Subgroup([HeckeOperator(J,n) : n in [3,4,5,10]]);
> Sat := Saturation(S);
> Index(Sat,S);
6
> Index(S,Sat);
1/6
> Invariants(Sat/S);
[ 6 ]
```

142.5.6 Invariants

Domain and codomain of the homomorphisms in a space of homomorphisms of abelian varieties can both be retrieved from the space as well as a field which all homomorphisms in a space are defined over.

A possibly time consuming computation is that of the discriminant of a space of homomorphisms. Discriminants of Hecke algebras are particularly interesting to compute because they are closely related to congruences between eigenforms.

`Domain(H)`

The domain of the homomorphisms in the group H of homomorphisms between modular abelian varieties.

`Codomain(H)`

The codomain of the homomorphisms in the group H of homomorphisms between modular abelian varieties.

`FieldOfDefinition(H)`

A field over which all homomorphisms in the group H of homomorphisms of abelian varieties are defined. It is not guaranteed to be minimal.

`Discriminant(H)`

The discriminant of the space H of homomorphisms of abelian varieties with respect to the trace pairing matrix. This is the trace of endomorphisms acting on homology, not left multiplication on themselves, so, e.g., the discriminant of the Hecke algebra will be 2^d times as big as it would be otherwise (if the sign is 0), where d is the dimension of A. If H is over $\mathbf{Q}$, this function returns the discriminant of the lattice of elements in H that are homomorphisms.

Example H142E73

```
> H := Hom(JZero(11),JZero(33));
> Domain(H);
Modular abelian variety JZero(11) of dimension 1 and level 11 over Q
> Codomain(H);
Modular abelian variety JZero(33) of dimension 3 and level 3*11 over
Q
> FieldOfDefinition(H);
Rational Field
> A := BaseExtend(JZero(11),ComplexField());
> H := End(A);
> FieldOfDefinition(H);
Complex Field
```

Though $H = \mathbf{Z}$, so the discriminant of the abstract ring H is 1, the discriminant of the trace pairing of H acting on homology is 2:

```
> Discriminant(H);
2
[2]
```

The prime $p = 389$ is the only known example where p divides the discriminant of the Hecke algebra of $J_0(p)$.

```
> T := HeckeAlgebra(JZero(389,2 : Sign := +1));
> d := Discriminant(T);
> J := JZero(389,2,+1);
```

```
> T := HeckeAlgebra(J);
> d := Discriminant(T);
> d mod 389;
0
> Factorization(d);
[ <2, 53>, <3, 4>, <5, 6>, <31, 2>, <37, 1>, <389, 1>, <3881, 1>,
<215517113148241, 1>, <477439237737571441, 1> ]
```

All the "action" at 389 comes from the 20-dimensional simple factor.

```
> A := J(5); A;
Modular abelian variety 389E of dimension 20, level 389 and
conductor 389^20 over Q with sign 1
> Factorization(Discriminant(HeckeAlgebra(A)));
[ <2, 98>, <5, 41>, <389, 1>, <215517113148241, 1>,
<477439237737571441, 1> ]
```

142.5.7 Structural Invariants

The following commands provide access to a basis and generators for spaces of homomorphisms.

`Basis(H)`

`Generators(H)`

A basis for the space H of homomorphisms of abelian varieties.

`Dimension(H)`

`Rank(H)`

The rank of the space H of homomorphisms of abelian varieties as a $\mathbf{Z}$-module or $\mathbf{Q}$-vector space.

`Ngens(H)`

The number of generators of the space H of homomorphisms of abelian varieties.

`H . i`

The ith generator of the space H of homomorphisms of abelian varieties.

Example H142E74

The following example illustrates each of the commands for $\mathrm{Hom}(J_0(11), J_0(33))$.

```
> H := Hom(JZero(11),JZero(33));
> Basis(H);
[
    Homomorphism from JZero(11) to JZero(33) given on integral homology
    by:
    [ 0  2 -1 -2  0  1]
    [ 1  0 -1 -1  1  1],
    Homomorphism from JZero(11) to JZero(33) given on integral homology
    by:
    [ 1  0 -2  2 -3  0]
    [ 1 -1  0  1 -2  1]
]
> Dimension(H);
2
> Ngens(H);
2
> Rank(H);
2
> H.1;
Homomorphism from JZero(11) to JZero(33) given on integral homology by:
[ 0  2 -1 -2  0  1]
[ 1  0 -1 -1  1  1]
```

142.5.8 Matrix and Module Structure

The following commands associate lattices, vector spaces, matrix algebras, and matrix spaces to subspaces H of $\mathrm{Hom}(A, B)$.

`Lattice(H)`

`VectorSpace(H)`

A lattice or vector space with basis obtained from the components of the matrices of a basis for the space H of homomorphisms of abelian varieties. This free **Z**-module is constructed from the `Eltseq`s of the all of the basis elements of H.

`MatrixAlgebra(H)`

The matrix algebra generated by the underlying matrices of all elements in the space H of homomorphisms of abelian varieties, acting on homology.

`RMatrixSpace(H)`

The matrix space whose basis is the generators for the space H of homomorphisms of abelian varieties.

`RModuleWithAction(H)`

A module over the ring R generated by the space H of homomorphisms of abelian varieties equipped with the action of H, where H must be a ring of endomorphisms.

`RModuleWithAction(H, p)`

A module over H tensor F_p equipped with the action of H tensor F_p, where H must be a ring of endomorphisms of an abelian variety that has not been tensored with $\mathbf{Q}$.

Example H142E75

We first demonstrate some of the commands with $\mathrm{Hom}(J_0(11), J_0(33))$.

```
> H := Hom(JZero(11),JZero(33));
> Lattice(H);
Lattice of rank 2 and degree 12
Basis:
( 0  2 -1 -2  0  1  1  0 -1 -1  1  1)
( 1  0 -2  2 -3  0  1 -1  0  1 -2  1)
> RMatrixSpace(H);
RMatrixSpace of 2 by 6 matrices and dimension 2 over Integer Ring
> RMatrixSpace(BaseExtend(H,RationalField()));
KMatrixSpace of 2 by 6 matrices and dimension 2 over Rational
Field
> VectorSpace(H);
Vector space of degree 12, dimension 2 over Rational Field
User basis:
( 1  0 -2  2 -3  0  1 -1  0  1 -2  1)
( 0 -2  1  2  0 -1 -1  0  1  1 -1 -1)
```

Next we consider the endomorphism algebra of $J_0(22)$.

```
> H := End(JZero(22));
> MatrixAlgebra(H);
Matrix Algebra of degree 4 with 4 generators over Integer Ring
> RModuleWithAction(H);
RModule(IntegerRing(), 4)
Action:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
[ 0  1  0  1]
[ 0  0  0  0]
[ 1  0  1 -1]
[ 0  1  0  1]
[ 0  1  0 -1]
[ 0  1  0  0]
[-1  2 -1  1]
```

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

Example H142E76

The following example illustrates that `MatrixAlgebra` computes the algebra generated by H, even if H is not itself an algebra.

```
> H := Subgroup([HeckeOperator(JZero(33),2)]); H;
Group of homomorphisms from JZero(33) to JZero(33)
> A := MatrixAlgebra(H); A;
Matrix Algebra of degree 6 with 1 generator over Integer Ring
> Dimension(A);
2
> Dimension(H);
1
```

142.5.9 Predicates

These commands determine whether a space of homomorphisms is a ring, if it's commutative, if it's a field (and what field), if it was created using the `HeckeAlgebra` command, whether it has been tensored with $\mathbf{Q}$, or whether it is saturated in the full ring of endomorphisms. Equality and inclusion can also be tested.

`IsRing(H)`

Return `true` if the space H of homomorphisms of abelian varieties is a ring. (Note that a ring does not have to contain unity.)

`IsField(H)`

Return `true` if the space H of homomorphisms of abelian varieties is a field, and if so returns that field, a map from the field to H, and a map from H to the field.

`IsCommutative(H)`

Return `true` if and only if the space H of homomorphisms of abelian varieties is a commutative ring.

`IsHeckeAlgebra(H)`

Return `true` if the space H of homomorphisms of abelian varieties was constructed using the `HeckeAlgebra` command.

`IsOverQ(H)`

Return `true` if the space H of homomorphisms of abelian varieties is a $\mathbf{Q}$-vector space instead of just a $\mathbf{Z}$-module, i.e., a space of homomorphisms up to isogeny.

`IsSaturated(H)`

Return `true` if the space H of homomorphisms of abelian varieties is equal to its saturation, i.e., the quotient of the ambient $\mathrm{Hom}(A, B)$ by H is torsion free.

`H1 eq H2`

Return `true` if the spaces H_1 and H_2 of homomorphisms of abelian varieties are equal.

`H1 subset H2`

Return `true` if the spaces H_1 and H_2 of homomorphisms of abelian varieties are both subgroups of a common $\mathrm{Hom}(A, B)$, and in addition H_1 is a subset of H_2.

Example H142E77

We illustrate several of the commands for the endomorphism ring of $J_0(33)$.

```
> H := End(JZero(33));
> IsCommutative(H);
false
> IsField(H);
false 0 0 0
> IsHeckeAlgebra(H);
false
> IsOverQ(H);
false
> IsOverQ(BaseExtend(H,RationalField()));
true
> IsRing(H);
true
> IsSaturated(H);
true
```

Next we compare the endomorphism ring with the Hecke algebra of $J_0(33)$.

```
> T := HeckeAlgebra(JZero(33));
> T eq H;
false
> T subset H;
true
> IsSaturated(T);
true
> IsRing(T);
true
> IsHeckeAlgebra(T);
```

```
true
> IsCommutative(T);
true
> IsField(BaseExtend(T,RationalField()));
false 0 0 0
```

The Hecke algebra of $J_0(33)$ is actually a product of 3 fields, so it is not a field. In contrast, the Hecke algebra of $J_0(23)$ is a field.

```
> T := HeckeAlgebra(JZero(23));
> IsField(T);
false 0 0 0
```

In the following code, the answer you get might be different, since computation of the defining polynomial for the number field involves a randomized algorithm.

```
> IsField(BaseExtend(T,RationalField()));
true Number Field with defining polynomial x^2 + 11*x + 29 over
the Rational Field
Mapping from: Number Field with defining polynomial x^2 + 11*x +
    29 over the Rational Field to HeckeAlg(JZero(23))_Q: Group of
homomorphisms from JZero(23) to JZero(23) in the category of abelian
varieties up to isogeny  given by a rule [no inverse]
Mapping from: HeckeAlg(JZero(23))_Q: Group of homomorphisms from
JZero(23) to JZero(23) in the category of abelian varieties up to
isogeny  to Number Field with defining polynomial x^2 + 11*x + 29
over the Rational Field given by a rule [no inverse]
```

142.5.10 Elements

Elements of spaces of homomorphisms exist as maps between abelian varieties as discussed in Section 142.4.

`H ! x`

Coerce x into the space H of homomorphisms of abelian varieties. For this coercion to be successful x must be a homomorphism between 2 abelian varieties, an integer or a rational number or a matrix coercible into the matrix space of H base extended to the rational numbers.

Example H142E78

```
> H := End(JZero(22));
> H ! Matrix(Integers(), 4, 4,
>              [9, -4, 0, 2, 0, 6, 0, 0, 2, -6, 11, -2, 3, -4, -0, 8]);
Homomorphism from JZero(22) to JZero(22) given on integral homology by:
[ 9 -4  0  2]
[ 0  6  0  0]
[ 2 -6 11 -2]
```

```
[ 3 -4  0  8]
```

142.6 Arithmetic of Abelian Varieties

142.6.1 Direct Sum

One can take arbitrary finite direct sums of modular abelian varieties. We do not write `A+B` for the direct sum, since it is already used for the sum of A and B inside a common ambient abelian variety, and this sum need not be direct, unless $A \cap B = 0$.

`DirectSum(A, B)`

`DirectProduct(A, B)`

`A * B`

The direct sum D of abelian varieties A and B, together with the embedding maps from A into D and B into D, respectively, and the projection maps from D onto A and B, respectively. It is not possible to take the direct sum of abelian varieties with different signs.

`DirectSum(X)`

`DirectProduct(X)`

The direct sum D of the sequence X of modular abelian varieties, together with a list containing the embedding maps from each modular abelian variety of X into D and a list containing the projection maps from D onto each modular abelian variety in X. It is not possible to take the direct sum of abelian varieties with different signs.

`A ^ n`

The direct sum of n copies of the abelian variety A. If $n = 0$, the zero subvariety of A. If n is negative, the $(-n)$-th power of the dual of A.

Example H142E79

Using the product operator we can take the direct product of any two modular abelian varieties, even ones of different weights or levels. We first illustrate taking the product of two abelian subvarieties of $J_0(65)$, then taking the product of one of the subvarieties of $J_0(65)$ with the weight 4 motive $J_1(11, 4)$.

```
> J := JZero(65);
> D := Decomposition(J); D;
[
    Modular abelian variety 65A of dimension 1, level 5*13 and
    conductor 5*13 over Q,
    Modular abelian variety 65B of dimension 2, level 5*13 and
    conductor 5^2*13^2 over Q,
```

```
    Modular abelian variety 65C of dimension 2, level 5*13 and
    conductor 5^2*13^2 over Q
]
> A := D[1];
> B := D[2];
> A*B;
Modular abelian variety 65A x 65B of dimension 3 and level 5*13
over Q
Homomorphism from 65A to 65A x 65B given on integral homology by:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
Homomorphism from 65B to 65A x 65B given on integral homology by:
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
Homomorphism from 65A x 65B to 65A (not printing 6x2 matrix)
Homomorphism from 65A x 65B to 65B (not printing 6x4 matrix)
> M := JZero(11,4);M;
Modular motive JZero(11,4) of dimension 2 and level 11 over Q
> P := A*M; P;
Modular motive 65A x JZero(11,4) of dimension 3 and level 5*11*13 over Q
```

The product also returns inclusions of each factor into the product and projection from the product onto each factor.

```
> C,f,g := A*B;
> f;
[*
Homomorphism from 65A to 65A x 65B given on integral homology by:
[1 0 0 0 0 0]
[0 1 0 0 0 0],
Homomorphism from 65B to 65A x 65B given on integral homology by:
[0 0 1 0 0 0]
[0 0 0 1 0 0]
[0 0 0 0 1 0]
[0 0 0 0 0 1]
*]
```

Here we compare direct sums of abelian varieties to the sum of of abelian varieties in a common ambient abelian variety. Thus if A is as above, then

```
> Dimension(A);
1
> Dimension(A*A);
2
> Dimension(A+A);
```

```
1
```

If you take a direct sum of abelian varieties that are defined over different base rings, then MAGMA will first attempt to express them over a common over-ring.

```
> A := JZero(11);
> B := BaseExtend(JZero(14),CyclotomicField(3));
> C := A*B; C;
Modular abelian variety JZero(11) x JZero(14) of dimension 2 and level
2*7*11 over Q(zeta_3)
```

The above would not work if `CyclotomicField(3)` were replaced by `GF(3)`, since the base ring $\mathbf{Q}$ of A is not contained in `GF(3)`.

142.6.2 Sum in an Ambient Variety

The sum `A+B` is the sum of A and B inside a common ambient abelian variety. This sum need not be direct, unless the intersection of A and B is 0.

`A + B`

The sum of the images of the abelian varieties A and B in a common ambient abelian variety.

`SumOf(X)`

The sum of the modular abelian varieties in the sequence X.

`SumOfImages(phi, psi)`

The sum D of the images of the morphisms ϕ and ψ of abelian varieties in their common codomain, a morphism from D into their common codomain, and a list containing a morphism from the domain of each of ϕ and ψ to D. If the codomains are not the same, then the homomorphisms are replaced by homomorphisms into an appropriate direct sum of codomains.

`SumOfMorphismImages(X)`

The sum D of the images of the morphisms of abelian varieties in the list X in their common codomain, a morphism from D into their common codomain, and a list containing a morphism from the domain of each morphism in X to D. If not all codomains of the elements of X are the same, then the homomorphisms are replaced by homomorphisms into an appropriate direct sum of codomains.

`FindCommonEmbeddings(X)`

Return `true` and a list of embeddings into a common abelian variety, if one can be found using `Embeddings(A)` for all abelian varieties A in the sequence X.

142.6.3 Intersections

Two abelian varieties cannot, by themselves, by intersected without choosing an embedding of both varieties in a common ambient abelian variety. The algorithm for computing an intersection is to compute the kernel of a certain homomorphism.

Intersections are computed in MAGMA by finding a homomorphism whose kernel is isomorphic to the intersection. For example, if $f : A \to C$ and $g : B \to C$ are injective homomorphisms, then the intersection of their images is isomorphic to the kernel of $f - g$.

As mentioned above, kernels of morphisms of abelian varieties are frequently not themselves abelian varieties. Instead a kernel is an extension of an abelian variety by a finite group of components. Likewise, intersections of abelian varieties are often not abelian varieties.

The intersection commands also take a sequence of abelian varieties or list of morphisms in order to facilitate computation of n-fold intersections, for any positive integer n.

`A meet B`

`Intersection(X)`

Given abelian varieties A and B or a sequence X of abelian varieties, compute a finite lift G of the component group of the intersection, the *connected* component of the intersection C, and a map from the abelian variety that contains C to the abelian variety that contains G. The relevant intersection is $C + G$. The elements of X are replaced by their images via their modular embedding map. All the elements of X must be embedded in the same abelian variety.

`IntersectionOfImages(X)`

Given a sequence X of morphisms from abelian varieties into a common abelian variety, compute a finite lift G of the component group of the intersection, the *connected* component C of the intersection, and a map from the abelian variety that contains C to the abelian variety that contains G. The morphisms in X do not have to be injective.

`ComponentGroupOfIntersection(A, B)`

`ComponentGroupOfIntersection(X)`

Given abelian varieties A and B or a sequence X of abelian varieties compute the group of components of the intersection of A and B or the varieties in X. (For more details, see the discussion of kernels in Section 142.4.3).

Example H142E80

The intersection of the three simple newform abelian subvarieties of $J_0(65)$ is a group isomorphic to $\mathbf{Z}/2\mathbf{Z}$.

```
> D := Decomposition(JZero(65));
> G := ComponentGroupOfIntersection(D); G;
Finitely generated subgroup of abelian variety with
invariants [ 2 ]
```

```
> FieldOfDefinition(G);
Rational Field
```

The quotient of D[1] by this subgroup of order 2 is an elliptic curve over $\mathbf{Q}$ isogenous to D[1], but not isomorphic to D[1].

```
> B  := D[1]/G; B;
Modular abelian variety of dimension 1 and level 5*13 over Q
> IsIsomorphic(D[1],B);
false
```

Next we compute some non-finite intersections.

```
> A := D[1] + D[2];
> B := D[1] + D[3];
> A meet B;
Finitely generated subgroup of abelian variety with invariants [ 2, 2, 2 ]
Modular abelian variety of dimension 1 and level 5*13 over Q
Homomorphism from modular abelian variety of dimension 1 to
modular abelian variety of dimension 6 given on integral homology
by:
[ 1 -1  0  0  0  0  1 -1  0  0  0 -1]
[ 0  0  1 -1  1 -1  0  0  1 -1  1  0]
Homomorphism from modular abelian variety of dimension 6 to
JZero(65) (not printing 12x10 matrix)
```

We can also intersect images of morphisms.

```
> f := ModularEmbedding(A);
> g := ModularEmbedding(B);
> _, C := IntersectionOfImages([* f, g *]);
> C eq D[1];
true
```

Example H142E81

The following example illustrates failure of multiplicity one for $J_0(p)$ for a prime number p. This is the first such example known, and it was discovered by Lloyd Kilford using MAGMA [Kil02]. There are two elliptic curve factors A and B inside $J_0(431)$. The eigenforms associated to A and B are congruent modulo 2, but the intersection of A and B is trivial.

```
> J := JZero(431);
> IsPrime(431);
true
> A := Decomposition(J)[1];
> B := Decomposition(J)[2];
> G, C := A meet B;
> G;
{ 0 }: finitely generated subgroup of abelian variety with
invariants []
> C;
```

```
Modular abelian variety ZERO of dimension 0 and level 431 over Q
> Newform(A) - Newform(B);
-2*q^3 + 4*q^5 + 2*q^6 - 4*q^7 + O(q^8)
```

142.6.4 Quotients

If B is an abelian subvariety of A (or some natural image of B lies in A), then the quotient A/B is an abelian variety. Also, the cokernel of a homomorphism of abelian varieties is an abelian variety.

`A / B`

The quotient of the abelian variety A by a natural image B' of the abelian variety B. Here B' is the image of B under the modular embedding composed with the modular parameterization to A.

`Cokernel(phi)`

The cokernel of the morphism ϕ of abelian varieties and a morphism from the codomain of ϕ to the cokernel.

Example H142E82

We compute a 2-dimensional quotient of the 3-dimensional abelian variety $J_0(33)$ using the Hecke operator T_2.

```
> J := JZero(33);
> T := HeckeOperator(J,2);
> Factorization(CharacteristicPolynomial(T));
[
    <x - 1, 2>,
    <x + 2, 4>
]
> C := ConnectedKernel(T-1);
> B,psi := J/C;
> B;
Modular abelian variety of dimension 2 and level 3*11 over Q
```

142.7 Decomposing and Factoring Abelian Varieties

By the Poincare reducibility theorem, every abelian variety is isogenous to a product of simple abelian subvarieties. If A is a modular abelian variety over $\mathbf{Q}$, then A is isogenous to a product of simple abelian varieties A_f attached to newforms. The `Decomposition` and `Factorization` commands compute such decompositions.

142.7.1 Decomposition

`Decomposition(A)`

Given an abelian variety A, return a sequence $[B_i]$ of simple modular abelian varieties, whose product is isogenous to A. Each B_i is equipped with an embedding into A such that the sum of the images of the B_i is equal to A. This embedding is the first element of the output of `Embeddings` on page 4873, given a B_i.

`A(n)`

The nth factor in `Decomposition(A)`, denoted $A(n)$ where A is an abelian variety.

Example H142E83

We decompose $A = J_0(37) \times J_0(22)$, then find the embedding into A of a factor which is isogenous to $J_0(11)$.

```
> A := JZero(37) * JZero(22);
> D := Decomposition(A); D;
[
    Modular abelian variety 37A of dimension 1, level 2*11*37 and
    conductor 37 over Q,
    Modular abelian variety 37B of dimension 1, level 2*11*37 and
    conductor 37 over Q,
    Modular abelian variety N(11,814,1)(11A) of dimension 1,
    level 2*11*37 and conductor 11 over Q,
    Modular abelian variety N(11,814,2)(11A) of dimension 1,
    level 2*11*37 and conductor 11 over Q
]
> B := D[3];
> Embeddings(B);
[*
Homomorphism from N(11,814,1)(11A) to JZero(37) x JZero(22) given on
integral homology by:
[ 0  0  0  0  1  0 -1  3]
[ 0  0  0  0  0  1 -2  3]
*]
```

142.7.2 Factorization

`Factorisation(A)`

`Factorization(A)`

Given an abelian variety A, compute pairwise non-isogenous simple newform abelian varieties A_f whose product, with multiplicities, is isomorphic to A. A list of pairs $\langle B, [\phi, \ldots] \rangle$ is returned, where B is an isogeny simple abelian variety and $[\phi, \ldots]$ is a sequence of maps from B into A (whose length is the "multiplicity"), such that the product of all images of all B is isogenous to A, and the sum of the dimensions of the images of B is the dimension of A. Moreover, the B are pairwise non-isogenous. To obtain a list of the images of the B canonically embedded into A, use `Decomposition(A)`.

Example H142E84

```
> A := JZero(37) * JZero(22);
> Factorization(A);
[*
<Modular abelian variety 37A of dimension 1, level 37 and
conductor 37 over Q, [
    Homomorphism N(37,814,1) from 37A to JZero(37) x JZero(22) given on
    integral homology by:
    [ 1 -1  1  0  0  0  0  0]
    [ 1 -1 -1  1  0  0  0  0]
]>,
<Modular abelian variety 37B of dimension 1, level 37 and
conductor 37 over Q, [
    Homomorphism N(37,814,1) from 37B to JZero(37) x JZero(22) given on
    integral homology by:
    [1 1 1 0 0 0 0 0]
    [0 0 0 1 0 0 0 0]
]>,
<Modular abelian variety 11A of dimension 1, level 11 and
conductor 11 over Q, [
    Homomorphism N(11,814,1) from 11A to JZero(37) x JZero(22) given on
    integral homology by:
    [ 0  0  0  0  0  1 -2  3]
    [ 0  0  0  0  1 -1  1  0],
    Homomorphism N(11,814,2) from 11A to JZero(37) x JZero(22) given on
    integral homology by:
    [ 0  0  0  0 -1  0  2 -2]
    [ 0  0  0  0 -1  2 -1  0]
]>
*]
```

142.7.3 Decomposition with respect to an Endomorphism or a Commutative Ring

The following commands use the elements of a commutative subring of endomorphisms to decompose a modular abelian variety A into a direct sum of abelian subvarieties by taking kernels (which are analogous to generalized eigenspaces).

`DecomposeUsing(R)`

Decompose an abelian variety A using the commutative ring of endomorphisms generated by the space of homomorphisms R of A.

`DecomposeUsing(phi)`

Decompose an abelian variety A using the endomorphism ϕ of A.

Example H142E85

```
> T2 := HeckeOperator(JZero(100),2);
> DecomposeUsing(T2);
[
    Modular abelian variety of dimension 1 and level 2^2*5^2 over Q,
    Modular abelian variety of dimension 5 and level 2^2*5^2 over Q,
    Modular abelian variety of dimension 1 and level 2^2*5^2 over Q
]
> W := AtkinLehnerOperator(JZero(100),100);
> DecomposeUsing(W);
[
    Modular abelian variety of dimension 3 and level 2^2*5^2 over Q,
    Modular abelian variety of dimension 4 and level 2^2*5^2 over Q
]
```

142.7.4 Additional Examples

Example H142E86

We compute a decomposition of $J_0(46)$ as a product of simple abelian subvarieties.

```
> J := JZero(46); J;
Modular abelian variety JZero(46) of dimension 5 and level 2*23 over Q
> Decomposition(J);
[
    Modular abelian variety 46A of dimension 1, level 2*23 and
    conductor 2*23 over Q,
    Modular abelian variety N(23,46,1)(23A) of dimension 2, level
    2*23 and conductor 23^2 over Q,
    Modular abelian variety N(23,46,2)(23A) of dimension 2, level
    2*23 and conductor 23^2 over Q
```

```
]
```

Thus J decomposes as a product $E \times A \times B$, where E is an elliptic curve of conductor 46, and A and B are two isogenous images of $J_0(23)$.

```
> J(1);
Modular abelian variety 46A of dimension 1, level 2*23 and
conductor 2*23 over Q
> Conductor(J(1));
46
> Factorization(Conductor(J(2)));
[ <23, 2> ]
```

The `Factorization` command gives an explicit decomposition with embeddings of each factor into $J_0(46)$.

```
> Factorization(Conductor(J(2)));
[ <23, 2> ]
> Factorization(J);
[*
<Modular abelian variety 46A of dimension 1, level 2*23 and
conductor 2*23 over Q, [
    Homomorphism from 46A to JZero(46) given on integral homology
    by:
    [ 1  0 -2 -1 -1  1  1  1 -2  1]
    [ 0  1 -1 -1  0  0  0  1 -1  0]
]>,
<Modular abelian variety 23A of dimension 2, level 23 and
conductor 23^2 over Q, [
    Homomorphism N(23,46,1) from 23A to JZero(46) given on integral
    homology by:
    [-1  1 -1  1  0 -1 -1  1 -1  2]
    [ 0  0 -1  2 -2 -1  0  0  1  0]
    [ 0  0  0  1 -2  0  0  1  0  0]
    [ 0  1  0 -1  0  0  1  0  0  0],
    Homomorphism N(23,46,2) from 23A to JZero(46) given on integral
    homology by:
    [ 0 -1  0  0  1 -1  0  1  0  0]
    [-1  0  0  0  0 -1  2 -1  1 -1]
    [-1  1 -1  0  0  0  2 -2  2 -2]
    [ 0  0 -1  2 -1  0  0  0  0 -1]
]>
*]
```

142.8 Building Blocks

The abelian variety $A_f/\mathbf{Q}$ associated to a newform f is isogenous over $\overline{\mathbf{Q}}$ to a power ${B_f}^r$ of some simple variety B_f, which is called a "building block". This section is concerned with computations related to the field of definition of B_f, and its endomorphism algebra.

The functions in this section take spaces of modular symbols, and *must have sign* +1. For instance, the modular symbols of level N, weight 2 and sign +1 are obtained from `ModularSymbols(N,2,1)`. (These spaces have the same dimension as the corresponding spaces of modular forms.) Furthermore, the space of symbols is expected to be *cuspidal, new and irreducible over* $\mathbf{Q}$ (in other words, the corresponding space of cusp forms is spanned by a single Galois orbit of newforms).

Acknowledgement: This section was contributed by Jordi Quer. For more details of the theory, and some tables, see [Que09].

142.8.1 Background and Notation

Let $f = \sum_{n=1}^{\infty} a_n q^n \in S_2^{new}(N, \varepsilon)$ be a newform of weight 2, level N and Nebentypus ε. Let E be the number field generated by the coefficients a_n, and let F be the field generated by the numbers $\mu_p := a_p^2/\varepsilon(p)$ for primes p not dividing N.

The abelian variety $A_f/\mathbf{Q}$ attached to f (that is, associated to the space spanned by f and its conjugates) is a variety of dimension equal to the degree $[E : \mathbf{Q}]$ with endomorphism algebra $\mathrm{End}_{\mathbf{Q}}(A_f) \otimes \mathbf{Q}$ isomorphic to E.

The variety A_f has complex multiplication if there is a nontrivial character χ (necessarily of order 2) such that $a_p = \chi(p)a_p$ for all p not dividing N, and in this case the variety B_f is an elliptic curve with complex multiplication by the quadratic field fixed by the kernel of χ.

We assume for the rest of this section that A_f has no complex multiplication. Then, the field F is totally real, and E/F is an abelian extension. E is totally real when ε is trivial, and otherwise E is a CM field. The endomorphism algebra $\mathrm{End}(B_f) \otimes \mathbf{Q}$ is a division algebra with centre F. It is either equal to F, and then $\dim(B_f) = [F : \mathbf{Q}]$ and $r = [E : F]$, or otherwise is a quaternion algebra over F, and then $\dim(B_f) = 2[F : \mathbf{Q}]$ and $r = [E : F]/2$.

For every element $s \in G_F$ there exists a unique primitive Dirichlet character χ_s, which only depends on the action of s on the field E, such that $a_p^s = \chi_s(p)a_p$ for all p not dividing N. The characters χ_s are called the *inner twists* of the form f.

Let F_δ be the extension of F generated by the square roots of the elements $\mu_p \in F$ (for p not dividing N). Note that $EF_\delta = E(\sqrt{\varepsilon})$, the field obtained by adjoining square roots of the character values, which is either E or a quadratic extension of E. We may associate to each $s \in \mathrm{Gal}(F_\delta/F)$ a primitive quadratic Dirichlet character ψ_s, defined by $\sqrt{\mu_p}^s = \psi_s(p)\sqrt{\mu_p}$ (for p not dividing N). These are called the *quadratic degree characters* of the form f. They are related with the inner twists of f by the identity $\chi_s(p) = \psi_s(p)\sqrt{\varepsilon(p)}^s/\sqrt{\varepsilon(p)}$.

Let $K_P/\mathbf{Q}$ be the field fixed by the kernel of the homomorphism $\delta : G_{\mathbf{Q}} \to F^*/F^{*2}$ that sends $Frob_p$ to μ_p for all p not dividing N with a_p nonzero. (This is enough to determine the homomorphism, because the primes with $a_p = 0$ have density zero). This homomorphism is returned by `DegreeMap` (the syntax uses the notation below).

The Galois group $Gal(K_P/\mathbf{Q})$ is abelian of exponent 2; choose a basis $\sigma_1, ..., \sigma_r$. Let $\psi_1, ..., \psi_r$ be the basis of $\mathrm{Hom}(Gal(K_P/\mathbf{Q}), \mathbf{Z}/2)$ dual to the basis $\{\sigma_i\}$, and consider the ψ_i's as characters on $G_{\mathbf{Q}}$. Also let t_i be a quadratic discriminant such that $\mathbf{Q}(\sqrt{t_i}) \subset K_P$ is the field fixed by the kernel of ψ_i.

Denote by γ_ε the element of $Br(\mathbf{Q})[2]$ given by the two-cocycle sending σ, τ to $\sqrt{\varepsilon(\sigma)}\sqrt{\varepsilon(\tau)}\sqrt{\varepsilon(\sigma\tau)}^{-1}$. Then the class in $Br(F)[2]$ of the endomorphism algebra $\mathrm{End}^0(B_f)$ is the restriction to F of $\gamma_\varepsilon \prod(t_i, \delta(\sigma_i))$. This is computed by `BrauerClass`.

In general the smallest fields of definition up to isogeny for B_f are quadratic extensions of K_P. It is sometimes possible to "descend" B_f (up to isogeny) to the field K_P. The obstruction to this descent lies in $Br(K_P)[2]$, and is given by the restriction of γ_ε to K_P. This is computed by `ObstructionDescentBuildingBlock`.

`BoundedFSubspace(epsilon, k, degrees)`

A sequence containing the irreducible subspaces of the modular symbols of weight k and Nebentypus character ϵ corresponding to non-CM newforms for which the degree $[F : \mathbf{Q}]$ of the centre of the endomorphism algebra (of the associated abelian variety) is in the sequence *degrees*.

`HasCM(M : parameters)`

`IsCM(M : parameters)`

`Proof` BoolElt *Default :* `false`

Return `true` if and only if the modular abelian variety attached to the given space of modular symbols has complex multiplication. When the level is larger than 100, an unproved bound is used in the computation, unless `Proof` is set to `true`.

`InnerTwists(A : parameters)`

`InnerTwists(M : parameters)`

`Proof` BoolElt *Default :* `false`

The inner twists of the newform $f = \sum a_n q^n$ corresponding to the given space of modular symbols, or to the given modular abelian variety. This should be irreducible over $\mathbf{Q}$, and f should not have CM, and only spaces of modular symbols with sign $+1$ are accepted.

The inner twists are Dirichlet characters satisfying $\chi_s(p) = a_p^s/a_p$ (for primes p not dividing the level), where s is in the absolute Galois group of $\mathbf{Q}$.

Warning: when the level is larger than 100, a non-rigorous bound is used in the computation, unless `Proof` is set to `true`. Even in that case, the returned twists are only checked to be inner twists up to precision 10^{-5}.

`DegreeMap(M : parameters)`

`Proof` BoolElt *Default :* `false`

The homomorphism $\delta : G_{\mathbf{Q}} \to F^*/F^{*2}$ (as defined in the introduction), attached to the given space M of modular symbols (which should be new, irreducible over $\mathbf{Q}$, and have sign +1). The second object returned is F. The first object returned is a sequence of tuples $< t_i, f_i >$, where $t_i \in \mathbf{Q}$ are quadratic discriminants and $f_i \in F$. The t_i determine a basis σ_i of $\mathrm{Gal}(K_P/\mathbf{Q})$ as in the introduction, and δ is the map sending σ_i to f_i.

`BrauerClass(M)`

Given a space of modular symbols M corresponding to a newform f, this function computes the Brauer class of the endomorphism algebra of the associated abelian variety A_f (or motive M_f). The endomorphism algebra is either a number field F (in which case an empty sequence is returned), or a quaternion algebra over a number field F. The corresponding class in the Brauer group $Br(F)[2]$ is specified by returning the sequence of places of F that ramify in the quaternion algebra.

The given space M should be irreducible over $\mathbf{Q}$, and have sign +1, and f should not have CM.

`ObstructionDescentBuildingBlock(M)`

Given a space of modular symbols M corresponding to a newform f, this function computes the obstruction to "descending" the building block B_f to the field K_P (for definitions, see the introduction). The obstruction to the existence of a building block over K_P isogenous to B_f is an element of the Brauer group $Br(K_P)[2]$. (More precisely, for any field L there exists such a building block over L if and only if L is an extension of K_P over which this Brauer element splits.) The Brauer class is specified by returning the sequence of places of K_P where the class is not locally trivial.

The given space M should be irreducible over $\mathbf{Q}$, and have sign +1, and f should not have CM.

Example H142E87

The lowest level where a nontrivial obstruction occurs is 28, with a character of order 6. We find that the space of modular symbols (with sign 1) for this character has dimension 2 over the field of character values.

```
> Chi28 := FullDirichletGroup(28);
> chi := Chi28.1*Chi28.2;
> Chi28;
Group of Dirichlet characters of modulus 28
   over Cyclotomic Field of order 6 and degree 2
> Order(Chi28), Order(chi);
12 6
>  M28chi := CuspidalSubspace( ModularSymbols(chi, 2, 1));
> M28chi;
```

```
Modular symbols space of level 28, weight 2, character $.1*$.2, and dimension
2 over Cyclotomic Field of order 6 and degree 2
> qEigenform(M28chi);
q + (1/3*(-zeta_6 - 1)*a - zeta_6)*q^2 + a*q^3 + 1/3*(4*zeta_6 - 2)*a*q^4 +
    (zeta_6 - 2)*q^5 + (-zeta_6*a + (2*zeta_6 - 1))*q^6 + 1/3*(-zeta_6 - 4)*a*q^7
    + O(q^8)
> Parent(Coefficients(qEigenform(M28chi))[2]);
Univariate Quotient Polynomial Algebra in a
   over Cyclotomic Field of order 6 and degree 2
   with modulus a^2 + 3*zeta_6
```

So the q-eigenform is defined over a quadratic extension of $\mathbf{Q}(\zeta_6)$, In fact, this extension is $\mathbf{Q}(\zeta_6, i)$. Since the space `M28chi` has dimension 2 over $\mathbf{Q}(\zeta_6)$, it is irreducible over $\mathbf{Q}(\zeta_6)$. The corresponding abelian variety over $\mathbf{Q}$ therefore has dimension 4.

```
> A := ModularAbelianVariety(M28chi);
> A;
Modular abelian variety of dimension 4 and level 2^2*7 over Q with sign 1
> delta, F := DegreeMap(M28chi);
> F;
Rational Field
```

This means that A is isogenous to B^2 for some abelian variety B over $\overline{\mathbf{Q}}$ of dimension 2, and that the endomorphism algebra of B is a quaternion algebra over $F = \mathbf{Q}$.

```
> BrauerClass(M28chi);
[ 2, 3 ]
```

This means the endomorphism algebra of B is the quaternion algebra over $\mathbf{Q}$ ramified only at 2 and 3. Now we determine the possible fields of definition of B.

```
> delta;  // This came from DegreeMap, above.
[ <-7, 3> ]
```

In particular, this tells us that $K_P = \mathbf{Q}(\sqrt{-7})$.

```
> ObstructionDescentBuildingBlock(M28chi);
[ Place at Prime Ideal
Two element generators:
    [2, 0]
    [0, 1],
Place at Prime Ideal
Two element generators:
    [2, 0]
    [3, 1] ]
> Universe($1);  // What are these places elements of?
Set of Places of Number Field with defining polynomial x^2 + 7
over the Rational Field
```

The obstruction is given as a list of places of $\mathbf{Q}(\sqrt{-7})$. Recall that B can be defined over any extension of K_P for which the obstruction is trivial. In this case, any extension of $\mathbf{Q}(\sqrt{-7})$ which splits the quaternion algebra over $\mathbf{Q}(\sqrt{-7})$ ramified at the two primes above 2.

142.9 Orthogonal Complements

142.9.1 Complements

The following two commands find a complement of an abelian subvariety of an abelian variety. Existence of a complement is guaranteed by the Poincare reducibility theorem (if we were just working with n-dimensional complex tori, then there need not be complements of subtori). MAGMA computes a complement using the module-theoretic structure of the ambient variety.

`Complement(A : parameters)`

`IntPairing` BOOLELT *Default :* `false`

The complement of the image of the abelian variety A under the first embedding of A (the first map in the sequence returned by `Embeddings` on page 4873). If the parameter `IntPairing` is set to `true`, the intersection pairing is used to compute the homology complement.

`ComplementOfImage(phi : parameters)`

`IntPairing` BOOLELT *Default :* `false`

Suppose $\phi : A \to B$ is a morphism of abelian varieties. By the Poincare reducibility theorem, there is an abelian variety C such that $\phi(A) + C = B$ and the intersection of $\phi(A)$ with C is finite. This intrinsic returns a choice of C and an embedding of C into B. If the parameter `IntPairing` is set to `true`, the intersection pairing is used to compute the homology complement.

Example H142E88

We compute a decomposition of $J_0(33)$ as a product of simples, then find a decomposition of the complement of one of the factors.

```
> J := JZero(33);
> D := Decomposition(J); D;
[
    Modular abelian variety 33A of dimension 1, level 3*11 and
    conductor 3*11 over Q,
    Modular abelian variety N(11,33,1)(11A) of dimension 1, level
    3*11 and conductor 11 over Q,
    Modular abelian variety N(11,33,3)(11A) of dimension 1, level
    3*11 and conductor 11 over Q
]
> A := D[3];
> B := Complement(A);
> B;
Modular abelian variety of dimension 2 and level 3*11 over Q
> Decomposition(B);
[
    Modular abelian variety image(33A) of dimension 1, level 3*11
```

```
    and conductor 3*11 over Q,
    Modular abelian variety image(11A) of dimension 1, level 3*11
    and conductor 11 over Q
]
```

Here we compute a somewhat random map from $J_0(11)$ to $J_0(33)$, and compute the complement of the image.

```
> phi := 2*NaturalMap(JZero(11),JZero(33),1) - 3*NaturalMap(JZero(11),
>                         JZero(33),3);
> phi;
Homomorphism from JZero(11) to JZero(33) given on integral homology by:
[ 2  6 -7 -2 -6  3]
[ 5 -2 -3 -1 -1  5]
> C,pi := ComplementOfImage(phi);
> C;
Modular abelian variety of dimension 2 and level 3*11 over Q
> Decomposition(C);
[
    Modular abelian variety image(33A) of dimension 1, level 3*11
    and conductor 3*11 over Q,
    Modular abelian variety image(11A) of dimension 1, level 3*11
    and conductor 11 over Q
]
```

142.9.2 Dual Abelian Variety

It is possible to compute an abelian variety which is dual to a given variety in many of the cases that are of interest. Let A be an abelian variety and suppose the modular map $A \to J$ is injective, where J is attached to a space of modular symbols and J is isomorphic to its dual (e.g., $J = J_0(N)$). To compute the dual of A, we find a complement B of A in J whose homology is orthogonal to the homology of A with respect to the intersection pairing. This can frequently be accomplished (e.g., when A is attached to a newform) without using the intersection pairing by finding a complement B such that the rational homology of B as a module over the Hecke algebra has no simple factors in common with that of A. Then J/B is isomorphic to the dual of A.

`IsDualComputable(A)`

Return `true` if the dual of the abelian variety A can be computed, and the dual. Otherwise, return `false` and a message.

`Dual(A)`

The dual abelian variety of the abelian variety A. The modular map to a modular symbols abelian variety must be injective.

`ModularPolarization(A)`

The polarization on the abelian variety A induced by pullback of the theta divisor.

Example H142E89

We compute the dual of a 2-dimensional newform abelian variety of level 43, and note that it is isomorphic to itself.

```
> J := JZero(43);
> A := Decomposition(J)[2]; A;
Modular abelian variety 43B of dimension 2, level 43 and
conductor 43^2 over Q
> Adual := Dual(A); Adual;
Modular abelian variety of dimension 2 and level 43 over Q
> IsIsomorphic(A,Adual);
true Homomorphism from 43B to modular abelian variety of
dimension 2 given on integral homology by:
[-1  1 -1  1]
[-1  0  1  0]
[-1  0  0  0]
[ 0  0 -1  1]
```

Next we compute the dual of a 2-dimensional newform abelian variety of level 69, and find that it is not isomorphic to itself.

```
> A := Decomposition(JZero(69))[2]; A;
Modular abelian variety 69B of dimension 2, level 3*23 and
conductor 3^2*23^2 over Q
> Adual := Dual(A); Adual;
Modular abelian variety of dimension 2 and level 3*23 over Q
> IsIsomorphic(A,Adual);
false
```

One can show that the natural map from A to its dual is a polarization of degree 484.

```
> phi := NaturalMap(A,Adual);
> phi;
Homomorphism N(1) from 69B to modular abelian variety of
dimension 2 given on integral homology by:
[  3   1   5  -7]
[ -1   5  -1   1]
[ -6   4  -1   7]
[ 11  -7   6 -12]
> Degree(phi);
484
> Factorization(484);
[ <2, 2>, <11, 2> ]
1
```

142.9.3 Intersection Pairing

These commands compute the matrix of the intersection pairing on homology with respect to the fixed basis for rational or integral homology. If A is not a modular symbols abelian variety (such as $J_0(N)$), then the intersection pairing on homology computed below *is the one obtained by pulling back from the pairing on the codomain of the modular embedding* of A. This may not be what you expect, but is easy to compute in great generality.

Computation of intersection pairings is currently only implemented for weight 2.

`IntersectionPairing(H)`

The intersection pairing matrix on the basis of the homology H of an abelian variety.

`IntersectionPairing(A)`

The intersection pairing matrix on the basis for the rational homology of the abelian variety A, pulled back using the modular embedding.

`IntersectionPairingIntegral(A)`

The intersection pairing matrix on the basis for the integral homology of the abelian variety A, pulled back using the modular embedding.

Example H142E90

The intersection pairing on $J_0(11)$ is very simple.

```
> J := JZero(11);
> IntersectionPairing(J);
[ 0 -1]
[ 1  0]
> IntersectionPairingIntegral(J);
[ 0 -1]
[ 1  0]
```

The intersection pairing associated to $J_0(33)$ is more interesting. Note that the representing matrix is skew symmetric and has determinant 1.

```
> I := IntersectionPairingIntegral(JZero(33)); I;
[ 0  1  0  1  0  1]
[-1  0  0  1  0  1]
[ 0  0  0  1  0  1]
[-1 -1 -1  0  0  1]
[ 0  0  0  0  0  1]
[-1 -1 -1 -1 -1  0]
> Determinant(I);
1
```

The intersection pairing on **33A** is surprising, because it is pulled back from the intersection pairing on $J_0(33)$. Thus instead of having determinant 1, it has determinant 9.

```
> A := ModularAbelianVariety("33A");  A;
Modular abelian variety 33A of dimension 1 and level 3*11 over Q
```

```
> I := IntersectionPairingIntegral(A); I;
[ 0  3]
[-3  0]
> Determinant(I);
9
```

142.9.4 Projections

Suppose ϕ is a homomorphism from A to B. Then the image $\phi(A)$ is an abelian subvariety of B. The commands below compute a map π in the endomorphism algebra of B whose image is $\phi(A)$ and such that $\pi^2 = \pi$, i.e., π is projection onto $\phi(A)$. A projection map is not canonical, unless it is required to respect the intersection pairing.

`ProjectionOnto(A` : *parameters*`)`

`ProjectionOntoImage(phi` : *parameters*`)`

`IntPairing` BoolElt *Default :* `false`

Given a morphism $\phi : A \to B$ return a projection onto $\phi(A)$ as an element of $E \otimes \mathbf{Q}$, where E is the endomorphism ring of B. If the optional parameter `IntPairing` is set to `true`, then the projection is also required to respect the intersection pairing, which uniquely determines π.

Example H142E91

```
> pi := ProjectionOnto(ModularAbelianVariety("33A"));  pi;
Homomorphism pi from JZero(33) to JZero(33) (up to isogeny) on
integral homology by:
 (not printing 6x6 matrix)
> Matrix(pi);
[ 2/3  1/3 -1/3  1/3    0 -2/3]
[ 1/3  2/3 -2/3  2/3    0 -1/3]
[ 1/3  1/3 -1/3  1/3    0 -1/3]
[   0  2/3 -2/3  2/3    0    0]
[   0  1/3 -1/3  1/3    0    0]
[-1/3  1/3 -1/3  1/3    0  1/3]
> pi^2 eq pi;
true
> Rank(pi);
1
```

Example H142E92

```
> phi := NaturalMap(JZero(11),JZero(44));
> pi := ProjectionOntoImage(phi); pi;
Homomorphism pi from JZero(44) to JZero(44) (up to isogeny) on
integral homology by:
 (not printing 8x8 matrix)
> A := Image(5*pi); A;
Modular abelian variety of dimension 1 and level 2^2*11 over Q
> IsIsomorphic(JZero(11),A);
true Homomorphism from JZero(11) to modular abelian variety of
dimension 1 given on integral homology by:
[ 0 -1]
[-1  1]
```

142.9.5 Left and Right Inverses

Left and right inverses of homomorphisms in the category of abelian varieties up to isogeny can be computed. A left or right inverse times a minimal integer can be computed instead so that the result is a homomorphism.

MAGMA computes a right inverse of a finite-degree homomorphism ϕ by finding the projection map onto the complement of the image of ϕ, and composing with a inverse from the image. To find a left inverse of a surjective homomorphism $\phi : A \to B$, MAGMA computes the complement C of the kernel of ϕ and inverts ϕ restricted to C. This complement C is an abelian subvariety of A that maps isomorphically onto B.

`LeftInverse(phi : parameters)`

`IntPairing` BOOLELT *Default :* `false`

Given a surjective homomorphism $\phi : A \to B$ of abelian varieties return a homomorphism $\psi : B \to A$ of minimal degree in the category of abelian varieties up to isogeny such that $\psi * \phi$ is the identity map on B and an integer d such that $d * \psi$ is a homomorphism.

If the parameter `IntPairing` is `true` then the intersection pairing is used to compute the homology complement.

`LeftInverseMorphism(phi : parameters)`

`IntPairing` BOOLELT *Default :* `false`

Given a surjective homomorphism $\phi : A \to B$ of abelian varieties, return a homomorphism $\psi : B \to A$ of minimal degree such that $\psi * \phi$ is multiplication by an integer.

If the parameter `IntPairing` is `true` then the intersection pairing is used to compute the homology complement.

`RightInverse(phi : parameters)`

`IntPairing` BoolElt *Default :* `false`

Given a homomorphism $\phi : A \to B$ of abelian varieties with finite kernel, return a map $\psi : B \to A$ in the category of abelian varieties up to isogeny such that $\phi * \psi : A \to A$ is the identity map.

If the parameter `IntPairing` is `true` then the intersection pairing is used to compute the homology complement.

`RightInverseMorphism(phi : parameters)`

`IntPairing` BoolElt *Default :* `false`

Given a homomorphism $\phi : A \to B$ of abelian varieties with finite kernel return a minimal-degree homomorphism $\psi : B \to A$ such that $\phi * \psi : A \to A$ is multiplication by an integer.

If the parameter `IntPairing` is `true` then the intersection pairing is used to compute the homology complement.

Example H142E93

First we compute the difference ϕ of the two natural degeneracy maps $J_0(11) \to J_0(33)$, which has as kernel a group of order 5 (called the Shimura subgroup in this case).

```
> J11 := JZero(11);  J33 := JZero(33);
> d1 := NaturalMap(J11,J33,1);
> d3 := NaturalMap(J11,J33,3);
> phi := d1-d3;
> Degree(phi);
5
```

A right inverse of ϕ is a homomorphism up to isogeny from $J_0(33)$ to $J_0(11)$.

```
> RightInverse(phi);
Homomorphism from JZero(33) to JZero(11) (up to isogeny) on integral
homology by:
 (not printing 6x2 matrix)
15
> RightInverseMorphism(phi);
Homomorphism from JZero(33) to JZero(11) (not printing 6x2 matrix)
> phi*RightInverseMorphism(phi);
Homomorphism from JZero(11) to JZero(11) given on integral homology by:
[15  0]
[ 0 15]
```

Finally we find a left inverse of a map from $J_0(33)$ to $J_0(11)$.

```
> psi := NaturalMap(J33,J11,1) - NaturalMap(J33,J11,3);
> IsSurjective(psi);
true
> LeftInverse(psi);
```

```
Homomorphism from JZero(11) to JZero(33) (up to isogeny) on integral
homology by:
[ 1/5    0 -2/5  2/5 -3/5    0]
[ 1/5 -1/5    0  1/5 -2/5  1/5]
5
> LeftInverseMorphism(psi);
Homomorphism from JZero(11) to JZero(33) given on integral homology by:
[ 1  0 -2  2 -3  0]
[ 1 -1  0  1 -2  1]
> LeftInverseMorphism(psi)*psi;
Homomorphism from JZero(11) to JZero(11) given on integral homology by:
[5 0]
[0 5]
```

142.9.6 Congruence Computations

The congruence modulus and the modular degree are each an integer which measures "congruences" between an abelian variety and other abelian varieties. These two quantities are related because if a prime divides the modular degree, then it divides the congruence modulus, though the converse need not be true (see the example below).

`CongruenceModulus(A)`

Given $A = A_f$, an abelian variety attached to a newform f, return the congruence modulus of the newform f, taken in the space $S_2(N, \epsilon)$, where ϵ is the character of the newform, which is an integer that measures congruences between f and nonconjugate forms in the Peterson complement of f. More precisely, if $f \in S_k(N, \varepsilon)$, which is the direct sum of the spaces of modulus forms with character a Galois conjugate of ε, then we define the congruence modulus of f to be the order of the group $S_k(N, \varepsilon; \mathbf{Z})/(W + W^{\perp})$, where W is the intersection of $S_k(N, \varepsilon; \mathbf{Z})$ with the span of the Galois conjugates of f.

`ModularDegree(A)`

The modular degree of the abelian variety A. This is the square root of the degree of the map from A to its dual A' induced by virtue of A being modular. In some cases where no algorithm is implemented for computing A', a message is printed and the square of the degree of the composition of the modular embedding with the modular parameterization is computed. When any weight of a factor of A is bigger than 2 the square root is not taken.

Example H142E94

The modular degree and congruence modulus of one of the two elliptic curves of conductor 54 are interesting because they are not equal. This is the smallest level of an elliptic curve where these two invariants differ. (For more details, see [AS04].)

```
> J := JZero(54);
> A,B := Explode(Decomposition(NewSubvariety(J)));
> ModularDegree(A);
6
> CongruenceModulus(A);
6
> ModularDegree(B);
2
> CongruenceModulus(B);
6
```

The modular degree and congruence modulus are 4 for a certain abelian surface A of level 65. We also compute the kernel of the modular map and see that it is $A[2]$.

```
> J := JZero(65);
> A := J(2); A;
Modular abelian variety 65B of dimension 2, level 5*13 and
conductor 5^2*13^2 over Q
> CongruenceModulus(A);
4
> ModularDegree(A);
4
> phi := NaturalMap(A,Dual(A));
> Invariants(Kernel(phi));
[ 2, 2, 2, 2 ]
```

142.10 New and Old Subvarieties and Natural Maps

142.10.1 Natural Maps

Suppose M and N are positive integers and M divides N. There are natural maps in both directions between $J_0(N)$ and $J_0(M)$ (and likewise for J_1, etc.), for each divisor of $t = N/M$, which correspond to maps of the form $f(q) \mapsto f(q^t)$ and their duals. Since any modular abelian variety A in MAGMA is equipped with a map $A \to J_e$ and $J_p \to A$, where J_e and J_p are attached to modular symbols, the problem of defining natural maps between A and B is reduced to defining natural maps between modular abelian varieties attached to modular symbols.

`NaturalMap(A, B, d)`

Given abelian varieties A and B and an integer d return the natural map from A to B induced, in a potentially complicated way, from the map $f(q) \mapsto f(q^d)$ on modular forms. In situations where the modular forms associated to A and B have nothing to do with each other, then we define this map to be the zero map.

`NaturalMap(A, B)`

The natural map from the abelian variety A to the abelian variety B induced by the identity on modular forms, or the zero map if there is none.

`NaturalMaps(A, B)`

Given abelian varieties A and B return a sequence of the natural maps from A to B for all divisors d of the level of A over the level B, or the level of B over the level A.

Example H142E95

```
> A := JZero(11)*JZero(22);
> B := JZero(11)*JZero(33);
> phi := NaturalMap(A,B);
> phi;
Homomorphism N(1) from JZero(11) x JZero(22) to JZero(11) x JZero(33) (not
printing 6x8 matrix)
> Nullity(phi);
1
> f := NaturalMap(A,B,3); f;
Homomorphism N(3) from JZero(11) x JZero(22) to JZero(11) x JZero(33) (not
printing 6x8 matrix)
> Nullity(f);
2
> NaturalMaps(JZero(11),JZero(33));
[
    Homomorphism N(1) from JZero(11) to JZero(33) given on integral
    homology by:
    [ 1  0 -2  2 -3  0]
    [ 1 -1  0  1 -2  1],
    Homomorphism N(3) from JZero(11) to JZero(33) given on integral
    homology by:
    [ 0 -2  1  2  0 -1]
    [-1  0  1  1 -1 -1]
]
```

If we take a product of several copies of $J_0(11)$ and of several copies of $J_0(22)$, the `NaturalMaps` command still only returns 2 natural maps, one for each divisor of the quotient of the levels.

```
> A := JZero(11)^2;
> B := JZero(22)^3;
```

```
> NaturalMaps(A,B);
[
    Homomorphism N(1) from JZero(11) x JZero(11) to JZero(22) x JZero(22) x
    JZero(22) given on integral homology by:
    [ 0  1 -2  3  0  1 -2  3  0  1 -2  3]
    [ 1 -1  1  0  1 -1  1  0  1 -1  1  0]
    [ 0  1 -2  3  0  1 -2  3  0  1 -2  3]
    [ 1 -1  1  0  1 -1  1  0  1 -1  1  0],
    Homomorphism N(2) from JZero(11) x JZero(11) to JZero(22) x JZero(22) x
    JZero(22) given on integral homology by:
    [-1  0  2 -2 -1  0  2 -2 -1  0  2 -2]
    [-1  2 -1  0 -1  2 -1  0 -1  2 -1  0]
    [-1  0  2 -2 -1  0  2 -2 -1  0  2 -2]
    [-1  2 -1  0 -1  2 -1  0 -1  2 -1  0]
]
```

142.10.2 New Subvarieties and Quotients

These commands compute the new and r-new subvarieties and quotients of an abelian variety A of level N. The *r-new subvariety* of A is the intersection of the kernels of all natural maps from A to modular abelian varieties of level N/r. The *new subvariety* is the intersection of the r-new subvarieties over all prime divisors r of N. The *r-new quotient* of A is the quotient of A by the sum of all images in A under all natural maps of abelian varieties of level N/r.

`NewSubvariety(A, r)`

The r-new subvariety of the abelian variety A.

`NewSubvariety(A)`

The new subvariety of the abelian variety A.

`NewQuotient(A, r)`

The r-new quotient of the abelian variety A.

`NewQuotient(A)`

The new quotient of the abelian variety A.

Example H142E96

```
> J := JZero(33);
> Dimension(J);
3
> Dimension(NewSubvariety(J,3));
1
> Dimension(NewSubvariety(J));
1
> Dimension(NewSubvariety(J,11));
3
> Dimension(NewQuotient(J));
1
> Dimension(OldSubvariety(J));
2
> Dimension(OldSubvariety(J,3));
2
```

142.10.3 Old Subvarieties and Quotients

These commands compute the old and r-old subvarieties and quotients of an abelian variety A of level N. The *r-old subvariety* of A is the sum of the images of all natural maps from modular abelian varieties of level N/r to A. The *old subvariety* is the sum of the r-old subvarieties as r varies over the divisors of N. The r-old quotient of A is the quotient of A by its r-new subvariety.

`OldSubvariety(A, r)`

The r-old subvariety of the abelian variety A.

`OldSubvariety(A)`

The old subvariety of the abelian variety A.

`OldQuotient(A, r)`

The r-old quotient of the abelian variety A.

`OldQuotient(A)`

The old quotient of the abelian variety A.

Example H142E97

We compute the old subvariety and old quotient of $J_0(100)$, both of which have dimension 6.

```
> J := JZero(100); J;
Modular abelian variety JZero(100) of dimension 7 and level 2^2*5^2
over Q
> J_old := OldSubvariety(J); J_old;
Modular abelian variety JZero(100)_old of dimension 6 and level
2^2*5^2 over Q
> phi := Embeddings(J_old)[1];
> Codomain(phi);
Modular abelian variety JZero(100) of dimension 7 and level 2^2*5^2
over Q
> Jold := OldQuotient(J); Jold;
Modular abelian variety JZero(100)^old of dimension 6 and level
2^2*5^2 over Q
```

The new subvariety and new quotient of $J_0(100)$ intersect in a finite subgroup isomorphic to $\mathbf{Z}/12\mathbf{Z} \times \mathbf{Z}/12\mathbf{Z}$.

```
> J_new := NewSubvariety(J); J_new;
Modular abelian variety JZero(100)_new of dimension 1 and level
2^2*5^2 over Q
> G, A := J_new meet J_old; G;
Finitely generated subgroup of abelian variety with invariants
[ 12, 12 ]
> Dimension(A);
0
```

142.11 Elements of Modular Abelian Varieties

We represent torsion points on modular abelian varieties as follows. Suppose A is an abelian variety defined over the complex numbers $\mathbf{C}$. Then $A(\mathbf{C})$ is canonically isomorphic to $H_1(A, \mathbf{R})/H_1(A, \mathbf{Z})$, and the torsion subgroup of $A(\mathbf{C})$ is isomorphic to $H_1(A, \mathbf{Q})/H_1(A, \mathbf{Z})$. We represent a torsion element of $A(\mathbf{C})$ by giving a representative element of $H_1(A, \mathbf{Q})$. The functions below provide basic arithmetic operations with such elements, application of homomorphisms, and conversion functions.

Sometimes it is useful to consider elements of $H_1(A, \mathbf{R})$, given by floating point vectors (i.e., over `RealField()`). These represent certain points of infinite order, but without further information we do not know exactly what point they represent, or even whether such a point is 0.

Elements can only be created independently of other elements by coercion, see Section 142.2.14.

142.11.1 Arithmetic

The following commands describe the basic arithmetic operations available for elements of modular abelian varieties. Operations include addition, subtraction, and multiplication by an integer, rational number, or real number.

`a * x`

Product of the integer, rational or real number a by the element x of a modular abelian variety.

`x * a`

Product of the element x of a modular abelian variety by the integer, rational or real number a.

`x + y`

The sum of elements x and y of a modular abelian variety.

`x - y`

The difference of elements x minus y of a modular abelian variety.

Example H142E98

In this example, we construct $J_0(23)$, and consider the finite subgroup $\ker(T_3 - 5)$, which has order 400. We then do various arithmetic operations with some of its elements.

```
> A := JZero(23);
> t3 := HeckeOperator(A,3);
> Factorization(CharacteristicPolynomial(t3));
[
    <x^2 - 5, 2>
]
> G := Kernel(t3-5);
> #G;
400
> Generators(G);
[
    Element of abelian variety defined by [1/10 0 1/10 1/5] modulo homology,
    Element of abelian variety defined by [0 0 0 -5/2] modulo homology,
    Element of abelian variety defined by [1/10 -1/10 0 -1/5] modulo homology,
    Element of abelian variety defined by [1 -3/2 2 1] modulo homology
]
> x := G.1;
> 1.5*x;
Element of abelian variety defined by [0.14999999999999999999999999998 0.E-28
0.14999999999999999999999999998
0.29999999999999999999999999996] modulo homology
> (3/2)*x;
Element of abelian variety defined by [3/20 0 3/20 3/10] modulo homology
> 10*x;
```

```
0
> x*1.5;
Element of abelian variety defined by [0.14999999999999999999999999998 0.E-28
0.14999999999999999999999999998
0.29999999999999999999999999996] modulo homology
> 1.5*x eq x*1.5;
true
> x*(3/2);
Element of abelian variety defined by [3/20 0 3/20 3/10] modulo homology
> x*5;
Element of abelian variety defined by [1/2 0 1/2 1] modulo homology
> G.1 + G.2;
Element of abelian variety defined by [1/10 0 1/10 -23/10] modulo homology
> G.1 - G.2;
Element of abelian variety defined by [1/10 0 1/10 27/10] modulo homology
```

142.11.2 Invariants

These commands compute information about the order of an element, the degree of the homology of the parent variety, and a field that the point is defined over.

`Order(x)`

Given an element of a modular abelian variety x, return the order of x, if x is known exactly. Otherwise an error occurs.

`ApproximateOrder(x)`

Given a point x on a modular abelian variety return the exact order of x, if x is known exactly as a torsion point, and if not the order of an approximation of x by a torsion point, obtained using continued fractions.

`Degree(x)`

The dimension of the homology of the parent of x, where x is an element of a modular abelian variety.

`FieldOfDefinition(x)`

A field that x is defined over, which need not be minimal, where x is an element of a modular abelian variety.

Example H142E99

We compute a 2-torsion point on the elliptic curve $J_0(11)$, compute some approximate orders, and compute the degree.

```
> A := JZero(11);
> G := Kernel(nIsogeny(A,2));
> G;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2 ]
> x := G.1;
> ApproximateOrder(Sqrt(2)*x);
1023286908188737
> ApproximateOrder(1.000000000000001*x);
2
> Degree(x);
2
```

Notice that `FieldOfDefinition(x)` is valid, but far from optimal. It would be better to return the number field generated by the 2-torsion point.

```
> FieldOfDefinition(x);
Algebraically closed field with no variables
> FieldOfDefinition(0*x);
Rational Field
> Order(x);
2
```

142.11.3 Predicates

These are commands for testing equality, inclusion, whether an element is 0, and whether an element is known exactly, (i.e., as an element of $H_1(A, \mathbf{Q})$, or just as an element of $H_1(A, \mathbf{R})$).

`x eq y`

Return `true` if the elements x and y of a modular abelian variety are equal.

`x in X`

Return `true` if the element x of a modular abelian variety is an element of the list X.

`IsExact(x)`

Return `true` if the element x of a modular abelian variety is known exactly, i.e., x is defined by an element of the rational homology.

IsZero(x)

Return `true` if the element x of a modular abelian variety is known exactly and is equal to 0. If x is not known exactly, return `true` if a real homology vector that represents x is "very close" to an element of the integral homology, where very close means that the distance is within $1/10^n$, where n is `M‘point_precision` and M is the parent of x.

Example H142E100

We demonstrate each of these commands using elements of the 2-torsion subgroups of the two elliptic curves of conductor 37.

```
> J := JZero(37);
> A, B := Explode(Decomposition(J));
> A;
Modular abelian variety 37A of dimension 1, level 37 and
conductor 37 over Q
> B;
Modular abelian variety 37B of dimension 1, level 37 and
conductor 37 over Q
> A2 := Kernel(nIsogeny(A,2));
> B2 := Kernel(nIsogeny(B,2));
> x := A2.1;
> y := B2.2;
> x eq y;
false
> x in [* x, y *];
true
> IsZero(x);
false
> IsZero(0*x);
true
> IsExact(1.00000000000000000001*x);
false
> IsExact((2/3)*x);
true
```

For non-exact elements, `IsZero` means "is quite close to 0".

```
> IsZero(0.0001*x);
false
> IsZero(0.00001*x);
true
> IsZero(0.000000000001*x);
true
> A‘point_precision;
10
```

142.11.4 Homomorphisms

There are two notations for applying a homomorphism to an element. One can find an inverse image of an element using the @@ command.

```
x @ phi
```

```
phi(x)
```

The image of the element x of a modular abelian variety under the homomorphism ϕ of abelian varieties.

```
x @@ phi
```

An inverse image of the element x of a modular abelian variety under the homomorphism ϕ of abelian varieties.

Example H142E101

Let $\phi = T_3 - 5$ acting on the abelian surface $J_0(23)$. We apply ϕ to an element of the kernel G of ϕ, and get 0. We also find an element y such that $\phi(y)$ is a certain element of G.

```
> A := JZero(23);
> phi := HeckeOperator(A,3) - 5;
> G := Kernel(phi);
> x := G.1;
> Order(x);
10
> phi(x);
0
> zero := A!0;
> z := zero@@phi; z;
0
> y := x@@phi; y;
Element of abelian variety defined by [-1/20 1/20 -1/20 -1/20] modulo homology
> phi(y) in G;
true
> y@phi eq phi(y);
true
```

142.11.5 Representation of Torsion Points

An exact torsion point representation of an element of a modular abelian variety can be found using continued fractions to find good rational approximations for each coordinate of a representative real homology class. A representative element of the homology can also be retrieved.

The `Eltseq` command gives the sequence of entries of the vector returned by `Element`.

`ApproximateByTorsionPoint(x : parameters)`

`Cutoff` RngIntElt *Default :* 10^3

If the modular abelian variety element x is defined by an element z in the real homology $H_1(A, R)$, find an element of $H_1(A, \mathbf{Q})$ which approximates z, using continued fractions, and return the corresponding point.

`Element(x)`

The vector in homology which represents the element x of a modular abelian variety.

`LatticeCoordinates(x)`

A vector over the rational or real field which represents the element x with respect to the basis for integral homology of the parent abelian variety of x.

`Eltseq(x)`

The `Eltseq` of `LatticeCoordinates(x)` where x is an element of a modular abelian variety.

Example H142E102

This code illustrates each of the commands for a 3-torsion point in $J_0(33)$.

```
> A := JZero(33);
> x := A![1/3,0,0,0,0,0];
> x;
Element of abelian variety defined by [1/3 0 0 0 0 0] modulo homology
> Order(x);
3
> ApproximateByTorsionPoint(1.001*x);
Element of abelian variety defined by [1001/3000 0 0 0 0 0] modulo homology
> Element(x);
(1/3   0   0   0   0   0)
> Eltseq(x);
[ 1/3, 0, 0, 0, 0, 0 ]
> LatticeCoordinates(x);
(1/3   0   0   0   0   0)
```

The `Element` and `LatticeCoordinates` can differ when the integral structure on the homology is complicated. This is common when the weight is bigger than 2.

```
> A := JZero(11,4); A;
```

```
Modular motive JZero(11,4) of dimension 2 and level 11 over Q
> x := A![1/3,0,0,0];
> Element(x);
(  1/8  1/24 -1/24 -1/24)
> Eltseq(x);
[ 1/3, 0, 0, 0 ]
> LatticeCoordinates(x);
(1/3   0   0   0)
> x;
Element of abelian variety defined by [1/3 0 0 0] modulo homology
```

142.12 Subgroups of Modular Abelian Varieties

142.12.1 Creation

Subgroups can be created from a sequence of elements of a modular abelian variety which generate it. It is also possible to create n-torsion subgroups $A[n]$ and to create subgroups as kernels of homomorphisms or by taking an image of the difference of two cusps.

If a subgroup G contains elements that are not known exactly (i.e., they are defined by floating point approximations to real homology elements), a group of torsion points that approximates G can be found.

`Subgroup(X)`

The subgroup of A generated by the nonempty sequence X of elements of modular abelian variety A.

`ZeroSubgroup(A)`

The zero subgroup of the abelian variety A.

`nTorsionSubgroup(A, n)`

The kernel $A[n]$ of the multiplication by n isogeny on the modular abelian variety A.

`nTorsionSubgroup(G, n)`

The kernel $G[n]$ of the multiplication by n homomorphism on the subgroup G of a modular abelian variety.

`ApproximateByTorsionGroup(G : parameters)`

`Cutoff` RngIntElt *Default :* 10^3

The subgroup generated by torsion approximations of a set of generators of the subgroup G of a modular abelian variety.

Example H142E103

First we list the elements of the 2-torsion subgroup of the elliptic curve 100A, then we compute the 0 subgroup.

```
> A := ModularAbelianVariety("100A"); A;
Modular abelian variety 100A of dimension 1 and level 2^2*5^2
over Q
> G := nTorsionSubgroup(A,2); G;
Finitely generated subgroup of abelian variety with invariants [ 2, 2 ]
> Elements(G);
[
    0,
    Element of abelian variety defined by [1/2 0] modulo homology,
    Element of abelian variety defined by [0 1/2] modulo homology,
    Element of abelian variety defined by [1/2 1/2] modulo homology
]
> ZeroSubgroup(A);
{ 0 }: finitely generated subgroup of abelian variety with
invariants []
```

We can also use the `nTorsionSubgroup` command on subgroups.

```
> nTorsionSubgroup(G,2);
Finitely generated subgroup of abelian variety with
invariants [ 2, 2 ]
> nTorsionSubgroup(G,3);
{ 0 }: finitely generated subgroup of abelian variety with
invariants []
```

One of the 2-torsion elements generates a subgroup H of order 2.

```
> G.1;
Element of abelian variety defined by [0 1/2] modulo homology
> H := Subgroup([G.1]); H;
Finitely generated subgroup of abelian variety
> #H;
2
```

To illustrate the approximation command, we consider the subgroup generated by an approximation to one of the 2-torsion elements.

```
> K := Subgroup([1.00001*G.1]);
> L := ApproximateByTorsionGroup(K);
Finitely generated subgroup of abelian variety with
invariants [ 2 ]
> L eq H;
true
```

142.12.2 Elements

These commands enumerate elements of a finite subgroup of a modular abelian variety and allow access to the generators.

`Elements(G)`

A sequence of all elements of the finite subgroup G of a modular abelian variety.

`Generators(G)`

A sequence of generators for the subgroup G of a modular abelian variety. These correspond to generators for the underlying abelian group.

`Ngens(G)`

The number generators of the subgroup G of a modular abelian variety.

`G . i`

The i-th generator of the subgroup G of a modular abelian variety.

Example H142E104

We illustrate each of the commands using the kernel of the Hecke operator T_3 acting on $J_0(67)$.

```
> J := JZero(67);
> T := HeckeOperator(J,3);
> G := Kernel(T);
> #G;
4
> Elements(G);
[
    0,
    Element of abelian variety defined by [0 0 1/2 0 0 -1/2 -1/2 0 1/2 0] modulo
    homology,
    Element of abelian variety defined by [1/2 -1/2 0 0 -1/2 0 0 -1/2 1/2 0]
    modulo homology,
    Element of abelian variety defined by [1/2 -1/2 1/2 0 -1/2 -1/2 -1/2 -1/2 1 0]
    modulo homology
]
> Generators(G);
[
    Element of abelian variety defined by [1/2 -1/2 0 0 -1/2 0 0 -1/2 1/2 0]
    modulo homology,
    Element of abelian variety defined by [0 0 1/2 0 0 -1/2 -1/2 0 1/2 0]
    modulo homology
]
> Ngens(G);
2
> G.1;
Element of abelian variety defined by [1/2 -1/2 0 0 -1/2 0 0 -1/2 1/2 0] modulo
homology
```

```
> G.2;
Element of abelian variety defined by [0 0 1/2 0 0 -1/2 -1/2 0 1/2 0] modulo
homology
```

142.12.3 Arithmetic

Quotients of abelian varieties by finite subgroups can be formed, finite subgroups can be intersected with other finite subgroups or abelian varieties, and the group generated by two subgroups can be computed.

For several of the arithmetic operations below, finite groups or abelian varieties are replaced by their image in a common abelian variety, so the operation makes sense. This common abelian variety is the one returned by `FindCommonEmbeddings` on page 4920. Note that the "embedding" is only guaranteed to be an embedding up to isogeny.

`Quotient(A, G)`

The quotient of the abelian variety A by the finite subgroup G of A, the isogeny $A \to A/G$ and an isogeny $A/G \to A$, such that composition of the two isogenies is multiplication by the exponent of G.

`Quotient(G)`

The quotient A/G, where A is the ambient variety of the subgroup G of an abelian variety, an isogeny from A to A/G with kernel G, and an isogeny from A/G to A such that the composition of the two isogenies is multiplication by the exponent of G.

`A / G`

Given an abelian variety A and a subgroup G of an abelian variety return the quotient A/G, the isogeny $A \to A/G$ with kernel G, and an isogeny $A/G \to A$ where A is not necessarily the ambient variety of G.

`A meet G`
`G meet A`

The intersection of the finite subgroup G of an abelian variety B with the abelian variety A. If A is not equal to B, then G and A are replaced by their image in a common abelian variety.

`G1 + G2`

The sum of the subgroups G_1 and G_2 of abelian varieties A_1 and A_2. If A_1 is not equal to A_2, then G_1 and G_2 are replaced by their image in a common abelian variety.

`G1 meet G2`

The intersection of the finite subgroups G_1 and G_2 of an abelian variety. If their ambient varieties are not equal, G_1 and G_2 are replaced by their image in a common abelian variety.

Example H142E105

We illustrate these commands using the 2-torsion of $J_0(67)$. First we compute the kernel of T_3, which is a 2-torsion group of order 4.

```
> J := JZero(67); J;
Modular abelian variety JZero(67) of dimension 5 and level 67 over Q
> T := HeckeOperator(J,3);
> Factorization(CharacteristicPolynomial(T));
[
    <x + 2, 2>,
    <x^2 - x - 1, 2>,
    <x^2 + 3*x + 1, 2>
]
> G := Kernel(T); #G;
4
```

Next we quotient $J_0(67)$ out by this subgroup of order 4.

```
> A := Quotient(J,G); A;
Modular abelian variety of dimension 5 and level 67 over Q
```

The result is, of course, isogenous to $J_0(67)$. Unfortunately, testing of isomorphism in this generality is not yet implemented.

```
> IsIsogenous(A,J);
true
> Degree(ModularParameterization(A));
4
```

If the `Quotient` command is given only one argument then the variety being quotiented out by is the ambient variety.

```
> B := Quotient(G); B;
Modular abelian variety of dimension 5 and level 67 over Q
> Degree(ModularParameterization(B));
4
```

We can also use the divides notation for quotients.

```
> C := J/G; C;
Modular abelian variety of dimension 5 and level 67 over Q
```

Next we list the 2-torsion subgroups of the simple factors of $J_0(67)$. Interestingly, the sum of the 2-torsion subgroups of these simple factors is much smaller than the full 2-torsion subgroup $J_0(67)[2]$.

```
> D := Decomposition(J); D;
[
    Modular abelian variety 67A of dimension 1, level 67 and
    conductor 67 over Q,
    Modular abelian variety 67B of dimension 2, level 67 and
    conductor 67^2 over Q,
```

```
    Modular abelian variety 67C of dimension 2, level 67 and
    conductor 67^2 over Q
]
> for A in D do print #(A meet G); end for;
4
1
1
> G2 := nTorsionSubgroup(D[2],2);
> G3 := nTorsionSubgroup(D[3],2);
> H := G + G2 + G3;
> #H;
64
> H eq nTorsionSubgroup(J,2);
false
> #nTorsionSubgroup(J,2);
1024
> G2 eq G3;
true
> G meet G2;
{ 0 }: finitely generated subgroup of abelian variety with invariants []
```

142.12.4 Underlying Abelian Group and Lattice

`AbelianGroup(G)`

Let G be a finitely generated subgroup of an abelian variety A. Return an abstract abelian group H which is isomorphic to G along with isomorphisms in both directions.

`Lattice(G)`

Let G be a finite torsion subgroup of its ambient abelian variety A whose elements are all known exactly, i.e. G is generated by elements of $H_1(A, \mathbf{Q})/H_1(A, \mathbf{Z})$. Return the lattice L in the rational homology of A generated by $H_1(A, \mathbf{Z})$ and all x such that G can be viewed as a set of equivalence classes of the form $x + H_1(A, \mathbf{Z})$.

Example H142E106

This examples illustrate these commands for the 3-torsion subgroup of $J_0(11)$.

```
> A := JZero(11);
> G := nTorsionSubgroup(A,3);
> H,f,g := AbelianGroup(G);
> H;
Abelian Group isomorphic to Z/3 + Z/3
Defined on 2 generators
Relations:
    3*H.1 = 0
```

```
    3*H.2 = 0
```

The lattice of G is 1/3 times the integral homology.

```
> Lattice(G);
Lattice of rank 2 and degree 2
Basis:
[Identity matrix]
Basis Denominator: 3
> L := IntegralHomology(A); L;
Standard Lattice of rank 2 and degree 2
> Lattice(G)/L;
Abelian Group isomorphic to Z/3 + Z/3
Defined on 2 generators
Relations:
    3*$.1 = 0
    3*$.2 = 0
```

142.12.5 Invariants

`AmbientVariety(G)`

Let G be a finitely generated subgroup of an abelian variety. Return the abelian variety whose elements were used to create G.

`Exponent(G)`

Given a finitely generated subgroup G of an abelian variety, return the smallest positive integer e which kills G, i.e. such that $eG = 0$. We assume G is finite.

`Invariants(G)`

Given a finitely generated subgroup of an abelian variety, return the invariants of an abstract abelian group isomorphic to G.

`Order(G)`

`#G`

Given a finitely generated subgroup G of an abelian variety, return the number of elements in G, when G is known to be finite (an error occurs otherwise).

`FieldOfDefinition(G)`

Given a finitely generated subgroup G of an abelian variety return a field over which the group G is defined. This is a field K such that if σ is an automorphism that fixes K, then $\sigma(G) = G$. Note that K *is not guaranteed to be minimal.*

Example H142E107

We illustrate each command using the kernel of T_3 on $J_0(67)$.

```
> A := JZero(67);
> T3 := HeckeOperator(A,3);
> G := Kernel(T3); G;
Finitely generated subgroup of abelian variety with
[ 2, 2 ]
> AmbientVariety(G);
Modular abelian variety JZero(67) of dimension 5 and level 67 over Q
> Exponent(G);
2
> Invariants(G);
[ 2, 2 ]
> Order(G);
4
> #G;
4
```

The field of definition of G is $\mathbf{Q}$, since G is the kernel of a homomorphism defined over $\mathbf{Q}$ (a Hecke operator).

```
> FieldOfDefinition(G);
Rational Field
```

However, the field of definition of the subgroup of G generated by one of the elements of G could take significant extra work to determine. Currently MAGMA simply chooses the easiest answer, which is $\overline{\mathbf{Q}}$.

```
> H := Subgroup([G.1]);
> FieldOfDefinition(H);
Algebraically closed field with no variables
```

142.12.6 Predicates and Comparisons

A subgroup G of an abelian variety is finite exactly when every element of G is known exactly, since then all elements are torsion and G is finitely generated.

Abelian varieties and subgroups thereof can be tested for inclusion and equality. Equality and subset testing is liberal, in that if the ambient varieties containing the two groups are not equal, then MAGMA attempts to find a natural embedding of both subgroups into a common ambient variety, and checks equality or inclusion there.

`IsFinite(G)`

Return `true` if the subgroup G of a modular abelian variety is known to be finite, i.e., generated by torsion elements. If G is not known exactly, i.e., has elements defined by floating point approximations to homology, then return `false`.

`G1 subset G2`

Return `true` if $G1$ is a subset of $G2$, where $G1$ and $G2$ are both subgroups of modular abelian varieties.

`G subset A`

Return `true` if the subgroup G is a subset of the abelian variety A. If A is not the ambient variety of G, then G and A are first mapped to a common ambient variety and compared.

`A subset G`

Return `true` if the abelian variety A is a subset of the finitely generated subgroup G of a modular abelian variety. This is true only if A is a point, i.e., the 0 dimensional abelian variety.

`G1 eq G2`

Return `true` if the subgroups G_1 and G_2 of modular abelian varieties are equal.

Example H142E108

We work with $J_0(389)$, but work in the $+1$ quotient of homology for efficiency. First we let A and B be the first and fifth factors in the decomposition of J, and let G and H be the corresponding 5-torsion subgroups.

```
> J := JZero(389,2,+1);
> D := Decomposition(J);
> A := D[1];
> B := D[5];
> G := nTorsionSubgroup(A,5);
> H := nTorsionSubgroup(B,5);
```

Note that the torsion subgroups aren't as big because we are working in the $+1$ quotient.

```
> #G;
5
> #H;
95367431640625
```

We now demonstrate each of the above commands for A, B, G, and H.

```
> IsFinite(G);
true
> A subset G;
false
> ZeroModularAbelianVariety() subset G;
true
> G subset A;
true
> G subset B;
true
> H subset A;
```

```
false
```

Since the ambient varieties of G and H are A and B, respectively, the following commands implicitly embed G and H into $J_0(389)$ and make comparisons there.

```
> G subset H;
true
> G eq H;
false
> G eq G;
true

> J := JZero(37);
> A, B := Explode(Decomposition(J));
> A2 := Kernel(nIsogeny(A,2));
> B2 := Kernel(nIsogeny(B,2));
> A2 eq B2;
true
> x := A2.1;
> x in B2;   // uses embedding of both into $J_0(37)$.
true
```

142.13 Rational Torsion Subgroups

The following functions are used for computing information about certain torsion points on modular abelian varieties.

142.13.1 Cuspidal Subgroup

For simplicity, assume A is a modular abelian variety and $\pi : J_0(N) \to A$ is the modular parameterization (the case when $J_0(N)$ is replaced by a more general modular abelian variety is similar). The *cuspidal subgroup* of $J_0(N)$ is the finite torsion group generated by all classes of differences of cusps on $X_0(N)$. The *cuspidal subgroup* of $A(\overline{\mathbf{Q}})$ is the image under π of the cuspidal subgroup of $J_0(N)$. The *rational cuspidal subgroup* is the subgroup generated by differences of cusps that are defined over $\mathbf{Q}$. (Computation of the rational points in the cuspidal subgroup has not yet been implemented.) One important use of the rational cuspidal subgroup is that it gives a lower bound on the cardinality (and structure) of the torsion subgroup of $A(\mathbf{Q})$, which is important in computations involving the Birch and Swinnerton-Dyer conjecture.

`CuspidalSubgroup(A)`

The subgroup of the abelian variety A generated by all differences of cusps, where we view A as a quotient of a modular symbols abelian variety. Note that this subgroup need not be defined over $\mathbf{Q}$.

`RationalCuspidalSubgroup(A)`

Return the finite subgroup of the abelian variety A generated by all differences of $\mathbf{Q}$-rational cusps, where we view A in some way as a quotient of a modular symbols abelian variety.

Example H142E109

We compute the cuspidal and rational cuspidal subgroups of $J_0(100)$.

```
> J := JZero(100);
> G := CuspidalSubgroup(J); G;
Finitely generated subgroup of abelian variety with invariants
[ 6, 30, 30, 30, 30 ]
> [Eltseq(x) : x in Generators(G)];
[
    [ 29/30, -2/5, 16/15, 121/30, -2, -61/30, 3/5, 31/15, 1,
    -89/30, -7/2, -3/2, -3, -1 ],
    [ 1, -5/6, 0, -1, -1, 2/3, -3/2, 0, -2, 0, 2, 5/3, 5/6, 0 ],
    [ -2, 17/15, 1/10, -29/15, 89/30, 26/15, 7/10, -2, 2/5, 2,
    -3/10, -7/30, 59/30, -1/2 ],
    [ 29/30, -1, 1/2, 67/15, -2, -3/2, 14/15, 91/30, 1, -3,
    -38/15, -29/30, -91/30, 1/2 ],
    [ 31/30, -31/30, 2/5, 67/15, -29/15, -43/30, 5/6, 3, 1, -3,
    -13/5, -31/30, -91/30, 0 ]
]
> H := RationalCuspidalSubgroup(J); H;
Finitely generated subgroup of abelian variety with invariants
[ 3, 15, 30 ]
```

Next we compute the cuspidal and rational subgroups for the optimal new elliptic curve of conductor 100.

```
> D := Decomposition(J); A := D[1];
> CuspidalSubgroup(A);
Finitely generated subgroup of abelian variety with invariants
[ 2, 2 ]
> Generators(CuspidalSubgroup(A));
[
    Element of abelian variety defined by [0 1/2] modulo homology,
    Element of abelian variety defined by [1/2 0] modulo homology
]
> RationalCuspidalSubgroup(A);
Finitely generated subgroup of abelian variety with invariants []
> TorsionMultiple(A);
2
```

Because the torsion multiple is 2, some of the cuspidal subgroup can not be defined over $\mathbf{Q}$.

142.13.2 Upper and Lower Bounds

Let A be an abelian variety over a number field K. MAGMA can compute upper and lower bounds on the cardinality of the torsion subgroup of $A(K)$ in the form of a multiple and a divisor of this cardinality.

`TorsionLowerBound(A)`

Given an abelian variety A return a divisor of the cardinality of the K-rational torsion subgroup of $A(K)$. Currently, to compute a bound we require that A be the base extension of an abelian variety B over $\mathbf{Q}$, and the lower bound is simply the cardinality of the rational cuspidal subgroup of $B(\mathbf{Q})$.

`TorsionMultiple(A)`

`TorsionMultiple(A, n)`

Given an abelian variety A return a multiple of the cardinality of the K-rational torsion subgroup of A over K. If n is not given it is assumed to be 50.

This multiple is usually fairly sharp, and is computed as follows. For each good prime $p \leq n$ with $[K : Q] + 1 < p$ and such that p does not divide the level of A, MAGMA computes $\#A(k)$, where k varies over residue class fields of K of characteristic p. Since reduction on torsion is injective for such primes, the greatest common divisor of the $\#A(k)$ is a multiple of the order of the torsion subgroup of $A(K)$. MAGMA computes $\#A(k)$ by using Hecke operators to find the characteristic polynomial of Frobenius on a Tate module of A, and uses this characteristic polynomial to deduce $\#A(k)$. For details, see [AS05].

Example H142E110

```
> J := JZero(100);
> TorsionLowerBound(J);
1350
> #RationalCuspidalSubgroup(J);
1350
> TorsionMultiple(J);
16200
> 16200/1350;
12
> J2 := BaseExtend(J,QuadraticField(2));
> TorsionMultiple(J2);
129600
> 129600/16200;
8
```

142.13.3 Torsion Subgroup

`TorsionSubgroup(A)`

Let A be an abelian variety over a field K. Attempt to compute the subgroup of torsion elements in $A(K)$. Return either `false` and a subgroup of the torsion subgroup, or `true` and the exact torsion subgroup of A over the base field.

Example H142E111

```
> TorsionSubgroup(JZero(11));
true Finitely generated subgroup of abelian variety with invariants [ 5 ]
> TorsionSubgroup(JZero(33));
false Finitely generated subgroup of abelian variety with
invariants [ 10, 10 ]
> TorsionSubgroup(BaseExtend(JZero(11),QuadraticField(5)));
true Finitely generated subgroup of abelian variety with
invariants [ 5 ]
> TorsionSubgroup(ChangeRing(JZero(11),GF(5)));
false { 0 }: finitely generated subgroup of abelian variety with
invariants []
> TorsionSubgroup(JZero(100));
false Finitely generated subgroup of abelian variety with
invariants [ 3, 15, 30 ]
> TorsionSubgroup(JZero(125));
true Finitely generated subgroup of abelian variety with
invariants [ 25 ]
```

142.14 Hecke and Atkin-Lehner Operators

142.14.1 Creation

These commands compute endomorphisms induced by the Atkin-Lehner and Hecke operators on modular abelian varieties. The Atkin-Lehner involution W_q is defined for each positive integer q that exactly divides the level (and is divisible by the conductor of any relevant character).

`AtkinLehnerOperator(A, q)`

The Atkin-Lehner operator W_q of index q induced on the abelian variety A by virtue of A being modular. In general W_q need not be a morphism except in the category of abelian varieties up to isogeny so this intrinsic also returns an integer d such that $d * W_q$ is an endomorphism of A, and when W_q doesn't leave A invariant, returns $d = 0$. If the ambient modular symbols space of A contains a space with character of conductor r, then currently an error occurs unless r divides q.

`AtkinLehnerOperator(A)`

The morphism (or morphism tensor Q) on (or from) the abelian variety A induced by the Atkin-Lehner operator.

`HeckeOperator(A, n)`

The Hecke operator T_n of index n induced on the abelian variety A by virtue of its morphism to a modular symbols abelian variety. In general T_n need not be a morphism. Also, if A is contained in e.g., $J_0(N)$, then the T_n on $J_0(N)$ need not even leave A invariant. In that case this command composes T_n with a map back to A to obtain an endomorphism of A. For the exact Hecke operators induced by their action on $J_0(N)$, say, use the `RestrictEndomorphism` command.

Example H142E112

We compute the main Atkin-Lehner operator and the Hecke operator T_2 on $J_0(23)$.

```
> A := JZero(23);
> AtkinLehnerOperator(A,23);
Homomorphism W23 from JZero(23) to JZero(23) given on integral homology by:
[-1  0  0  0]
[ 0 -1  0  0]
[ 0  0 -1  0]
[ 0  0  0 -1]
> HeckeOperator(A,2);
Homomorphism T2 from JZero(23) to JZero(23) given on integral homology by:
[ 0  1 -1  0]
[ 0  1 -1  1]
[-1  2 -2  1]
[-1  1  0 -1]
```

Next we compute w_4 and w_{25} on J_{100}, and note that their product equals w_{100}.

```
> A := JZero(100); A;
Modular abelian variety JZero(100) of dimension 7 and
level 2^2*5^2 over Q
> w4 := AtkinLehnerOperator(A,4);
> Factorization(CharacteristicPolynomial(w4));
[
    <x - 1, 4>,
    <x + 1, 10>
]
> w25 := AtkinLehnerOperator(A,25);
> Factorization(CharacteristicPolynomial(w25));
[
    <x - 1, 8>,
    <x + 1, 6>
]
> w4*w25 eq AtkinLehnerOperator(A);
```

```
true
```

Next we compute W_{25} acting on $J_1(25)$.

```
> A := Js(17);
> B := BaseExtend(A,CyclotomicField(17));
> w := AtkinLehnerOperator(B);
> Factorization(CharacteristicPolynomial(w));
[
    <x - 1, 4>,
    <x + 1, 6>
]
```

Finally we compute Hecke operators on the quotient of a simple factor of $J_0(65)$ by a finite subgroup.

```
> A := Decomposition(JZero(65))[2]; A;
Modular abelian variety 65B of dimension 2, level 5*13 and conductor
5^2*13^2 over Q
> G := nTorsionSubgroup(A,2); G;
Finitely generated subgroup of abelian variety with invariants
[ 2, 2, 2, 2 ]
> H := Subgroup([G.1]); H;
Finitely generated subgroup of abelian variety with invariants [ 2 ]
> B := A/H; B;
Modular abelian variety of dimension 2 and level 5*13 over Qbar
> T2 := HeckeOperator(B,2); T2;
Homomorphism from modular abelian variety of dimension 2 to
modular abelian variety of dimension 2 (up to isogeny) on
integral homology by:
[ -2 1/2   0   0]
[ -2   2   0   0]
[ -2   1  -2   1]
[ -6   1  -1   2]
> FactoredCharacteristicPolynomial(T2);
[
    <x^2 - 3, 2>
]
```

142.14.2 Invariants

Intrinsics are provided which compute characteristic polynomials, factored characteristic polynomials and minimal polynomials of Hecke operators.

`HeckePolynomial(A, n)`

The characteristic polynomial of the Hecke operator T_n acting on the abelian variety A.

`FactoredHeckePolynomial(A, n)`

The factored characteristic polynomial of the Hecke operator T_n acting on the abelian variety A. This can be faster than first computing T_n, then computing the characteristic polynomial, and factoring, because we can take into account information about the decomposition of A, in order to avoid factoring.

`MinimalHeckePolynomial(A, n)`

The minimal polynomial of the Hecke operator T_n acting on the abelian variety A.

Example H142E113

```
> FactoredHeckePolynomial(JZero(65),2);
[
    <x + 1, 2>,
    <x^2 - 3, 2>,
    <x^2 + 2*x - 1, 2>
]
> HeckePolynomial(JZero(65),2);
x^10 + 6*x^9 + 5*x^8 - 32*x^7 - 62*x^6 + 28*x^5 + 130*x^4 +
    48*x^3 - 51*x^2 - 18*x + 9
> MinimalHeckePolynomial(JZero(65),2);
x^5 + 3*x^4 - 2*x^3 - 10*x^2 - 3*x + 3
```

142.15 L-series

142.15.1 Creation

The `LSeries` command creates the L-series $L(A, s)$ associated to a modular abelian variety A over $\mathbf{Q}$ or a cyclotomic field. No actual computation is performed.

`LSeries(A)`

The L-series associated to the abelian variety A.

Example H142E114

```
> A := JZero(23);
> L := LSeries(A);
> L;
L(JZero(23),s): L-series of Modular abelian variety JZero(23) of
dimension 2 and level 23 over Q
> LSeries(ModularAbelianVariety("65B"));
L(65B,s): L-series of Modular abelian variety 65B of dimension 2
and level 5*13 over Q
```

You can create L-series of abelian varieties over cyclotomic fields, but currently no interesting functionality is implemented for them.

```
> LSeries(BaseExtend(JZero(11),CyclotomicField(5)));
L(JZero(11),s): L-series of Modular abelian variety JZero(11) of
dimension 1 and level 11 over Q(zeta_5)
```

142.15.2 Invariants

`CriticalStrip(L)`

Let L be the L-function of some modular abelian variety A. This function returns integers x and y such that the critical strip for L is the set of complex numbers with real part strictly between x and y. If W is the set of weights of newforms that give rise to factors of A, then this command returns 0 and `Max(W)`.

`ModularAbelianVariety(L)`

The abelian variety the L-series L is associated to.

Example H142E115

We define several L-functions of modular abelian varieties and modular motives, and compute their critical strip (which is from 0 to k, where k is the weight).

```
> L := LSeries(JZero(37));
> CriticalStrip(L);
0 2
> L := LSeries(JZero(37,6));
> CriticalStrip(L);
0 6
> J := JOne(11,3);  J;
Modular motive JOne(11,3) of dimension 5 and level 11 over Q
> CriticalStrip(LSeries(J));
0 3
> A_delta := JZero(1,12);
> L := LSeries(A_delta);
> CriticalStrip(L);
```

```
0 12
> ModularAbelianVariety(L);
Modular motive JZero(1,12) of dimension 1 and level 1 over Q
```

142.15.3 Characteristic Polynomials of Frobenius Elements

Let A be a modular abelian variety. The characteristic polynomials of Frobenius elements acting on the ℓ-adic Tate modules of A define the local L-factors of $L(A, s)$.

`FrobeniusPolynomial(A : parameters)`

`Factored` BoolElt *Default :* `false`

The characteristic polynomial of Frobenius on the abelian variety A defined over a finite field. If `Factored` is set to `true`, return a factorization instead of a polynomial.

`FrobeniusPolynomial(A, p : parameters)`

`Factored` BoolElt *Default :* `false`

The characteristic polynomial of Frob_p acting on any ℓ-adic Tate module of the abelian variety A over a number field, where p and ℓ do not divide the level of A. If the base ring has degree bigger than 1, then return a sequence of characteristic polynomials, one for each prime lying over p, sorted by degree. If `Factored` is set to `true`, return a factorization instead of a polynomial.

`FrobeniusPolynomial(A, P)`

The characteristic polynomial of Frobenius at the nonzero prime ideal P of a number field on the modular abelian variety A, where P is assumed to be a prime of good reduction for A, and A is defined over a field that contains the prime P.

Example H142E116

```
> A := JZero(23);
> FrobeniusPolynomial(A,2);
x^4 + x^3 + 3*x^2 + 2*x + 4
> A := JZero(23) * JZero(11,4) * JOne(13);
> FrobeniusPolynomial(A,2);
x^12 + 2*x^11 + 17*x^10 + 40*x^9 + 145*x^8 + 362*x^7 + 798*x^6 +
    1408*x^5 + 2104*x^4 + 2528*x^3 + 2528*x^2 + 1792*x + 1024
> Factorization($1);
[
    <x^4 - 2*x^3 + 14*x^2 - 16*x + 64, 1>,
    <x^4 + x^3 + 3*x^2 + 2*x + 4, 1>,
    <x^4 + 3*x^3 + 5*x^2 + 6*x + 4, 1>
]
> A := BaseExtend(JZero(23),CyclotomicField(22));
> FrobeniusPolynomial(A,2);
```

```
[
    x^4 + 25*x^3 - 327*x^2 + 25600*x + 1048576
]
```

These characteristic polynomials are used in the algorithm to compute the number of points on modular abelian varieties over finite fields.

```
> A := ChangeRing(JZero(23),GF(2^10));
> NumberOfRationalPoints(A);
1073875 1073875
> Factorization($1);
[ <5, 3>, <11, 2>, <71, 1> ]
```

142.15.4 Values at Integers in the Critical Strip

MAGMA allows evaluation of L-series at integers lying within the critical strip.

There exist algorithms for computing $L(A, s)$ for any complex number s, but these are not currently implemented in MAGMA.

`L(s)`

`Evaluate(L, s)`

`Evaluate(L, s, prec)`

The value of L-series L at s, where s must be an integer that lies in the critical strip for L, computed using *prec* terms of the power series or 100 if *prec* is not given. The power series used are the q-expansions of modular forms corresponding to differentials on A. It is not clear, a priori, what the relation is between *prec* and the precision of the real number output by this command. (It is theoretically possible to give bounds, but we have not done this.) In practice, one can increase *prec* and see how the output result changes.

`LRatio(A, s)`

`LRatio(L, s)`

Given an abelian variety A over $\mathbf{Q}$ attached to a newform or an L-series L of an abelian variety and an integer s, return the ratio $L(A, s) * (s-1)!/((2\pi)^{s-1} * \Omega_s)$, where s is a "critical integer", and Ω_s is the integral (Neron) volume of the group of real points on the optimal quotient A' associated to A when s is odd, and the volume of the -1 eigenspace for conjugation when s is even.

`IsZeroAt(L, s)`

Given an L-series L of a modular abelian variety and an integer s in the critical strip for L return `true` is $L(A, s)$ is zero. In contrast to the output of the `Evaluate` command above, the result returned by this command is provably correct.

Example H142E117

First we demonstrate each evaluation command for the L-series of $J_0(23)$.

```
> L := LSeries(JZero(23));
> L(1);
0.24843186659059928468330576929 +  0.E-29*i
> Evaluate(L,1);
0.24843186659059928468330576929 +  0.E-29*i
> Evaluate(L,1,200);
0.24843186659059968120725033907 +  0.E-29*i
> LRatio(L,1);
1/11
> L := LSeries(JZero(23));
> L(1);
0.24843186659059928468330577047 +  0.E-29*i
> Evaluate(L,1,200);
0.24843186659059968120725034014 +  0.E-29*i
```

Next we compute the L-series of the motive attached to the weight 12 level 1 modular form Δ.

```
> A := JZero(1,12);
> L := LSeries(A);
> Evaluate(L,1);
0.037441281268515541738770315844 3
> L(5);
0.66670918843400364382613022164
> Evaluate(L,1,200);
0.037441281268515541738770315844 3
> LRatio(L,1);
11340/691
> LRatio(L,2);
24
> LRatio(L,3);
7
```

We compute some ratios for $J_1(N)$ and factors of $J_1(N)$.

```
> LRatio(JOne(13),1);
1/361
> J := JOne(23);
> Evaluate(LSeries(J),1);
0.00000008077769707478577542009070006 6 +
0.00000005367962127748221777320766933 2*i
```

It looks kind of like $L(J_1(23), 1)$ is zero. However, this is not the case! We can not compute `LRatio` for $J_1(23)$, since it not attached to a newform. We can, however, compute `LRatio` for each simple factor.

```
> LRatio(J(1),1);
1/11
```

```
> LRatio(J(2),1);
1/1382426761
```

Each simple factor has nonzero `LRatio`, so $L(J,1) \neq 0$.

142.15.5 Leading Coefficient

The L-function $L(A,s)$ has a Taylor expansion about any critical integer. The leading coefficient and order of vanishing of $L(A,s)$ about a critical integer can be computed.

`LeadingCoefficient(L, s, prec)`

Given an L-series L associated to a modular abelian variety A and an integer s in the critical strip for L return the leading coefficient of the Taylor expansion about s and the order of vanishing of L at s. At present, A must have weight 2 and trivial character (so $s = 1$). It does not have to be attached to a newform. The argument *prec* is the number of terms of the power series which are used.

Example H142E118

```
> LeadingCoefficient(LSeries(JZero(37)),1,100);
0.24426406492583898134986778296 5
> LeadingCoefficient(LSeries(JZero(37)(1)) ,1,100);
0.30599977380008529004409407572 5
> J := JZero(3^5);
> LeadingCoefficient(LSeries(J),1,100);
15.140660788463628991688955015326 +  0.E-27*i 4
```

The order of vanishing of 4 for $J_0(3^5)$ comes from an elliptic curve and a 3-dimensional abelian variety that have order of vanishing 1 and 3, respectively.

```
> LeadingCoefficient(LSeries(J(1)),1,100);
1.4192096493382156160031880842 81
> LeadingCoefficient(LSeries(J(5)),1,100);
1.2280519528591420520347698584 45 3
```

Example H142E119

We give a few more examples.

```
> L := LSeries(ModularAbelianVariety("389A",+1));
> LeadingCoefficient(L,1,100);
0.75931650029224679065762600319 2
>
> A := JZero(65)(2); A;
Modular abelian variety 65B of dimension 2, level 5*13 and
conductor 5^2*13^2 over Q
> L := LSeries(A);
> LeadingCoefficient(L,1,100);
```

```
0.91225158869818984109351402175 +  0.E-29*i 0
> A := JZero(65)(3); A;
Modular abelian variety 65C of dimension 2, level 5*13 and
conductor 5^2*13^2 over Q
> L := LSeries(A);
> LeadingCoefficient(L,1,100);
0.45206792176803106991748613500 0 +  0.E-29*i 0
```

142.16 Complex Period Lattice

142.16.1 Period Map

Let A be a modular abelian variety. The period mapping of A is a map from the rational homology of A to a complex vector space.

`PeriodMapping(A, prec)`

The complex period mapping from the rational homology of the abelian variety A to $\mathbf{C}^d$, where $d = \dim A$, computed using *prec* terms of q-expansions.

142.16.2 Period Lattice

`Periods(A, n)`

Given an abelian variety A and an integer n return generators for the complex period lattice of A, computed using n terms of q-expansions. We use the map from A to a modular symbols abelian variety to define the period mapping (so this map must be injective).

142.17 Tamagawa Numbers and Component Groups of Neron Models

142.17.1 Component Groups

Suppose A is a newform modular abelian variety over $\mathbf{Q}$ over level N. For any prime p that exactly divides N, the order of the component group of A over the algebraic closure of $\mathbf{F}_p$ can be computed. The nontrivial algorithm is described in [CS01] and [KS00]. It is an open problem to compute the structure of the component group or the order under more general hypothesis.

`ComponentGroupOrder(A, p)`

The order of the component group of the special fiber of the Neron model of A over the algebraic closure of $\mathbf{F}_p$. The abelian variety A must be attached to a newform.

Example H142E120

```
> J := JZero(65); J;
Modular abelian variety JZero(65) of dimension 5 and level 5*13 over Q
> A := Decomposition(J)[3];
> ComponentGroupOrder(A,13);
1
> ComponentGroupOrder(A,5);
7
```

142.17.2 Tamagawa Numbers

Suppose A is an abelian variety over $\mathbf{Q}$ that is attached to a newform. A divisor and an integer some power of which is a multiple of the Tamagawa number of A at a prime p can be determined. When p^2 divides the level, the reduction is additive, we use the Lenstra-Oort bound from [LO85].

`TamagawaNumber(A, p)`

A divisor of the Tamagawa number of the abelian variety A at the prime p and an integer some power of which is a multiple of the Tamagawa number of A at p. Also return `true` if the divisor of the Tamagawa number is provably equal to the Tamagawa number of A. The abelian variety A must be attached to a newform.

`TamagawaNumber(A)`

Let c be the product of the Tamagawa numbers of A at primes of bad reduction, where A is an abelian variety over $\mathbf{Q}$ attached to a newform. This command returns a divisor of c, an integer some power of which is a multiple of c, and `true` if the divisor is provably equal to c.

Example H142E121

```
> J := JZero(65);
> TamagawaNumber(J(2),5);
2 2 false
> TamagawaNumber(J(2),13);
3 3 true
> TamagawaNumber(J(3),5);
7 7 true
> TamagawaNumber(J(3),13);
2 2 false
>
> J := JZero(5^2*7);
> TamagawaNumber(J(1));
2 30 false
> TamagawaNumber(J(1),5);
```

```
1 30 false
> TamagawaNumber(J(1),7);
2 2 false
```

142.18 Elliptic Curves

142.18.1 Creation

Modular abelian varieties of dimension 1 are elliptic curves. Given a modular abelian variety A over $\mathbf{Q}$ of dimension 1, MAGMA can compute an elliptic curve that is isogenous over $\mathbf{Q}$ to A. Given an elliptic curve E over $\mathbf{Q}$, a modular abelian variety over $\mathbf{Q}$ that is isogenous to E can be constructed.

It would be very desirable to make these commands more precise, and to extend them to work over other fields. For example, modular abelian varieties should (conjecturally) be associated to $\mathbf{Q}$-curves and their restriction of scalars.

`EllipticCurve(A)`

An elliptic curve isogenous to the modular abelian variety A over the rational field, if there is an elliptic curve associated to A. For A of weight greater than 2 use the `EllipticInvariants` command.

`ModularAbelianVariety(E)`

`Sign` RNGINTELT *Default :* 0

A modular abelian variety isogenous to the elliptic curve E. Note that elliptic curves with small coefficients can have quite large conductor, hence computing the massive modular abelian variety that has E as quotient, which is one thing this function does, could take some time.

Example H142E122

We apply the above two commands to the elliptic curve $J_0(49)$.

```
> A := JZero(49);
> E := EllipticCurve(A); E;
Elliptic Curve defined by y^2 + x*y = x^3 - x^2 - 2*x - 1 over
Rational Field
> B := ModularAbelianVariety(E); B;
Modular abelian variety 'Cremona 49A' of dimension 1 and level
7^2 over Q
```

To see how A and B compare, we first test equality and see they are not equal (since they were constructed differently). However, they are isomorphic.

```
> B eq A;
false
> IsIsomorphic(A,B);
```

```
true Homomorphism from JZero(49) to 'Cremona 49A' given on integral
homology by:
[1 0]
[0 1]
> phi := NaturalMap(A,B);
> Degree(phi);
1
> phi;
Homomorphism N(1) from JZero(49) to 'Cremona 49A' given on integral
homology by:
[1 0]
[0 1]
```

Thus B is embedded in A via the identity map.

142.18.2 Invariants

Let A be an abelian variety over $\mathbf{Q}$ of dimension 1. The following two functions use standard iterative algorithms (see Cremona's book) to compute the invariants c_4, c_6, j, and generators of the period lattice of the optimal quotient of $J_0(N)$ associated to A.

`EllipticInvariants(A, n)`

Invariants c_4, c_6, j, and an elliptic curve, of the one dimensional modular abelian variety A, computed using n terms of q-expansion.

`EllipticPeriods(A, n)`

Elliptic periods w_1 and w_2 of the $J_0(N)$-optimal elliptic curve associated to the modular abelian variety A, computed using n terms of the q-expansion. The periods have the property that w_1/w_2 has positive imaginary part.

Example H142E123

```
> A := ModularAbelianVariety("100A");
> c4,c6,j,E := EllipticInvariants(A,100);
> c4;
1600.052318304045803306867849111720 8 +  0.E-25*i
> c6;
-44002.166592330033618811790218678607 +  0.E-24*i
> j;
3276.801127299202275905948170653 93 +  0.E-25*i
> E;
Elliptic Curve defined by y^2 = x^3 +
(-43201.412594209236689285431925551172 +  0.E-24*i)*x +
(2376116.995985821815415836671800373 00 +  0.E-22*i) over Complex
Field
> jInvariant(E);
```

```
3276.80112729920227590594817070563 +  0.E-25*i
> w1,w2 := EllipticPeriods(A,100);
> w1;
1.26308870071276069371281657330245009108 8 +  0.E-38*i
> w2;
 0.E-38 - 1.01702927066995984919787906165005620863321*i
> w1/w2;
 0.E-38 + 1.24193937887422962244668740609486508404 97*i
```

142.19 Bibliography

[**AS04**] A. Agashe and W. A. Stein. The Manin Constant, Congruence Primes, and the Modular Degree. Preprint, URL:http://modular.fas.harvard.edu/papers/manin-agashe/, 2004.

[**AS05**] Amod Agashe and William Stein. Visible evidence for the Birch and Swinnerton-Dyer conjecture for modular abelian varieties of analytic rank zero. *Math. Comp.*, 74(249):455–484 (electronic), 2005. With an appendix by J. Cremona and B. Mazur.

[**Bos00**] Wieb Bosma, editor. *ANTS IV*, volume 1838 of *LNCS*. Springer-Verlag, 2000.

[**CS01**] Brian Conrad and William A. Stein. Component groups of purely toric quotients. *Math. Res. Lett.*, 8(5-6):745–766, 2001.

[**Kil02**] L. J. P. Kilford. Some non-Gorenstein Hecke algebras attached to spaces of modular forms. *J. Number Theory*, 97(1):157–164, 2002.

[**KS00**] David R. Kohel and William A. Stein. Component Groups of Quotients of $J_0(N)$. In Bosma [Bos00].

[**LO85**] H. W. Lenstra, Jr. and F. Oort. Abelian varieties having purely additive reduction. *J. Pure Appl. Algebra*, 36(3):281–298, 1985.

[**Que09**] Jordi Quer. Fields of definition of building blocks. *Math. Comp.*, 78(265):537–554, 2009.

143 HILBERT MODULAR FORMS

Chapter 143
HILBERT MODULAR FORMS

143.1 Introduction

The first version of this package was released in V2.15 (December 2008). It has been developed further in each subsequent release, and is still under development. We encourage users to send feedback regarding the package, and desirable features or improvements.

The package contains implementations of two separate algorithms. The primary focus is on parallel weight 2. Higher weight spaces (including non-parallel weight) are also handled; however some features are only available for weight 2, and the main routines are better optimized in weight 2. All levels $\Gamma_0(N)$ are allowed; some spaces with nontrivial character are also handled.

The main purposes of the current functionality are to efficiently compute Hecke operators on these spaces, to decompose spaces into newforms, and to efficiently obtain large numbers of eigenvalues for newforms (at least those having small degree). Some additional features such as Atkin-Lehner operators are also included.

143.1.1 Definitions and Background

Hilbert modular forms are a generalization of classical modular forms where the modular group is replaced by a subgroup of $\mathrm{GL}_2(\mathbf{Z}_F)$ where $\mathbf{Z}_F$ is the ring of integers in a totally real number field. This section gives a brief and partly informal introduction to Hilbert modular forms. The sections about the algorithms also introduce quaternionic modular forms and the Jacquet-Langlands theory in order to use the package!

The books by Freitag [Fre90] and Garrett [Gar90] are good references for standard material on Hilbert modular forms.

Let F be a totally real field of degree n over $\mathbf{Q}$, with ring of integers $\mathbf{Z}_F$. Let $v_1, \ldots, v_n$ be the embeddings of F into $\mathbf{R}$. For $x \in F$, we write x_i as a shorthand for $v_i(x)$. Each embedding v_i induces an embedding $v_i : \mathrm{Mat}_2(\mathrm{F}) \hookrightarrow \mathrm{Mat}_2(\mathbf{R})$. Extending our shorthand to matrices, for $\gamma \in \mathrm{Mat}_2(\mathrm{F})$ we write γ for $v_i(\gamma)$. Let

$$\mathrm{GL}_2^+(F) = \{\gamma \in \mathrm{GL}_2(F) : \mathrm{det}\gamma_\mathrm{i} > 0 \text{ for all i}\}.$$

The group $\mathrm{GL}_2^+(F)$ acts on the cartesian product H^n by the rule

$$\gamma(\tau, \ldots, \tau_n) = (\gamma\tau_1, \ldots, \gamma\tau_n)$$

where as usual $\mathrm{GL}_2^+(\mathbf{R})$ acts on H by linear fractional transformations.

Let $\gamma = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{GL}_2(\mathbf{R})$ and $\tau \in \mathrm{H}$. We define

$$j(\gamma, \tau) = c\tau + d \in \mathbf{C}.$$

For a *weight* $k = (k_1, \ldots, k_n) \in (\mathbf{Z}_{>0})^n$, we define a right *weight-k action* of $\mathrm{GL}_2^+(F)$ on the space of complex-valued functions on H^n by

$$(f|_k\gamma)(\tau) = f(\gamma\tau)\prod_{i=1}^{n}(\det\gamma)^{\mathrm{k_i}/2}\mathrm{j}(\gamma,\tau_\mathrm{i})^{-\mathrm{k_i}}$$

for $f : \mathrm{H}^n \to \mathbf{C}$ and $\gamma \in \mathrm{GL}_2^+(F)$. The center $F^\times$ of $\mathrm{GL}_2^+(F)$ acts trivially on such f. Therefore, the weight-k action descends to an action of $\mathrm{PGL}_2^+(F) = \mathrm{GL}_2^+(F)/F^\times$.

Now let N be a (nonzero) ideal of $\mathbf{Z}_F$. Define

$$\Gamma_0(N) = \left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{GL}_2^+(\mathbf{Z}_F) : c \in N \right\}.$$

A *Hilbert modular cusp form of weight-k and level* $\Gamma_0(N)$ is an analytic function $f : \mathrm{H}^n \to \mathbf{C}$ such that $f|_k\gamma = f$ for all $\gamma \in \Gamma_0(N)$ and such that f vanishes at the cusps of $\Gamma_0(N)$.

When F has narrow class number 1, it is possible to define double coset operators in the same way as for classical modular forms; among these are Hecke operators, Atkin-Lehner operators and degeneracy maps. More precisely, the definitions work for primes in the trivial ideal class. In the general case, the adelic definition of Hilbert modular forms is more convenient for the purpose of defining operators, among other considerations. We refer to the book by Freitag [Fre90].

143.1.2 Algorithms and the Jacquet-Langlands Correspondence

Two separate algorithms are implemented for computing the Hecke action on cuspidal spaces of Hilbert modular forms. Both rely on the *Jacquet-Langlands correspondence*, which states that the Hecke action on a space of Hilbert cusp forms is the same as the Hecke action on a space of automorphic forms on some order in a suitable quaternion algebra. See [Hid06] for definitions of quaternionic automorphic forms and the correspondence.

Let F be a totally real field of degree n over $\mathbf{Q}$. Algorithm I uses a definite quaternion algebra over F (ramified at all n infinite places). Algorithm II uses the Shimura curve associated to a quaternion algebra ramified at exactly $n-1$ infinite places. By default, the algorithm (and the quaternion order to be used) are selected automatically, but can also be specified by the user. The essential requirement is that the quaternion order can be ramified only at primes p for which the space is p-new. The required conditions are stated fully in the descriptions of the commands `HilbertCuspForms` and `NewSubspace`. It is worth noting that a different algorithm and order may be selected for a newspace than for the space containing it. In fact, there is more freedom for a new space; in particular, over $\mathbf{Q}$ neither algorithm can compute full spaces, but can compute new subspaces for nonsquare levels.

Algorithm II is implemented only for parallel weight 2 (weight $[2, 2, \ldots, 2]$). In the current implementation, Algorithm I is more optimized than Algorithm II, and therefore should be preferred in most cases. For spaces over odd degree fields with level not divisible by any small primes, Algorithm II may be preferable.

An exposition of both algorithms is given in [DV12], in addition to the references mentioned in the following two sections.

143.1.3 Algorithm I (Using Definite Quaternion Orders)

This algorithm, described in [Dem07] and [DD08], is an efficient formulation of the Brandt module approach to computing modular forms.

A key advantage of this algorithm is that the most expensive steps come in the precomputation phase (computations depending only on the quaternion algebra). This means that for a given number field, forms of many different levels and weights can be computed based on the same precomputation.

143.1.4 Algorithm II (Using Indefinite Quaternion Orders)

This algorithm, described in [GV11], is based on computing the homology of a Shimura curve (and therefore is closer to the usual modular symbols algorithm over **Q** than Algorithm I). Voight's algorithm ([Voi09]) for the fundamental domain of the action of the Fuchsian group on the upper half plane is applied. The algorithm simultaneously determines a fundamental domain and generators and relations for the fundamental group. From this a description of the homology is obtained, in such a way that the Hecke action on it can then be explicitly calculated.

143.1.5 Categories

The MAGMA category for spaces of Hilbert modular forms is `ModFrmHil`, while elements in these spaces have type `ModFrmHilElt`.

143.1.6 Verbose Output

To see some information printed during computation about what the program is doing, use `SetVerbose("ModFrmHil",n)`, where `n` is 0 (silent, by default), 1 (prints concise information), 2 or 3 (which may display bulky data).

143.2 Creation of Full Cuspidal Spaces

In the current implementation only cusp forms are supported. In future releases it will be possible to compute Eisenstein series, and create the full space of Hilbert modular forms of given weight and level.

```
HilbertCuspForms(F, N, k)
```

```
HilbertCuspForms(F, N)
```

`QuaternionOrder` ALGASSVORD *Default :*

This creates the space of Hilbert modular forms over the field F (a number field or the rationals) on $\Gamma_0(N)$ with weight k. Here the level N should be an ideal in the maximal order of F, and k should be a sequence of $\deg(\mathrm{F}/\mathbf{Q})$ integers, all at least 2 and all of the same parity. If not specified, the weight is taken to be parallel weight 2, ie. [2, 2, ..., 2].

Computations in the space will be done by realising it as a space of automorphic forms on an order in a suitable quaternion algebra (as explained in the introduction).

This choice is made automatically (in most cases), when it is needed, and is hidden from the user; however the user may specify the order to be used by providing the optional argument `QuaternionOrder`. The quaternion algebra may be definite (ramified at all infinite places of F) or indefinite (ramified at all infinite places except one); this determines which of the two algorithms will be used (see the introduction). Indefinite algebras may only be used for spaces of parallel weight 2. The algebra must be unramified at all finite primes (for full cuspidal spaces, although often a different algebra may be used to compute a `NewSubspace`). In the definite case the order must be maximal, while in the indefinite case it must be an Eichler order with discriminant equal to the level N.

Example H143E1

Here we create some spaces of modular forms over $\mathbf{Q}(\sqrt{85})$.

```
> _<x> := PolynomialRing(Rationals());
> F := NumberField(x^2-85);
> level := 1*Integers(F);
> H := HilbertCuspForms(F, level);
> H;
Cuspidal space of Hilbert modular forms over Number Field with defining polynomial
x^2 - 85 over the Rational Field
   Level = Ideal of norm 1 generated by ( [1, 0] )
   Weight = [ 2, 2 ]
```

This returns instantly because no nontrivial computations have been done yet. We can find out which quaternion order will be used internally to do computations.

```
> QO := QuaternionOrder(H);
> A := Algebra(QO);
> A;
Quaternion Algebra with base ring F
> A.1^2, A.2^2, A.3^2;
-1 -1 -1
> IsMaximal(QO);
true
```

So the order is a maximal order in Hamilton's quaternion algebra over F. We now define a space with level equal to one of the split primes dividing 3.

```
> level := Factorization(3*Integers(F))[1][1];
> Norm(level);
3
> weight := [3,5];
> H := HilbertCuspForms(F, level, weight);
```

If we wish, we may tell Magma to use the same quaternion order. (This would avoid some of the time-consuming computations being done twice.)

```
> H := HilbertCuspForms(F, level, weight : QuaternionOrder:=QO);
```

143.3 Caching Spaces of Modular Forms

There is a feature to store spaces that have been computed over a particular field. This typically saves a lot of time in many of the routines, because the algorithms use precomputations and data from existing spaces over the same field. However, these objects also occupy a lot of memory.

By default, no caching is done; the user must manually set it for each field. It is recommended to always do this when the field is created, and then clear the cache whenever memory usage becomes a concern.

Note: in earlier versions (prior to V2.20) caching was always switched on for Hilbert modular forms.

`SetStoreModularForms(F, v)`

If v is `true`, modular forms over the field F that are created afterwards will be cached. If v is `false`, the existing cache is made empty and caching is switched off.

`ClearStoredModularForms(F)`

The cache of modular forms over the field F is made empty. If caching was switched on for F, it remains switched on.

143.4 Basic Properties

`BaseField(M)`

The field on which the space M was defined.

`Weight(M)`

The weight of the space M.

`CentralCharacter(M)`

The central character of the weight representation defining the space M of Hilbert modular forms. This is of significance only for higher weight spaces (not parallel weight 2).

`Level(M)`

The level of the space M.

`DirichletCharacter(M)`

The nebentypus of the space M of Hilbert modular forms.

`IsCuspidal(M)`

This is `true` if the space M was created as (a subspace of) a space of cusp forms. In the current implementation this is always `true`.

`IsNew(M)`

This is true if the space M was created as (a subspace of) a new space, for instance using `NewSubspace` or `NewformDecomposition`, or if the space is known to satisfy `NewLevel(M) = Level(M)`.

`NewLevel(M)`

This returns the level at which M is known to be new. (See `NewSubspace`.)

`Dimension(M)`

`UseFormula` BoolElt *Default :* `true`

This computes the dimension of the space M of Hilbert modular forms.

The dimension is determined either by using "dimension formulae" or else by explicitly constructing the space. By default, formulae are used when available and when considered cheap to evaluate. The optional argument `UseFormula` can be set `true` or `false` to override the default.

Dimension formulae are implemented for spaces of parallel weight 2. These formulae are sums over certain cyclotomic extensions of the base field of F.

The results by formula are guaranteed only under GRH, since they may involve class numbers that are calculated conditionally. If at some later point, the space is explicitly computed, the dimension is will then be verified unconditionally.

`QuaternionOrder(M)`

This returns the quaternion order that is used internally to compute the space M of Hilbert modular forms.

`IsDefinite(M)`

This indicates which of the two algorithms is used to compute the space M of Hilbert modular forms. It is equivalent to `IsDefinite(Algebra(QuaternionOrder(M)))`. Note that calling this causes a `QuaternionOrder` for M to be chosen (if it has not been set already).

Example H143E2

We continue with the first example above.

```
> _<x> := PolynomialRing(Rationals());
> F := NumberField(x^2-85);
> OF := Integers(F);
> H := HilbertCuspForms(F, 1*OF);
> Norm(Level(H));
1
> Weight(H);
[ 2, 2 ]
> time Dimension(H);
6
Time: 2.580
```

```
> IsDefinite(H);
true
```

This indicates that Algorithm I (described in the introduction) was used to compute the dimension.

```
> level := Factorization(3*OF)[1][1];
> H3 :=  HilbertCuspForms(F, level);
> Level(H3);
Prime Ideal of OF
Two element generators:
    [3, 0]
    [2, 2]
> time Dimension(H3);
14
Time: 2.370
> H3 :=  HilbertCuspForms(F, level : QuaternionOrder:=QuaternionOrder(H) );
> time Dimension(H3);
14
Time: 0.270
```

It is much faster when we recycle the same quaternion order used for the space of level 1. This is because the harder computations involved depend only on the quaternion order, not on the level.

143.5 Elements

The current implementation does not provide functionality for manipulating elements in spaces of Hilbert modular forms. Note that, unlike classical modular forms in MAGMA, Hilbert modular forms are allowed to be defined over extensions of the base field of their parent space.

`Parent(f)`

The space of Hilbert modular forms containing f.

`BaseField(f)`

The field over which the Hilbert modular form f is defined. This is either equal to, or an extension of, the base field of `Parent(f)`.

143.6 Operators

Operators on spaces of Hilbert modular forms are returned as matrices with respect to a basis of M. For spaces of parallel weight 2, operators are matrices over $\mathbf{Q}$, and the basis of M is permanently fixed. For all other weights, two finite extensions of $\mathbf{Q}$ may arise:

(i) The field that is used for the raw computations; operators are originally computed as matrices over this field.

(ii) The minimal field F for which there exists a basis of M such that operators are matrices with entries in F. For all parallel weights, this minimal field is $\mathbf{Q}$. We refer to such a basis as a "rational basis" of M.

Changing the basis can be expensive for large spaces, so for some spaces a "rational basis" is not computed by default. In these cases, Hecke operators are returned as matrices over the "raw" field, with respect to a basis which remains fixed until `SetRationalBasis(M)` is invoked by the user. When that occurs, the basis of the space is changed to a "rational basis" which is then permanently fixed; as a result, the operators also change (they are now given as matrices with respect to the new basis).

A space M is guaranteed to have a "rational basis" (which is permanently fixed) in any of the following circumstances:

(i) M has parallel weight,

(ii) M was constructed using `NewSubspace` (unless `RationalBasis` was set to `false`), or was constructed using `NewformDecomposition`,

(iii) `SetRationalBasis(M)` has been invoked.

`HeckeOperator(M, P)`

This returns a matrix representing the Hecke operator T_P on the space M of Hilbert modular forms.

Example H143E3

```
> R<x> := PolynomialRing(IntegerRing());
> F<a> := NumberField(x^2-2);  OF := Integers(F);
> M3 := HilbertCuspForms(F, 3*OF, [2,4]);
> Dimension(M3);
1
> P2 := Factorization(2*OF)[1][1];
> Norm(P2);
2
> HeckeOperator(M3, P2);
[2]
```

Since the space has dimension 1, it consists of a single eigenform, whose eigenvalues can be read from the Hecke matrices. We now consider the space of level 5, which has dimension 3. The Hecke matrices are given with respect to the basis used to compute the space, over the extension K displayed here.

```
> M5 := HilbertCuspForms(F, 5*OF, [2,4] : QuaternionOrder:=QuaternionOrder(M3) );
```

```
> Dimension(M5);
3
> T2 := HeckeOperator(M5, P2);
> K<b> := BaseRing(T2);
> K;
Number Field with defining polynomial $.1^2 + 1 over F
> T2;
[       0    8*a*b -2*b - 2]
[-1/2*a*b        2        0]
[   b - 1        0        0]
```

`AtkinLehnerOperator(M, P)`

This returns a matrix representing the Atkin-Lehner operator w_P on the space M of Hilbert modular forms.

`DegeneracyOperator(M, P, Q)`

This computes degeneracy maps in the "downward" direction as maps from M to itself. Here M is a space of Hilbert modular forms of level N, P is a prime dividing N, and Q either equals P or the unit ideal (1). The function returns a matrix representing a map from M to M whose image equals a copy of the space of level N/P.

When $Q = (1)$, this is double coset operator defined by cosets of an element with determinant 1 (which can be seen as a "norm" map); when $Q = P$, it is the double coset operator defined by cosets of an element with determinant Norm(P).

`DeleteHeckePrecomputation(O)`

`DeleteHeckePrecomputation(O, P)`

These procedures delete data obtained during the precomputation phase of the "definite" algorithm. This data is used in the computation of Hecke operators (and other operators) for given primes, and the same data can be re-used for all spaces that are computed with the some quaternion order O. (Often this is the same for spaces of all weights and levels over a particular number field.) Since the data is the most expensive part of the Hecke computation, it is stored by default. However it is very expensive in memory; these procedures allow the user to reclaim this memory when necessary.

143.7 Creation of Subspaces

`NewSubspace(M)`

`NewSubspace(M, I)`

`QuaternionOrder` AlgAssVOrd *Default :*

Given a cuspidal space M of Hilbert modular forms of level N, and an ideal I dividing N, this constructs the subspace of M consisting of forms that are new at the ideal I (or that are new at N, if I is not given). More precisely, this is the complement of the space generated by all images under degeneracy maps of spaces of level N/P for primes P dividing I. In the current implementation, I must be squarefree and coprime to N/I.

The `NewSubspace` of M is not necessarily represented as an explicit subspace of M: doing so is not always possible, or may not be optimal. (These choices are internal: the difference is not visible in the output.) In many cases new subspace is obtained as an explicit subspace by computing degeneracy maps. In other case it is computed independently of M, using a quaternion order which is chosen automatically. When the optional argument `QuaternionOrder` is specified, this will always be used (overriding other considerations). The allowable orders are similar to those for full cuspidal spaces (see `HilbertCuspForms` above), with the difference that here the quaternion algebra is allowed to be ramified at finite primes dividing I. When the algebra is indefinite, the finite primes where it is ramified must be precisely those dividing I.

In the non-parallel weight case, work may be saved by setting the optional argument `RationalBasis` to `false`. This defers the computation of a "rational basis" of the new space (this is explained below). By default, a rational basis is computed for `NewSubspace(M)`, and (equivalently) `NewSubspace(M)` where I equals `Level(M)`. If it is defered, one may later set a "rational basis" for the new space using `SetRationalBasis`.

`SetRationalBasis(M)`

This is a procedure which changes the basis of M. A full explanation is given in the first part of Section 143.6.

If the basis of M is already known to be a "rational basis", then nothing is done; the basis of M is not modified. In particular, this has no effect on spaces of parallel weight 2.

After this has been invoked, the basis of M is never modified again.

Example H143E4

We investigate new forms and old forms with level dividing 3 over $\mathbf{Q}(\sqrt{10})$.

```
> _<x> := PolynomialRing(Rationals());
> F := NumberField(x^2-10);
> OF := Integers(F);
> primes := [tup[1] : tup in Factorization(3*OF)];
```

```
> #primes;
2
> M1 := HilbertCuspForms(F, 1*OF);
> Dimension(M1);
2
> M3 := HilbertCuspForms(F, primes[1]);
> Dimension(M3);
4
```

Hence $M3$ must contain only oldforms (since there are two degeneracy maps $M1 \to M3$, whose images must be linearly independent). We verify this now.

```
> Dimension(NewSubspace(M3));
0
```

Next we consider level (3).

```
> M9 := HilbertCuspForms(F, 3*OF);
> Dimension(M9);
18
> Dimension(NewSubspace(M9));
10
```

The remaining dimension 8 all comes from level 1 (via the four images of $M1$).

```
> Dimension(NewSubspace(M9, primes[1]));
10
> Dimension(NewSubspace(M9, primes[2]));
10
```

The dimensions indicate that the four degeneracy maps $M1 \to M9$ have independent images.

Example H143E5

This illustrates that new subspaces may be computed independently, not using the same algorithm as for the full space. We compute forms over the real subfield of $\mathbf{Q}(\zeta_7)$, which has degree 3.

```
> _<x> := PolynomialRing(Rationals());
> _<zeta7> := CyclotomicField(7);
> F<a> := NumberField(MinimalPolynomial(zeta7 + 1/zeta7));
> F;
Number Field with defining polynomial x^3 + x^2 - 2*x - 1 over the Rational Field
```

We consider forms of level 3 (which generates a prime ideal in F).

```
> M := HilbertCuspForms(F, 3*Integers(F));
> M;
Cuspidal space of Hilbert modular forms over
Number Field with defining polynomial x^3 + x^2 - 2*x - 1 over the Rational Field
   Level = Ideal of norm 27 generated by ( [3, 0, 0] )
   Weight = [ 2, 2, 2 ]
> Mnew := NewSubspace(M);
> Dimension(M);
```

```
1
> Dimension(Mnew);
1
```

So in fact M equals its new subspace, and consists of just one newform. However, different algorithms have been chosen to compute them, as indicated below. (Algorithm I is chosen automatically for the new subspace because it is much faster. It cannot be used for the full space, since over an odd degree field there is no quaternion algebra ramified at precisely the infinite places.)

```
> IsDefinite(M);
false
> IsDefinite(Mnew);
true
```

We now compute some eigenvalues of the newform generating the space, using both algorithms. (The computations for M_{new} are entirely independent from those for M.)

```
> primes := PrimesUpTo(20,F);
> [Norm(P) : P in primes];
[ 7, 8, 13, 13, 13 ]
> time for P in primes do HeckeOperator(Mnew,P); end for;
[-5]
[-4]
[1]
[1]
[1]
Time: 0.810
> time HeckeOperator(M, primes[1]);
[-5]
Time: 38.800
```

143.8 Eigenspace Decomposition and Eigenforms

`HeckeEigenvalueBound(M, P)`

Returns a bound on the absolute value of the Hecke eigenvalue at the prime P which must hold for all newforms in the space M of Hilbert modular forms.

`NewformDecomposition(M)`

Given a space M of Hilbert modular forms which was created as a `NewSubspace`, this decomposes M into subspaces that are irreducible modules under the Hecke action.

`NewformsOfDegree1(M)`

This constructs the list of new eigenforms in M that have rational eigenvalues, i.e. corresponding to the 1-dimensional components in the `NewformDecomposition`. The space M is not required to be a new space. The algorithm avoids constructing the new subspace of M, and makes use of bounds on the eigenvalues.

`Eigenform(M)`

This constructs an eigenform contained in the space M of Hilbert modular forms (which should be an irreducible module under the Hecke action, for instance a space obtained using `NewformDecomposition`).

`Eigenforms(M)`

This is a list containing an eigenform from each space in `NewformDecomposition(M)`.

`HeckeEigenvalueField(M)`

Given a space M constructed using `NewformDecomposition`, this returns the number field over which the `Eigenform` of M is defined.

`HeckeEigenvalue(f, P)`

The computes the eigenvalue of the Hecke operator T_P acting on the eigenform f (which should be a Hilbert modular form constructed using `Eigenform`).

Example H143E6

We compute the newforms corresponding to elliptic curves over $\mathbf{Q}(\sqrt{2})$ of conductor 11, and find there is only the one coming from $\mathbf{Q}$.

```
> R<x> := PolynomialRing(IntegerRing());
> F := NumberField(x^2-2);  OF := Integers(F);
> M := HilbertCuspForms(F, 11*OF);
> Dimension(M);
6
> time decomp := NewformDecomposition(NewSubspace(M)); decomp;
Time: 11.130
[*
    New cuspidal space of Hilbert modular forms of dimension 1 over
    Number Field with defining polynomial x^2 - 2 over the Rational Field
       Level = Ideal of norm 121 generated by ( [11, 0] )
       Weight = [ 2, 2 ],
    New cuspidal space of Hilbert modular forms of dimension 5 over
    Number Field with defining polynomial x^2 - 2 over the Rational Field
       Level = Ideal of norm 121 generated by ( [11, 0] )
       Weight = [ 2, 2 ]
*]
```

We look at the first few eigenvalues of the 1-dimension piece (at split primes).

```
> f := Eigenform(decomp[1]);
```

```
> primes := [P : P in PrimesUpTo(40,F) | IsOdd(Norm(P)) and IsPrime(Norm(P))];
> for P in primes do
>   Norm(P),  HeckeEigenvalue(f,P);
> end for;
7 -2
7 -2
17 -2
17 -2
23 -1
23 -1
31 7
31 7
```

Happily, they agree with the eigenvalues of the elliptic cusp form of conductor 11 over $\mathbf{Q}$:

```
> fQ := Newforms(CuspForms(11))[1][1];
> for P in primes do
>   p := Norm(P);
>   p,  Coefficient(fQ, p);
> end for;
7 -2
7 -2
17 -2
17 -2
23 -1
23 -1
31 7
31 7
```

The 5-dimensional piece conjecturally corresponds to an abelian variety over F of dimension 5, which would be absolutely irreducible and have real multiplication by the following field.

```
> K := HeckeEigenvalueField(decomp[2]);
> K;
Number Field with defining polynomial $.1^5 - 8*$.1^3 + 10*$.1 + 4 over F
> IsTotallyReal(K);
true
```

143.9 Further Examples

Example H143E7

```
> R<x> := PolynomialRing(IntegerRing());
> F := NumberField(x^2-15);  OF := Integers(F);
> M := HilbertCuspForms(F, 1*OF, [2,4]);
> M;
Cuspidal space of Hilbert modular forms over Number Field with defining polynomial
x^2 - 15 over the Rational Field
   Level = Ideal of norm 1 generated by ( [1, 0] )
   Weight = [ 2, 4 ]
> Dimension(M);
2
> T2 := HeckeOperator(M, Factorization(2*OF)[1][1] );
> BaseRing(T2);
Number Field with defining polynomial $.1^2 + 1 over F
```

So the basis used to compute M is over this extension of F. We now replace this with a F-rational basis (this is implemented for new spaces, in particular spaces of level 1).

```
> IsNew(M);
true
> SetRationalBasis(M);
> for P in PrimesUpTo(7,F) do
>    Norm(P), HeckeOperator(M,P);
> end for;
2
[  0   1]
[-20   0]
3
[0 0]
[0 0]
5
[  0   4]
[-80   0]
7
[0 0]
[0 0]
7
[0 0]
[0 0]
> #NewformDecomposition(M);
1
```

In general the Hecke matrices would be over F, however for this space there is a basis where they have entries in $\mathbf{Z}$. The program was able to discover this because the basis is chosen by putting one of the matrices in rational canonical form.

Example H143E8

It is possible to use this package to compute classical modular forms (although this will usually be much slower). Here we compute the newform of level 14 three times independently. First we wish to use Algorithm 1, so we choose the quaternion algebra over **Q** ramified at 2 and infinity. (Note: Algorithm I is not implemented over `Rationals()`, so we must work over a number field isomorphic to **Q** instead!)

```
> QQ := RationalsAsNumberField();
> ZZ := Integers(QQ);
> M := HilbertCuspForms(QQ, 14*ZZ);
> A := QuaternionAlgebra(2*ZZ, InfinitePlaces(QQ) : Optimized);
> M14 := NewSubspace(M : QuaternionOrder:=MaximalOrder(A) );
> Dimension(M14);
1
> f := Eigenform(NewformDecomposition(M14)[1]);
> primes := PrimesUpTo(50);
> time eigenvalues1:= [ <p, HeckeEigenvalue(f,p*ZZ)> : p in primes ];
Time: 0.220
> eigenvalues1;
[ <2, -1>, <3, -2>, <5, 0>, <7, 1>, <11, 0>, <13, -4>, <17, 6>, <19, 2>, <23, 0>,
  <29, -6>, <31, -4>, <37, 2>, <41, 6>, <43, 8>, <47, -12> ]
```

Now we use Algorithm 2, choosing the indefinite quaternion algebra over **Q** ramified at 2 and 7.

```
> Q := Rationals();
> M := HilbertCuspForms(Q, 14);
> A := QuaternionAlgebra(14 : Al:="Smallest" );
> A.1^2, A.2^2;
7, -2
> M14 := NewSubspace(M : QuaternionOrder:=MaximalOrder(A) );
> IsDefinite(M14); // Not definite means Algorithm 2
false
> Dimension(M14);
1
> f := Eigenform(NewformDecomposition(M14)[1]);
> time eigenvalues2 := [ <p, HeckeEigenvalue(f,p)> : p in primes |
>                                                     GCD(p,14) eq 1];
Time: 2.750
> eigenvalues2;
[ <3, -2>, <5, 0>, <11, 0>, <13, -4>, <17, 6>, <19, 2>, <23, 0>,
  <29, -6>, <31, -4>, <37, 2>, <41, 6>, <43, 8>, <47, -12> ]
```

Finally we check both results agree with the standard modular forms package.

```
> M14 := CuspForms(14);
> time eigenvalues := [ <p, HeckeOperator(M14,p)[1,1]> : p in primes ];
Time: 0.160
> assert eigenvalues1 eq eigenvalues;
> assert eigenvalues2 subset eigenvalues;
```

143.10 Bibliography

[**DD08**] L. Dembele and S. Donnelly. Computing Hilbert Modular Forms Over Fields With Nontrivial Class Group. In S. Pauli F. Hess and M.Pohst, editors, *ANTS VIII*, volume 5011 of *LNCS*. Springer-Verlag, 2008.

[**Dem07**] L. Dembélé. Quaternionic Manin symbols, Brandt matrices, and Hilbert modular forms. *Math. Comp.*, 76(258):1039–1057 (electronic), 2007.

[**DV12**] L. Dembélé and J. Voight. Explicit methods for Hilbert modular forms. *to appear, Elliptic curves, Hilbert modular forms and Galois deformations*, 2012.

[**Fre90**] E. Freitag. *Hilbert modular forms*. Springer-Verlag, Berlin, 1990.

[**Gar90**] Paul B. Garrett. *Holomorphic Hilbert modular forms*. The Wadsworth & Brooks/Cole Mathematics Series. Wadsworth & Brooks/Cole Advanced Books & Software, Pacific Grove, CA, 1990.

[**GV11**] M. Greenberg and J. Voight. Computing systems of eigenvalues associated to Hilbert modular forms. *Math. Comp.*, 80:1071–1092, 2011.

[**Hid06**] Haruzo Hida. *Hilbert modular forms and Iwasawa theory*. Oxford Mathematical Monographs. The Clarendon Press Oxford University Press, Oxford, 2006.

[**Voi09**] J. Voight. Computing fundamental domains for cofinite Fuchsian groups. *J. Théor. Nombres Bordeaux*, 21(2):469–491, 2009.

144 MODULAR FORMS OVER IMAGINARY QUADRATIC FIELDS

Chapter 144
MODULAR FORMS OVER IMAGINARY QUADRATIC FIELDS

144.1 Introduction

This package deals with cuspidal spaces of weight 2 modular forms on $\Gamma_0(N)$, over any imaginary quadratic field. In the current version, one can compute Hecke operators for principal ideals on these spaces, and determine the newforms. The package will be developed further in the future.

We will refer to modular forms over imaginary quadratic fields as *Bianchi modular forms*. These are defined similarly to classical modular forms, with the modular group $SL(2, \mathbf{Z})$ is replaced by $SL(2, O_F)$, where O_F is the ring of integers in an imaginary quadratic field F. This group acts on $\mathbf{H_3}$, 3-dimensional hyperbolic space, and Bianchi modular forms (of weight $k \geq 2$) are functions on $\mathbf{H_3}$ that satisfy a natural automorphy relation under this action.

For an ideal N of O_F, the congruence subgroup $\Gamma_0(N)$ of $GL(2, O_F)$ is defined as the subgroup of matrices that are upper triangular modulo N. The space of *Bianchi modular forms on F of level N* is defined as the space of functions satisfying the automorphy relation on the subgroup $\Gamma_0(N)$.

For precise definitions see [EGM98].

There exist several previous implementations of weight 2 Bianchi modular forms, each for specific fields F. These projects are described in [Cre84] and references cited there for Euclidean fields, [Whi90] for fields of class number one, [Byg99] for $\mathbf{Q}(\sqrt{-5})$ and [Lin05] for $\mathbf{Q}(\sqrt{-23})$ and $\mathbf{Q}(\sqrt{-31})$.

144.1.1 Algorithms

A theorem of Franke states that for algebraic group $\mathbf{G}$ defined over $\mathbf{Q}$ and arithmetic subgroup Γ,

$$H^*(\Gamma; E) \simeq H^*(g, K; A(\Gamma, G) \otimes E)$$

where $A(\Gamma, G)$ denotes the space of automorphic forms. Thus we can think of the cohomology $H^*(\Gamma; E)$ as a concrete realization of certain automorphic forms.

Ash, Gunnells, and Lee-Szczarba [LS78, Ash94, Gun00] define a homology complex $S^*(\Gamma)$ known as the *sharbly complex*. A theorem of Borel-Serre [BS73] gives

$$H^{\nu-k}(\Gamma; \mathbf{C}) \simeq H_k(S_*(\Gamma)),$$

where $\nu = \mathrm{vcd}(\Gamma)$ is the *virtual cohomological dimension* of Γ.

There is a natural Hecke action on this complex which agrees with the Hecke action on the automorphic forms.

The space of positive definite binary Hermitian forms over F form an open cone in a real vector space. There is a natural decomposition of this cone into polyhedral cones corresponding to the facets of the Voronoi polyhedron [Gun99, Koe60, Ash77]. These facets are in 1-1 correspondence with perfect forms over F. The polyhedral cones give rise to ideal polytopes in $\mathbf{H_3}$, 3-dimensional hyperbolic space.

The structure of the polytopes allows us to find a finite (modulo Γ) spanning set for the sharbly complex. This is the analogue of unimodular symbols in the classical case. The modular symbol algorithm, for describing the action of Hecke operators, can now be replaced by a 0-sharbly reduction algorithm. An advantage of this is, compared with a straightforward generalization of the usual modular symbols algorithm, is that the number field does not need to be Euclidean.

Given an imaginary quadratic field F, we compute the structure of the Voronoi polyhedron by computing a complete set of $\mathrm{GL}_2(\mathcal{O})$-class representatives of perfect forms. Given a level $n \subseteq \mathcal{O}$ defining the level of the congruence subgroup, the polyhedron is used to compute $H^2(\Gamma_0(n))$. Given a prime ideal $p \subset \mathcal{O}$, the 0-sharbly reduction algorithm is used to compute the action of the Hecke operator T_p on $H^2(\Gamma_0(n))$.

The algorithm described in [Gun99] is an efficient algorithm for computing the Voronoi polyhedron and provides a replacement for the modular symbol algorithm for computing the action of Hecke operators.

One advantage of this algorithm is that the most expensive steps come in the precomputation phase (computations depending only on the imaginary quadratic field). This means that for a given field, forms of many different levels can be computed based on the same precomputation.

The cohomology, regarded as a module for the algebra generated by the Hecke operators T_p, decomposes into an Eisenstein piece and a cuspidal piece. The cuspidal piece is the same as the cuspidal subspace of modular forms (regarding both as Hecke modules). The Eisenstein part is recognisable by looking at Hecke eigenvalues of the eigenforms contained in it. By this means, we strip away the Eisenstein part; the space returned is the cuspidal subspace only.

144.1.2 Categories

Spaces of Bianchi modular forms in MAGMA are objects of type `ModFrmBianchi`.

This type shares many features with the type `ModFrmHil` of Hilbert modular forms. Many of the functions for Hilbert modular forms (see Chapter 143) can also be applied to Bianchi spaces, such as `NewSubspace`, `NewformDecomposition`, etc.

Technically, `ModFrmBianchi` is implemented as a subtype of `ModFrmHil`. Some intrinsics may not be displayed separately for the two types in MAGMA, for instance when one enters `NewSubspace;` at the MAGMA prompt.

144.1.3 Verbose Output

To see some information printed during computation about what the program is doing, use `SetVerbose("Bianchi",n)`, where `n` is 0 (silent, by default), 1 (for concise information), 2, 3 or 4 (which may display bulky data).

144.2 Creation

In the current implementation only cusp forms are supported.

`BianchiCuspForms(F, N)`

`VorData` REC *Default :*

This creates the cuspidal subspace of the space of Bianchi modular forms over the field F (an imaginary quadratic field) on $\Gamma_0(N)$ with weight 2. The level N should be an ideal in the maximal order of F.

The optional argument `VorData` may be set equal to `VoronoiData(M)` where `M` is a previously computed space of forms over the same field. This avoids repeating the time-consuming precomputations that depend only on the field.

144.3 Attributes

`BaseField(M)`

The field on which the space M of Bianchi modular forms was defined.

`Level(M)`

The level of the space M.

`Dimension(M)`

The dimension of the space M. Dimension formulas are not available, so the dimension is computed by explicit construction of the space.

`VoronoiData(M)`

This returns a record containing technical data that is computed in the precomputation phase of the algorithm. This depends only on the base field of M, and the data can be reused when computing spaces of different levels over the same field.

Example H144E1

We create spaces of modular forms over $\mathbf{Q}(\sqrt{-14})$ for various levels.

```
> _<x> := PolynomialRing(Rationals());
> F := NumberField(x^2 + 14);
> OF := Integers(F);
> level := 1*OF;
> M := BianchiCuspForms(F, level);
> M;
Cuspidal space of Bianchi modular forms over
    Number Field with defining polynomial x^2 + 14 over the Rational Field
    Level = Ideal of norm 1 generated by ( [1, 0] )
    Weight = 2
>  time Dimension(M);
0
Time: 0.050
```

We now define a space with level equal to the square of one of the split primes dividing 3.

```
> level := (Factorization(3*OF)[1][1])^2;
> Norm(level);
9
> time M9 := BianchiCuspForms(F, level);
Time: 1.370
> time Dimension(M9);
1
Time: 0.370
```

When defining this space, we may tell Magma to use the same Voronoi data, to avoid repeating this expensive precomputation:

```
> time M9 := BianchiCuspForms(F, level : VorData := VoronoiData(M) );
Time: 0.000
> time Dimension(M9);
1
Time: 0.370
> M9;
Cuspidal space of Bianchi modular forms over
    Number Field with defining polynomial z^2 + 14 over the Rational Field
    Level = Ideal of norm 9 generated by ( [9, 0], [5, 2] )
    Weight = 2
    Dimension 1
```

144.4 Hecke Operators

The computations are done essentially on the cohomology of $\Gamma\backslash\mathbf{H_3}$, and so the for a non-principal ideal P, the Hecke operator T_P does not act on this space, but sometimes the Hecke action can be deduced from the action of principal Hecke operators. See for instance [Lin05]. Specifically, suppose p is a prime ideal that is prime to the level. There exists an ideal a, prime to p and the level, such that a^2p is principal. Then the composition $T_{a,a}T_p$ acts on the cohomology.

`HeckeOperator(M, P)`

This returns a matrix representing a certain Hecke action T on the space M of Bianchi modular forms, with respect to the fixed basis of M. The ideal P must be principal (but not necessarily prime) or prime and a square in the class group. The ideal must also be coprime to the level (except if the space is dimension 0). If P is principal, then T is the Hecke operator T_P. When P is prime and a square in the class group, T is the composition $T_{a,a}T_P$ for a suitably chosen ideal a.

Example H144E2

We continue the previous example.

```
> _<x> := PolynomialRing(Rationals());
>  F := NumberField(x^2+14);
>  OF := Integers(F);
>  level := (Factorization(3*OF)[1][1])^2;
>  M9 := BianchiCuspForms(F, level);
>  P:=Factorization(23*OF);
>  P[1,1];
Prime Ideal of OF
Two element generators:
    [23, 0]
    [3, 1]
>  HeckeOperator(M9, P[1,1]);
[8]
>  P[2,1];
Prime Ideal of OF
Two element generators:
    [23, 0]
    [20, 1]
>  HeckeOperator(M9, P[2,1]);
[-8]
>  HeckeOperator(M9, 2*OF);
[1]
```

Since this cuspidal space has dimension 1, it consists of a single eigenform, whose eigenvalues can be read from the Hecke matrices.

144.5 New Spaces and Newforms

The functions described here are very similar to those for Hilbert modular forms (see Chapter 143). Some further related intrinsics described there, such as `HeckeEigenvalue`, can also be applied to Bianchi modular forms.

In these computations it is typically very important to switch on the caching feature (see Section 143.3). This is because spaces of lower levels tend to be used repeatedly.

`NewSubspace(M)`

This computes the new subspace of a given space of Bianchi modular forms. The algorithm identifies the old spaces in M by comparing the Hecke action on M and spaces of lower levels. The algorithm assumes that the dimensions of old spaces are as expected. This has been extensively tested.

`NewformDecomposition(M)`

This returns a list of the Hecke-irreducible newform spaces in M, which is required to be a new space. The algorithm is the same as for Hilbert modular forms.

`Eigenform(M)`

This constructs the newform associated to a Hecke-irreducible space M obtained from `NewformDecomposition`.

144.6 Bibliography

[**Ash77**] Avner Ash. Deformation retracts with lowest possible dimension of arithmetic quotients of self-adjoint homogeneous cones. *Math. Ann.*, 225(1):69–76, 1977.

[**Ash94**] Avner Ash. Unstable cohomology of $\mathrm{SL}(n, \mathcal{O})$. *J. Algebra*, 167(2):330–342, 1994.

[**BS73**] A. Borel and J.-P. Serre. Corners and arithmetic groups. *Comment. Math. Helv.*, 48:436–491, 1973. Avec un appendice: Arrondissement des variétés à coins, par A. Douady et L. Hérault.

[**Byg99**] Jeremy Bygott. *Modular forms and modular symbols over imaginary quadratic fields*. PhD thesis, University of Exeter, 1999.

[**Cre84**] J. E. Cremona. Hyperbolic tessellations, modular symbols, and elliptic curves over complex quadratic fields. *Compositio Math.*, 51(3):275–324, 1984.

[**EGM98**] J. Elstrodt, F. Grunewald, and J. Mennicke. *Groups acting on hyperbolic space*. Springer Monographs in Mathematics. Springer-Verlag, Berlin, 1998. Harmonic analysis and number theory.

[**Gun99**] Paul E. Gunnells. Modular symbols for $\mathbf{Q}$-rank one groups and Voronoi reduction. *J. Number Theory*, 75(2):198–219, 1999.

[**Gun00**] Paul E. Gunnells. Computing Hecke eigenvalues below the cohomological dimension. *Experiment. Math.*, 9(3):351–367, 2000.

[**Koe60**] Max Koecher. Beiträge zu einer Reduktionstheorie in Positivitätsbereichen. I. *Math. Ann.*, 141:384–432, 1960.

[**Lin05**] Mark Lingham. *Modular forms and elliptic curves over imaginary quadratic fields*. PhD thesis, University of Nottingham, 2005.
URL:http://etheses.nottingham.ac.uk/138/.

[**LS78**] Ronnie Lee and R. H. Szczarba. On the torsion in $K_4(\mathbf{Z})$ and $K_5(\mathbf{Z})$. *Duke Math. J.*, 45(1):101–129, 1978.

[**Whi90**] Elise Whitley. *Modular symbols and elliptic curves over imaginary quadratic fields*. PhD thesis, University of Exeter, 1990.

145 ADMISSIBLE REPRESENTATIONS OF $\mathrm{GL}_2(\mathbf{Q}_p)$

Chapter 145
ADMISSIBLE REPRESENTATIONS OF $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$

145.1 Introduction

This package enables one to do the following. Starting with a cuspidal newform, one may define the local component at p of the associated automorphic representation, and determine its key properties. Furthermore, via the local Langlands correspondence, there exists a related Galois representation on the absolute Galois group of $\mathbf{Q}_p$. One may compute (the restriction to inertia of) that Galois representation.

The algorithms implemented here are described in [LW].

145.1.1 Motivation

Let F be a local non-archimedean field and let G be a reductive group over F. The representation theory of G is a rich subject in its own right but also has fascinating (and often conjectural) connections with the representation theory of the absolute Galois group of F. This package deals with admissible irreducible representations in the case that G is the group GL_2 and F is the p-adic field $\mathbf{Q}_p$. Such objects correspond canonically to two-dimensional representations of the absolute Galois group of F of a certain sort.

There are applications to the study of local properties of (global) Galois representations arising from modular forms. Namely, if f is a cuspidal newform, then there is associated to f a family of ℓ-adic Galois representations $\rho_f : \mathrm{Gal}(\overline{\mathbf{Q}}/\mathbf{Q}) \to \mathrm{GL}_2(\overline{\mathbf{Q}}_\ell)$. For example, for a prime p different from ℓ, one may determine the restriction of ρ_f to the decomposition group at p.

145.1.2 Definitions

Here we introduce the category of admissible irreducible representations of GL_2 over a non-archimedean field, which for our purposes will be $\mathbf{Q}_p$. The first systematic study of admissible representations is [JL70]. For an accessible introduction, see [BH06].

Let G be the locally compact group $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$. An *admissible* representation of G on a complex vector space V is a homomorphism $\pi : G \to \mathrm{AutV}$ which satisfies the properties:

(i) every vector $v \in V$ is fixed by a compact open subgroup of G, and

(ii) for every compact open subgroup $K \subset G$, V^K is finite-dimensional.

The center of G is $\mathbf{Q}_p^\times$. If π is an irreducible admissible representation of G, then it has a unique *central character* $\varepsilon : \mathbf{Q}_p^\times \to \mathbf{C}^\times$ such that $\pi(g)$ acts as the scalar $\varepsilon(g)$ for all $g \in \mathbf{Q}_p^\times$. The *conductor* of an irreducible admissible representation π is a measure of how small a compact open subgroup $K \subset G$ must be before one sees nonzero K-invariant vectors; see [Cas73]. Consider the filtration $K_0(p^n)$ of subgroups of G, where $K_0(p^n)$ is the

subgroup of matrices $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{GL}_2(\mathbf{Z}_\mathrm{p})$ with $c \equiv 0 \pmod{p^n}$. If π admits a nonzero vector fixed by $K_0(1) = \mathrm{GL}_2(\mathbf{Z}_\mathrm{p})$, then π is called *spherical* or *unramified principal series* and has conductor 1. If π admits a nonzero vector v for which $\pi\left(\begin{pmatrix} a & b \\ c & d \end{pmatrix}\right) v = \varepsilon(a)v$ for all $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in K_0(p^n)$, and $n \geq 1$ is minimal for this condition, then the conductor of π is p^n. (Note the similarity with the convention used to define the level of a modular form for $\Gamma_0(N)$.) In both cases the vector v so described is unique up to scaling; see [Cas73]. We shall call v a *new vector* for π.

If χ is a character of $\mathbf{Q}_p^\times$ then let $\pi \otimes \chi$ be the representation $g \mapsto \chi(g)\pi(g)$; such a representation is a *twist* of χ. If π has minimal conductor among all its twists $\pi \otimes \chi$, then π is called *minimal*.

Admissible representations are generally infinite-dimensional, but we will nonetheless be able to present them using MAGMA infrastructure for representations of finite groups and Dirichlet characters.

145.1.3 The Principal Series

We can directly construct a large class of admissible representations of G. Let χ_1 and χ_2 be two characters of $\mathbf{Q}_p^*$. Let $B \subset G$ be the Borel subgroup of upper triangular matrices. Then $\begin{pmatrix} a & b \\ 0 & d \end{pmatrix} \mapsto |a/d|^{-1/2}\chi_1(a)\chi_2(d)$ is a character χ of B. An admissible representation π is a *principal series* representation if it is a composition factor of the induced representation $\pi(\chi_1, \chi_2) := \mathrm{Ind}_\mathrm{B}^\mathrm{G}\chi$. This induced representation is already irreducible unless $\chi_1\chi_2^{-1}$ equals $|.|^{\pm 1}$, in which case it has length two, with one 1-dimensional and one infinite-dimensional composition factor. For instance, $\mathrm{Ind}_\mathrm{B}^\mathrm{G}1$ has a trivial 1-dimensional submodule and an irreducible infinite-dimensional quotient St_G, the *Steinberg* representation. The unramified principal series representations are either 1-dimensional, in which case they factor through the determinant map, or else they take the form $\pi(\chi_1, \chi_2)$, where χ_1 and χ_2 are unramified characters of $\mathbf{Q}_p^\times$ (meaning they are trivial on $\mathbf{Z}_p^\times$).

The central character of $\pi(\chi_1, \chi_2)$ is $\chi_1\chi_2$, and its conductor is the product of the conductors of the χ_i. Note that $\pi(\chi_1, \chi_2)$ is minimal if and only if one of the characters χ_1, χ_2 is unramified. The Steinberg representation St_G has trivial central character and conductor p.

145.1.4 Supercuspidal Representations

For the purposes of this package, an admissible irreducible representation is *supercuspidal* if it does not belong to the principal series. A supercuspidal representation π has conductor p^c, where $c \geq 2$. There is a convenient, if technical, classification of supercuspidal representations of G. Let π be supercuspidal. By [BH06], Ch. 15, there is a representation Ξ of an open and compact-mod-center subgroup $K \subset G$ for which $\pi = \mathrm{Ind}_\mathrm{K}^\mathrm{G}\Xi$. If c is even, then we may take K to be $\mathbf{Q}_p^\times\mathrm{GL}_2(\mathbf{Z}_\mathrm{p})$, and if c is odd, we may take K to be the

normalizer of the Iwahori subgroup $K_0(p) = \begin{pmatrix} \mathbf{Z}_p^\times & \mathbf{Z}_p \\ p\mathbf{Z}_p & \mathbf{Z}_p^\times \end{pmatrix}$ in G. We call the pair (K, Ξ) a *cuspidal inducing datum.*

145.1.5 The Local Langlands Correspondence

The Local Langlands Correspondence is a canonical bijection $\pi \mapsto \sigma(\pi)$ between irreducible admissible representations of G and local 2-dimensional representations of the absolute Galois group of $\mathbf{Q}_p$ of a certain sort. (Note to purists: the proper Galois-theoretic object to study in this scenario is the Weil-Deligne representation, which consists of the datum of a representation of the Weil group, together with a monodromy operator. See [Tat79].) The bijection manifests as an agreement of L- and ε-factors that one constructs for each category. The foundation for the Local Langlands Correspondence for GL_2 over a non-archimedean field was laid in [JL70]; the work was completed in [Kut80] and [Kut84].

We remark that the conductor of π agrees with the Artin conductor of $\sigma(\pi)$, and that π is principal series (resp., Steinberg, supercuspidal) if and only if $\sigma(\pi)$ is a sum of two characters (resp., reducible but not decomposable, irreducible). We also remark that if $p \neq 2$ and σ is an irreducible 2-dimensional Galois representation of $\mathbf{Q}_p$, then σ must be induced from a character χ of a quadratic field extension $E/\mathbf{Q}_p$. Then (E, χ) is called an *admissible pair* (see [BH06], Ch. 18).

145.1.6 Connection with Modular Forms

The classical theory of modular forms has a modern interpretation in terms of *cuspidal automorphic representations.* These are representations Π of the adele group $\mathrm{GL}_2(\mathbf{A_Q})$ which appear in the Hilbert space of square-integrable cuspidal functions $L_0^2(\mathrm{GL}_2(\mathbf{Q})\backslash\mathrm{GL}_2(\mathbf{A_Q}), \varepsilon)$, where ε is a Dirichlet character. Let f be a cuspidal newform for $\Gamma_0(N)$ with Dirichlet character ε. Then there is associated to f a cuspidal automorphic representation Π_f, see [Gel75]. This is a restricted tensor product $\bigotimes_{p \leq \infty} \pi_{f,p}$, where if p is a finite prime, $\pi_{f,p}$ is an admissible representation of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$. There is also the Galois representation ρ_f attached to f constructed by Deligne. By [Car83] there is a straightforward relationship between $\sigma(\pi_{f,p})$ and the restriction of ρ_f to the decomposition group at p. Therefore to determine the local properties of ρ_f it is enough to compute the local components $\pi_{f,p}$. These are almost always unramified principal series; the only challenge is to compute $\pi_{f,p}$ when p divides N.

145.1.7 Category

In MAGMA, admissible representations are objects of type `RepLoc`.

145.1.8 Verbose Output

To see information about computations in progress, enter `SetVerbose("RepLoc", 1)`.

145.2 Creation of Admissible Representations

One starts with a classical cuspidal eigenform, given as a space of modular symbols.

`LocalComponent(M, p)`

This returns the admissible representation of $\mathrm{GL}(\mathbf{Q}_\mathrm{p})$ associated to the cuspidal eigenform specified by M. Here M must be a space of modular symbols that is cuspidal and contains only a single Galois conjugacy class of newforms. (Such spaces are created using `NewformDecomposition`).

Example H145E1

We create the local component at 11 of the representation associated to the newform of level 11 and weight 2. We specify the newform as a space of modular symbols of level 11, weight 2 and sign +1.

```
> S11 := CuspidalSubspace(ModularSymbols(11, 2, 1));
> newform_spaces := NewformDecomposition(S11);
> newform_spaces;
[
Modular symbols space for Gamma_0(11) of weight 2 and dimension 1
   over Rational Field
]
> Eigenform(newform_spaces[1]);
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
> LocalComponent(newform_spaces[1], 11);
Steinberg Representation of GL(2,Q_11)
```

145.3 Attributes of Admissible Representations

`CentralCharacter(pi)`

The central character of π, where π is an admissible representation on $\mathrm{GL}(\mathbf{Q}_\mathrm{p})$. This is a Dirichlet character of p-power conductor.

`Conductor(pi)`

The conductor of π, written multiplicatively.

`DefiningModularSymbolsSpace(pi)`

The space of modular symbols from which π was created.

`IsMinimal(pi)`

Given a representation π of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$, returns `true` if the conductor of π cannot be lowered by twisting by a character of $\mathbf{Q}_p^\times$. If π is not minimal, the function also returns a minimal representation π' together with a Dirichlet character χ, such that π is the twist of π' by χ.

This is true iff `IsMinimalTwist(DefiningModularSymbolsSpace(pi))` is true.

Example H145E2

We continue the previous example.

```
> S11 := CuspidalSubspace(ModularSymbols(11, 2, 1));
> E11 := NewformDecomposition(S11)[1];
> E11;
Modular symbols space for Gamma_0(11) of weight 2 and dimension 1
   over Rational Field
> pi := LocalComponent(E11, 11);
> pi;
Steinberg Representation of GL(2,Q_11)
> DefiningModularSymbolsSpace(pi) eq E11;
true
> Conductor(pi);
11
> IsTrivial(CentralCharacter(pi));
true
```

145.4 Structure of Admissible Representations

`IsPrincipalSeries(pi)`

This is `true` iff the admissible representation π belongs to the principal series.

`IsSupercuspidal(pi)`

This is `true` iff the admissible representation π is supercuspidal.

`PrincipalSeriesParameters(pi)`

Given a principal series representation π of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$, this returns two Dirichlet characters of p-power conductor which represent the restriction to $\mathbf{Z}_p^\times \times \mathbf{Z}_p^\times$ of the character of the split torus of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$ associated to π.

`CuspidalInducingDatum(pi)`

Given a minimal supercuspidal representation π of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$, this returns a cuspidal inducing datum that gives rise to π.

Recall (from Section 145.1.4) that a cuspidal inducing datum (K, Ξ) consists of a subgroup K of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$ and a representation Ξ of K that gives rise to π via induction. Importantly, Ξ factors through some finite quotient K/K_1 of K. This function returns such a representation of K/K_1. From this one can deduce the representation on K, and hence π.

145.5 Local Galois Representations

GaloisRepresentation(pi)

WeilRepresentation(pi)

Precision RngIntElt *Default :* 10

Given a minimal representation π of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$, this returns the representation ρ_π of the Weil-Deligne group associated to π under the local Langlands correspondence, as a local Galois representation. (See Section 145.1.5 and Chapter 56.) In the current implementation, the representation only agrees with ρ_π on inertia.

AdmissiblePair(pi)

Given an ordinary minimal supercuspidal representation π of $\mathrm{GL}_2(\mathbf{Q}_\mathrm{p})$, this returns the associated admissible pair (E, χ). (See Section 145.1.5.) Two objects are returned: a quadratic field extension $E/\mathbf{Q}_p$, and a map χ which is a character of the unit group of E.

145.6 Examples

Example H145E3

We consider a newform of weight 5 and level 7, whose local representation at 7 is principal series.

```
> S := CuspidalSubspace(ModularSymbols(Gamma1(7), 5, 1));
> newforms := NewformDecomposition(S);
> Eigenform(newforms[1], 15);
q + q^2 - 15*q^4 + 49*q^7 - 31*q^8 + 81*q^9 - 206*q^11 + 49*q^14 + O(q^15)
> pi := LocalComponent(newforms[1], 7);
> pi;
Ramified Principal Series Representation of GL(2,Q_7)
> chi := CentralCharacter(pi);
> Conductor(chi);
7
> parameters := PrincipalSeriesParameters(pi);
```

These are Dirichlet characters on $\mathbf{Z}/7\mathbf{Z}$ (the trivial character and the character of order 2):

```
> Conductor(parameters[1]), Order(parameters[1]);
1 1
> Conductor(parameters[2]), Order(parameters[2]);
7 2
```

The principal series representation π is the induction up to $\mathrm{GL}_2(\mathbf{Q}_7)$ of a character of the Borel subgroup inflated from a character of the diagonal group $\mathbf{Q}_7^\times \times \mathbf{Q}_7^\times$. The restriction of this character to $\mathbf{Z}_7^\times \times \mathbf{Z}_7^\times$ gives the pair of Dirichlet characters above. We now compute the Galois representation.

```
> rho := WeilRepresentation(pi); rho;
2-dim Galois representation (2,0) with G=C2, I=C2, conductor 7^1 over Q7[40]
```

```
> IsAbelian(Group(rho));
true
```

The Weil representation is simply the sum of the two characters above (up to unramified twists), considered as characters of the Galois group of $\mathbf{Q}_7$ via local class field theory.

```
> Decomposition(rho);
[
1-dim trivial Galois representation 1 over Q7[40],
1-dim Galois representation (1,-1) with G=C2, I=C2, conductor 7^1 over Q7[40]
]
```

Example H145E4

We consider a supercuspidal representation of conductor 121, associated to a newform of weight 2 and level 121.

```
> S := CuspidalSubspace(ModularSymbols(Gamma0(121), 2, 1));
> newforms := NewformDecomposition(S);
> newforms;
[
Modular symbols space for Gamma_0(121) of weight 2 and dimension 1
   over Rational Field,
Modular symbols space for Gamma_0(121) of weight 2 and dimension 1
   over Rational Field,
Modular symbols space for Gamma_0(121) of weight 2 and dimension 1
   over Rational Field,
Modular symbols space for Gamma_0(121) of weight 2 and dimension 1
   over Rational Field,
Modular symbols space for Gamma_0(121) of weight 2 and dimension 2
   over Rational Field
]
> Eigenform(newforms[2], 11);
q + q^2 + 2*q^3 - q^4 + q^5 + 2*q^6 - 2*q^7 - 3*q^8 + q^9 + q^10 + O(q^11)
> pi := LocalComponent(newforms[2], 11);
> pi;
Supercuspidal Representation of GL(2,Q_11)
```

This means the representation of the Weil group associated to pi is irreducible.

```
> Conductor(pi);
121
> W := CuspidalInducingDatum(pi);
> W;
GModule W of dimension 10 over Rational Field
```

W is a module over a group which is a quotient of $\mathrm{GL}_2(\mathbf{Z}_{11})$, namely $\mathrm{GL}_2(\mathbf{Z}/11\mathbf{Z})$. The representation π is induced from some extension of W to the open subgroup $\mathbf{Q}_{11}^\times \mathrm{GL}_2(\mathbf{Z}_{11})$.

```
> Group(W);
MatrixGroup(2, IntegerRing(11)) of order 2^4 * 3 * 5^2 * 11
```

```
Generators:
[2 0]
[0 1]
[1 1]
[0 1]
[ 0 1]
[10 0]
> Group(W) eq GL(2, Integers(11));
true
> rho:=WeilRepresentation(pi);
```

This gives the Weil representation attached to π up to multiplication by an unramified twist.

```
> rho;
2-dim Galois representation (2,0,-1) with G=S3, I=C3, conductor 11^2
   over Q11[10]
```

Example H145E5

We consider a supercuspidal representation of conductor 3^3, associated to a newform of weight 4 and level 27.

```
> S := CuspidalSubspace(ModularSymbols(Gamma0(27), 4, 1));
> newforms := NewformDecomposition(S);
> Eigenform(newforms[1], 13);
q + 3*q^2 + q^4 + 15*q^5 - 25*q^7 - 21*q^8 + 45*q^10 - 15*q^11 + O(q^13)
> pi:=LocalComponent(newforms[1], 3);
> pi;
Supercuspidal Representation of GL(2,Q_3)
> W:=CuspidalInducingDatum(pi);
> W;
GModule W of dimension 2 over Rational Field
> Group(W);
MatrixGroup(2, IntegerRing(9)) of order 2^2 * 3^5
Generators:
[1 1]
[0 1]
[2 0]
[0 1]
[1 0]
[0 2]
[1 0]
[3 1]
```

These matrices generate (topologically) the Iwahori subgroup of $\mathrm{GL}_2(\mathbf{Z}_3)$ consisting of matrices which are upper-triangular modulo 3. W is an irreducible two-dimensional G-module. The representation π is induced from some extension of W to the normalizer of the Iwahori in $\mathrm{GL}_2(\mathbf{Q}_3)$.

```
> E, chi:=AdmissiblePair(pi);
> E;
```

```
Totally ramified extension defined by the polynomial x^2 - 3
 over 3-adic ring mod 3^10
> E.1^2;
3
> chi(1+E.1);
-zeta_3 - 1
```

Note that `chi` can only be evaluated on units of E, so that `chi(E.1)` would result in an error.

```
> WeilRepresentation(pi);
2-dim Galois representation (2,0,-1) with G=S3, I=S3, conductor 3^3 over Q3[10]
```

145.7 Bibliography

[**BH06**] Colin J. Bushnell and Guy Henniart. *The local Langlands conjecture for* GL(2), volume 335 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 2006.

[**Car83**] Henri Carayol. Sur les Représentations ℓ-adiques attachees aux formes modulaires de Hilbert. *C. R. Acad. Sci. Paris.*, 296(15):629–632, 1983.

[**Cas73**] W. Casselman. The restriction of a representation of $\mathrm{GL}_2(\mathrm{k})$ to $\mathrm{GL}_2(\mathrm{O})$. *Mathematischen Annalen*, 206(4), 1973.

[**Gel75**] S. Gelbart. *Automorphic forms on adele groups.* Princeton University Press, 1975.

[**JL70**] H. Jacquet and R. P. Langlands. *Automorphic forms on* GL(2). Lecture Notes in Mathematics, Vol. 114. Springer-Verlag, Berlin, 1970.

[**Kut80**] Philip Kutzko. The Langlands conjecture for Gl_2 of a local field. *Ann. of Math. (2)*, 112(2):381–412, 1980.

[**Kut84**] P. C. Kutzko. The exceptional representations of Gl_2. *Compositio Math.*, 51(1):3–14, 1984.

[**LW**] David Loeffler and Jared Weinstein. On the computation of local components of a newform. preprint.

[**Tat79**] J. Tate. Number Theoretic Background. *Proc. Symp. Pure Math.*, 33, part 2:3–26, 1979.