Modular Symbols

Release 9.7

The Sage Development Team

Jul 21, 2024

CONTENTS

CHAPTER

ONE

CREATION OF MODULAR SYMBOLS SPACES

EXAMPLES: We create a space and output its category.

```
sage: C = HeckeModules(RationalField()); C
Category of Hecke modules over Rational Field
sage: M = ModularSymbols(11)
sage: M.category()
Category of Hecke modules over Rational Field
sage: M in C
True
```

We create a space compute the charpoly, then compute the same but over a bigger field. In each case we also decompose the space using T_2.

```
sage: M = ModularSymbols(23,2,base_ring=QQ)
sage: M.T(2).charpoly('x').factor()
(x - 3) * (x^2 + x - 1)^2
sage: M.decomposition(2)
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for
↪Gamma_0(23) of weight 2 with sign 0 over Rational Field,
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5 for
↪Gamma_0(23) of weight 2 with sign 0 over Rational Field
]
```

```
sage: M = ModularSymbols(23,2,base_ring=QuadraticField(5, 'sqrt5'))
sage: M.T(2).charpoly('x').factor()
(x - 3) * (x - 1/2*sqrt5 + 1/2)^2 * (x + 1/2*sqrt5 + 1/2)^2
sage: M.decomposition(2)
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for
↪Gamma_0(23) of weight 2 with sign 0 over Number Field in sqrt5 with defining
↪polynomial x^2 - 5 with sqrt5 = 2.236067977499790?,
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for
↪Gamma_0(23) of weight 2 with sign 0 over Number Field in sqrt5 with defining
↪polynomial x^2 - 5 with sqrt5 = 2.236067977499790?,
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for
↪Gamma_0(23) of weight 2 with sign 0 over Number Field in sqrt5 with defining
↪polynomial x^2 - 5 with sqrt5 = 2.236067977499790?
]
```

We compute some Hecke operators and do a consistency check:

```
sage: m = ModularSymbols(39, 2)
sage: t2 = m.T(2); t5 = m.T(5)
sage: t2*t5 - t5*t2 == 0
True
```

This tests the bug reported in trac ticket #1220:

```
sage: G = GammaH(36, [13, 19])
sage: G.modular_symbols()
Modular Symbols space of dimension 13 for Congruence Subgroup Gamma_H(36) with H␣
↪generated by [13, 19] of weight 2 with sign 0 over Rational Field
sage: G.modular_symbols().cuspidal_subspace()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 13 for␣
↪Congruence Subgroup Gamma_H(36) with H generated by [13, 19] of weight 2 with sign 0␣
↪over Rational Field
```

This test catches a tricky corner case for spaces with character:

```
sage: ModularSymbols(DirichletGroup(20).1**3, weight=3, sign=1).cuspidal_subspace()
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 6 and␣
↪level 20, weight 3, character [1, -zeta4], sign 1, over Cyclotomic Field of order 4␣
↪and degree 2
```

This tests the bugs reported in trac ticket #20932:

```
sage: chi = kronecker_character(3*34603)
sage: ModularSymbols(chi, 2, sign=1, base_ring=GF(3)) # not tested # long time (600␣
↪seconds)
Modular Symbols space of dimension 11535 and level 103809, weight 2, character [2, 2],␣
↪sign 1, over Finite Field of size 3
sage: chi = kronecker_character(3*61379)
sage: ModularSymbols(chi, 2, sign=1, base_ring=GF(3)) # not tested # long time (1800␣
↪seconds)
Modular Symbols space of dimension 20460 and level 184137, weight 2, character [2, 2],␣
↪sign 1, over Finite Field of size 3
```

sage.modular.modsym.modsym.**ModularSymbols**(*group=1*, *weight=2*, *sign=0*, *base_ring=None*, *use_cache=True*, *custom_init=None*)

Create an ambient space of modular symbols.

INPUT:

- `group` - A congruence subgroup or a Dirichlet character eps.
- `weight` - int, the weight, which must be >= 2.
- `sign` - int, The sign of the involution on modular symbols induced by complex conjugation. The default is 0, which means "no sign", i.e., take the whole space.
- `base_ring` - the base ring. Defaults to $\mathbf{Q}$ if no character is given, or to the minimal extension of $\mathbf{Q}$ containing the values of the character.
- `custom_init` - a function that is called with self as input before any computations are done using self; this could be used to set a custom modular symbols presentation. If self is already in the cache and use_cache=True, then this function is not called.

EXAMPLES: First we create some spaces with trivial character:

```
sage: ModularSymbols(Gamma0(11),2).dimension()
3
sage: ModularSymbols(Gamma0(1),12).dimension()
3
```

If we give an integer N for the congruence subgroup, it defaults to $\Gamma_0(N)$:

```
sage: ModularSymbols(1,12,-1).dimension()
1
sage: ModularSymbols(11,4, sign=1)
Modular Symbols space of dimension 4 for Gamma_0(11) of weight 4 with sign 1 over␣
↪Rational Field
```

We create some spaces for $\Gamma_1(N)$.

```
sage: ModularSymbols(Gamma1(13),2)
Modular Symbols space of dimension 15 for Gamma_1(13) of weight 2 with sign 0 over␣
↪Rational Field
sage: ModularSymbols(Gamma1(13),2, sign=1).dimension()
13
sage: ModularSymbols(Gamma1(13),2, sign=-1).dimension()
2
sage: [ModularSymbols(Gamma1(7),k).dimension() for k in [2,3,4,5]]
[5, 8, 12, 16]
sage: ModularSymbols(Gamma1(5),11).dimension()
20
```

We create a space for $\Gamma_H(N)$:

```
sage: G = GammaH(15,[4,13])
sage: M = ModularSymbols(G,2)
sage: M.decomposition()
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for␣
↪Congruence Subgroup Gamma_H(15) with H generated by [4, 7] of weight 2 with sign␣
↪0 over Rational Field,
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 5 for␣
↪Congruence Subgroup Gamma_H(15) with H generated by [4, 7] of weight 2 with sign␣
↪0 over Rational Field
]
```

We create a space with character:

```
sage: e = (DirichletGroup(13).0)^2
sage: e.order()
6
sage: M = ModularSymbols(e, 2); M
Modular Symbols space of dimension 4 and level 13, weight 2, character [zeta6],␣
↪sign 0, over Cyclotomic Field of order 6 and degree 2
sage: f = M.T(2).charpoly('x'); f
x^4 + (-zeta6 - 1)*x^3 - 8*zeta6*x^2 + (10*zeta6 - 5)*x + 21*zeta6 - 21
sage: f.factor()
(x - zeta6 - 2) * (x - 2*zeta6 - 1) * (x + zeta6 + 1)^2
```

We create a space with character over a larger base ring than the values of the character:

```
sage: ModularSymbols(e, 2, base_ring = CyclotomicField(24))
Modular Symbols space of dimension 4 and level 13, weight 2, character [zeta24^4],␣
↪sign 0, over Cyclotomic Field of order 24 and degree 8
```

More examples of spaces with character:

```
sage: e = DirichletGroup(5, RationalField()).gen(); e
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -1

sage: m = ModularSymbols(e, 2); m
Modular Symbols space of dimension 2 and level 5, weight 2, character [-1], sign 0,␣
↪over Rational Field
```

```
sage: m.T(2).charpoly('x')
x^2 - 1
sage: m = ModularSymbols(e, 6); m.dimension()
6
sage: m.T(2).charpoly('x')
x^6 - 873*x^4 - 82632*x^2 - 1860496
```

We create a space of modular symbols with nontrivial character in characteristic 2.

```
sage: G = DirichletGroup(13,GF(4,'a')); G
Group of Dirichlet characters modulo 13 with values in Finite Field in a of size 2^2
sage: e = G.list()[2]; e
Dirichlet character modulo 13 of conductor 13 mapping 2 |--> a + 1
sage: M = ModularSymbols(e,4); M
Modular Symbols space of dimension 8 and level 13, weight 4, character [a + 1],␣
↪sign 0, over Finite Field in a of size 2^2
sage: M.basis()
([X*Y,(1,0)], [X*Y,(1,5)], [X*Y,(1,10)], [X*Y,(1,11)], [X^2,(0,1)], [X^2,(1,10)],␣
↪[X^2,(1,11)], [X^2,(1,12)])
sage: M.T(2).matrix()
[    0     0     0     0     0     0     1     1]
[    0     0     0     0     0     0     0     0]
[    0     0     0     0     0 a + 1     1     a]
[    0     0     0     0     0     1 a + 1     a]
[    0     0     0     0 a + 1     0     1     1]
[    0     0     0     0     0     a     1     a]
[    0     0     0     0     0     0 a + 1     a]
[    0     0     0     0     0     0     1     0]
```

We illustrate the custom_init function, which can be used to make arbitrary changes to the modular symbols object before its presentation is computed:

```
sage: ModularSymbols_clear_cache()
sage: def custom_init(M):
....:     M.customize='hi'
sage: M = ModularSymbols(1,12, custom_init=custom_init)
sage: M.customize
'hi'
```

We illustrate the relation between custom_init and use_cache:

```
sage: def custom_init(M):
....:     M.customize='hi2'
sage: M = ModularSymbols(1,12, custom_init=custom_init)
sage: M.customize
'hi'
sage: M = ModularSymbols(1,12, custom_init=custom_init, use_cache=False)
sage: M.customize
'hi2'
```

sage.modular.modsym.modsym.**ModularSymbols_clear_cache**()

Clear the global cache of modular symbols spaces.

EXAMPLES:

```
sage: sage.modular.modsym.modsym.ModularSymbols_clear_cache()
sage: sorted(sage.modular.modsym.modsym._cache)
[]
sage: M = ModularSymbols(6,2)
sage: sorted(sage.modular.modsym.modsym._cache)
[(Congruence Subgroup Gamma0(6), 2, 0, Rational Field)]
sage: sage.modular.modsym.modsym.ModularSymbols_clear_cache()
sage: sorted(sage.modular.modsym.modsym._cache)
[]
```

sage.modular.modsym.modsym.**canonical_parameters**(*group*, *weight*, *sign*, *base_ring*)

Return the canonically normalized parameters associated to a choice of group, weight, sign, and base_ring. That is, normalize each of these to be of the correct type, perform all appropriate type checking, etc.

EXAMPLES:

```
sage: p1 = sage.modular.modsym.modsym.canonical_parameters(5,int(2),1,QQ) ; p1
(Congruence Subgroup Gamma0(5), 2, 1, Rational Field)
sage: p2 = sage.modular.modsym.modsym.canonical_parameters(Gamma0(5),2,1,QQ) ; p2
(Congruence Subgroup Gamma0(5), 2, 1, Rational Field)
sage: p1 == p2
True
sage: type(p1[1])
<class 'sage.rings.integer.Integer'>
```

CHAPTER

TWO

SPACE OF MODULAR SYMBOLS (BASE CLASS)

All the spaces of modular symbols derive from this class. This class is an abstract base class.

class sage.modular.modsym.space.**IntegralPeriodMapping**(*modsym*, *A*)

Bases: *sage.modular.modsym.space.PeriodMapping*

class sage.modular.modsym.space.**ModularSymbolsSpace**(*group*, *weight*, *character*, *sign*, *base_ring*, *category=None*)

Bases: sage.modular.hecke.module.HeckeModule_free_module

Base class for spaces of modular symbols.

Element

alias of *sage.modular.modsym.element.ModularSymbolsElement*

abelian_variety()

Return the corresponding abelian variety.

INPUT:

- self - modular symbols space of weight 2 for a congruence subgroup such as Gamma0, Gamma1 or GammaH.

EXAMPLES:

```
sage: ModularSymbols(Gamma0(11)).cuspidal_submodule().abelian_variety()
Abelian variety J0(11) of dimension 1
sage: ModularSymbols(Gamma1(11)).cuspidal_submodule().abelian_variety()
Abelian variety J1(11) of dimension 1
sage: ModularSymbols(GammaH(11,[3])).cuspidal_submodule().abelian_variety()
Abelian variety JH(11,[3]) of dimension 1
```

The abelian variety command only works on cuspidal modular symbols spaces:

```
sage: M = ModularSymbols(37)
sage: M[0].abelian_variety()
Traceback (most recent call last):
...
ValueError: self must be cuspidal
sage: M[1].abelian_variety()
Abelian subvariety of dimension 1 of J0(37)
sage: M[2].abelian_variety()
Abelian subvariety of dimension 1 of J0(37)
```

abvarquo_cuspidal_subgroup()

Compute the rational subgroup of the cuspidal subgroup (as an abstract abelian group) of the abelian variety quotient A of the relevant modular Jacobian attached to this modular symbols space.

We assume that `self` is defined over QQ and has weight 2. If the sign of `self` is not 0, then the power of 2 may be wrong.

EXAMPLES:

```
sage: D = ModularSymbols(66,2,sign=0).cuspidal_subspace().new_subspace().
↪decomposition()
sage: D[0].abvarquo_cuspidal_subgroup()
Finitely generated module V/W over Integer Ring with invariants (3)
sage: [A.abvarquo_cuspidal_subgroup().invariants() for A in D]
[(3,), (2,), ()]
sage: D = ModularSymbols(66,2,sign=1).cuspidal_subspace().new_subspace().
↪decomposition()
sage: [A.abvarquo_cuspidal_subgroup().invariants() for A in D]
[(3,), (2,), ()]
sage: D = ModularSymbols(66,2,sign=-1).cuspidal_subspace().new_subspace().
↪decomposition()
sage: [A.abvarquo_cuspidal_subgroup().invariants() for A in D]
[(), (), ()]
```

abvarquo_rational_cuspidal_subgroup()

Compute the rational subgroup of the cuspidal subgroup (as an abstract abelian group) of the abelian variety quotient A of the relevant modular Jacobian attached to this modular symbols space.

If C is the subgroup of A generated by differences of cusps, then C is equipped with an action of Gal(Qbar/Q), and this function computes the fixed subgroup, i.e., C(Q).

We assume that `self` is defined over QQ and has weight 2. If the sign of `self` is not 0, then the power of 2 may be wrong.

EXAMPLES:

First we consider the fairly straightforward level 37 case, where the torsion subgroup of the optimal quotients (which are all elliptic curves) are all cuspidal:

```
sage: M = ModularSymbols(37).cuspidal_subspace().new_subspace()
sage: D = M.decomposition()
sage: [(A.abvarquo_rational_cuspidal_subgroup().invariants(), A.T(19)[0,0]) for␣
↪A in D]
[((), 0), ((3,), 2)]
sage: [(E.torsion_subgroup().invariants(),E.ap(19)) for E in cremona_optimal_
↪curves([37])]
[((), 0), ((3,), 2)]
```

Next we consider level 54, where the rational cuspidal subgroups of the quotients are also cuspidal:

```
sage: M = ModularSymbols(54).cuspidal_subspace().new_subspace()
sage: D = M.decomposition()
sage: [A.abvarquo_rational_cuspidal_subgroup().invariants() for A in D]
[(3,), (3,)]
sage: [E.torsion_subgroup().invariants() for E in cremona_optimal_curves([54])]
[(3,), (3,)]
```

Level 66 is interesting, since not all torsion of the quotient is rational. In fact, for each elliptic curve quotient, the $\mathbf{Q}$-rational subgroup of the image of the cuspidal subgroup in the quotient is a nontrivial subgroup of $E(\mathbf{Q})_{tor}$. Thus not all torsion in the quotient is cuspidal!:

```
sage: M = ModularSymbols(66).cuspidal_subspace().new_subspace()
sage: D = M.decomposition()
sage: [(A.abvarquo_rational_cuspidal_subgroup().invariants(), A.T(19)[0,0]) for␣
↪A in D]
[((3,), -4), ((2,), 4), ((), 0)]
sage: [(E.torsion_subgroup().invariants(),E.ap(19)) for E in cremona_optimal_
↪curves([66])]
[((6,), -4), ((4,), 4), ((10,), 0)]
sage: [A.abelian_variety().rational_cuspidal_subgroup().invariants() for A in D]
[[6], [4], [10]]
```

In this example, the abelian varieties involved all having dimension bigger than 1 (unlike above). We find that all torsion in the quotient in each of these cases is cuspidal:

```
sage: M = ModularSymbols(125).cuspidal_subspace().new_subspace()
sage: D = M.decomposition()
sage: [A.abvarquo_rational_cuspidal_subgroup().invariants() for A in D]
[(), (5,), (5,)]
sage: [A.abelian_variety().rational_torsion_subgroup().multiple_of_order() for␣
↪A in D]
[1, 5, 5]
```

character()

Return the character associated to `self`.

EXAMPLES:

```
sage: ModularSymbols(12,8).character()
Dirichlet character modulo 12 of conductor 1 mapping 7 |--> 1, 5 |--> 1
sage: ModularSymbols(DirichletGroup(25).0, 4).character()
Dirichlet character modulo 25 of conductor 25 mapping 2 |--> zeta20
```

compact_system_of_eigenvalues(*v*, *names='alpha'*, *nz=None*)

Return a compact system of eigenvalues a_n for $n \in v$.

This should only be called on simple factors of modular symbols spaces.

INPUT:

- `v` - a list of positive integers
- `nz` - (default: `None`); if given specifies a column index such that the dual module has that column nonzero.

OUTPUT:

- `E` - matrix such that E*v is a vector with components the eigenvalues a_n for $n \in v$.
- `v` - a vector over a number field

EXAMPLES:

```
sage: M = ModularSymbols(43,2,1)[2]; M
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4␣
↪for Gamma_0(43) of weight 2 with sign 1 over Rational Field
```

(continues on next page)

(continued from previous page)

```
sage: E, v = M.compact_system_of_eigenvalues(prime_range(10))
sage: E
[ 2/3 -4/3]
[-2/3  4/3]
[ 4/3  4/3]
[-4/3 -4/3]
sage: v
(1, -3/4*alpha + 1/2)
sage: E*v
(alpha, -alpha, -alpha + 2, alpha - 2)
```

congruence_number(*other*, *prec=None*)

Given two cuspidal spaces of modular symbols, compute the congruence number, using `prec` terms of the q-expansions.

The congruence number is defined as follows. If V is the submodule of integral cusp forms corresponding to self (saturated in $\mathbf{Z}[[q]]$, by definition) and W is the submodule corresponding to other, each computed to precision prec, the congruence number is the index of $V + W$ in its saturation in $\mathbf{Z}[[q]]$.

If `prec` is not given it is set equal to the max of the `hecke_bound` function called on each space.

EXAMPLES:

```
sage: A, B = ModularSymbols(48, 2).cuspidal_submodule().decomposition()
sage: A.congruence_number(B)
2
```

cuspidal_submodule()

Return the cuspidal submodule of `self`.

Note: This should be overridden by all derived classes.

EXAMPLES:

```
sage: sage.modular.modsym.space.ModularSymbolsSpace(Gamma0(11),2,
↪DirichletGroup(11).gens()[0]**10,0,QQ).cuspidal_submodule()
Traceback (most recent call last):
...
NotImplementedError: computation of cuspidal submodule not yet implemented for␣
↪this class
sage: ModularSymbols(Gamma0(11),2).cuspidal_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3␣
↪for Gamma_0(11) of weight 2 with sign 0 over Rational Field
```

cuspidal_subspace()

Synonym for cuspidal_submodule.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma1(3),12); m.dimension()
8
sage: m.cuspidal_subspace().new_subspace().dimension()
2
```

default_prec()

Get the default precision for computation of q-expansion associated to the ambient space of this space of modular symbols (and all subspaces). Use `set_default_prec` to change the default precision.

EXAMPLES:

```
sage: M = ModularSymbols(15)
sage: M.cuspidal_submodule().q_expansion_basis()
[
q - q^2 - q^3 - q^4 + q^5 + q^6 + O(q^8)
]
sage: M.set_default_prec(20)
```

Notice that setting the default precision of the ambient space affects the subspaces.

```
sage: M.cuspidal_submodule().q_expansion_basis()
[
q - q^2 - q^3 - q^4 + q^5 + q^6 + 3*q^8 + q^9 - q^10 - 4*q^11 + q^12 - 2*q^13 -␣
↪q^15 - q^16 + 2*q^17 - q^18 + 4*q^19 + O(q^20)
]
sage: M.cuspidal_submodule().default_prec()
20
```

dimension_of_associated_cuspform_space()

Return the dimension of the corresponding space of cusp forms.

The input space must be cuspidal, otherwise there is no corresponding space of cusp forms.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma0(389),2).cuspidal_subspace(); m.dimension()
64
sage: m.dimension_of_associated_cuspform_space()
32
sage: m = ModularSymbols(Gamma0(389),2,sign=1).cuspidal_subspace(); m.
↪dimension()
32
sage: m.dimension_of_associated_cuspform_space()
32
```

dual_star_involution_matrix()

Return the matrix of the dual star involution, which is induced by complex conjugation on the linear dual of modular symbols.

Note: This should be overridden in all derived classes.

EXAMPLES:

```
sage: sage.modular.modsym.space.ModularSymbolsSpace(Gamma0(11),2,
↪DirichletGroup(11).gens()[0]**10,0,QQ).dual_star_involution_matrix()
Traceback (most recent call last):
...
NotImplementedError: computation of dual star involution matrix not yet␣
↪implemented for this class
sage: ModularSymbols(Gamma0(11),2).dual_star_involution_matrix()
```

(continues on next page)

(continued from previous page)

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

eisenstein_subspace()

Synonym for eisenstein_submodule.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma1(3),12); m.dimension()
8
sage: m.eisenstein_subspace().dimension()
2
sage: m.cuspidal_subspace().dimension()
6
```

group()

Return the group of this modular symbols space.

INPUT:

- `ModularSymbols self` - an arbitrary space of modular symbols

OUTPUT:

- `CongruenceSubgroup` - the congruence subgroup that this is a space of modular symbols for.

ALGORITHM: The group is recorded when this space is created.

EXAMPLES:

```
sage: m = ModularSymbols(20)
sage: m.group()
Congruence Subgroup Gamma0(20)
```

hecke_module_of_level(*level*)

Alias for `self.modular_symbols_of_level(level)`.

EXAMPLES:

```
sage: ModularSymbols(11, 2).hecke_module_of_level(22)
Modular Symbols space of dimension 7 for Gamma_0(22) of weight 2 with sign 0␣
↪over Rational Field
```

integral_basis()

Return a basis for the $\mathbf{Z}$-submodule of this modular symbols space spanned by the generators.

Modular symbols spaces for congruence subgroups have a $\mathbf{Z}$-structure. Computing this $\mathbf{Z}$-structure is expensive, so by default modular symbols spaces for congruence subgroups in Sage are defined over $\mathbf{Q}$. This function returns a tuple of independent elements in this modular symbols space whose $\mathbf{Z}$-span is the corresponding space of modular symbols over $\mathbf{Z}$.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: M.basis()
((1,0), (1,8), (1,9))
sage: M.integral_basis()
```

(continues on next page)

(continued from previous page)

```
((1,0), (1,8), (1,9))
sage: S = M.cuspidal_submodule()
sage: S.basis()
((1,8), (1,9))
sage: S.integral_basis()
((1,8), (1,9))
```

```
sage: M = ModularSymbols(13,4)
sage: M.basis()
([X^2,(0,1)], [X^2,(1,4)], [X^2,(1,5)], [X^2,(1,7)], [X^2,(1,9)], [X^2,(1,10)],␣
↪[X^2,(1,11)], [X^2,(1,12)])
sage: M.integral_basis()
([X^2,(0,1)], 1/28*[X^2,(1,4)] + 2/7*[X^2,(1,5)] + 3/28*[X^2,(1,7)] + 11/14*[X^
↪2,(1,9)] + 2/7*[X^2,(1,10)] + 11/28*[X^2,(1,11)] + 3/28*[X^2,(1,12)], [X^2,(1,
↪5)], 1/2*[X^2,(1,7)] + 1/2*[X^2,(1,9)], [X^2,(1,9)], [X^2,(1,10)], [X^2,(1,
↪11)], [X^2,(1,12)])
sage: S = M.cuspidal_submodule()
sage: S.basis()
([X^2,(1,4)] - [X^2,(1,12)], [X^2,(1,5)] - [X^2,(1,12)], [X^2,(1,7)] - [X^2,(1,
↪12)], [X^2,(1,9)] - [X^2,(1,12)], [X^2,(1,10)] - [X^2,(1,12)], [X^2,(1,11)] -␣
↪[X^2,(1,12)])
sage: S.integral_basis()
(1/28*[X^2,(1,4)] + 2/7*[X^2,(1,5)] + 3/28*[X^2,(1,7)] + 11/14*[X^2,(1,9)] + 2/
↪7*[X^2,(1,10)] + 11/28*[X^2,(1,11)] - 53/28*[X^2,(1,12)], [X^2,(1,5)] - [X^2,
↪(1,12)], 1/2*[X^2,(1,7)] + 1/2*[X^2,(1,9)] - [X^2,(1,12)], [X^2,(1,9)] - [X^2,
↪(1,12)], [X^2,(1,10)] - [X^2,(1,12)], [X^2,(1,11)] - [X^2,(1,12)])
```

This function currently raises a NotImplementedError on modular symbols spaces with character of order bigger than 2:

EXAMPLES:

```
sage: M = ModularSymbols(DirichletGroup(13).0^2, 2); M
Modular Symbols space of dimension 4 and level 13, weight 2, character [zeta6],␣
↪sign 0, over Cyclotomic Field of order 6 and degree 2
sage: M.basis()
((1,0), (1,5), (1,10), (1,11))
sage: M.integral_basis()
Traceback (most recent call last):
...
NotImplementedError
```

integral_hecke_matrix(n)

Return the matrix of the n (as returned by `self.integral_structure()`).

This is often (but not always) different from the matrix returned by `self.hecke_matrix`, even if the latter has integral entries.

EXAMPLES:

```
sage: M = ModularSymbols(6,4)
sage: M.hecke_matrix(3)
[27  0  0  0  6 -6]
[ 0  1 -4  4  8 10]
```

(continues on next page)

(continued from previous page)

```
[18  0  1  0  6 -6]
[18  0  4 -3  6 -6]
[ 0  0  0  0  9 18]
[ 0  0  0  0 12 15]
sage: M.integral_hecke_matrix(3)
[ 27   0   0   0   6  -6]
[  0   1  -8   8  12  14]
[ 18   0   5  -4  14   8]
[ 18   0   8  -7   2 -10]
[  0   0   0   0   9  18]
[  0   0   0   0  12  15]
```

integral_period_mapping()

Return the integral period mapping associated to `self`.

This is a homomorphism to a vector space whose kernel is the same as the kernel of the period mapping associated to `self`, normalized so the image of integral modular symbols is exactly $\mathbf{Z}^n$.

EXAMPLES:

```
sage: m = ModularSymbols(23).cuspidal_submodule()
sage: i = m.integral_period_mapping()
sage: i
Integral period mapping associated to Modular Symbols subspace of dimension 4␣
↪of Modular Symbols space of dimension 5 for Gamma_0(23) of weight 2 with sign␣
↪0 over Rational Field
sage: i.matrix()
[-1/11  1/11     0  3/11]
[    1     0     0     0]
[    0     1     0     0]
[    0     0     1     0]
[    0     0     0     1]
sage: [i(b) for b in m.integral_structure().basis()]
[(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)]
sage: [i(b) for b in m.ambient_module().basis()]
[(-1/11, 1/11, 0, 3/11),
 (1, 0, 0, 0),
 (0, 1, 0, 0),
 (0, 0, 1, 0),
 (0, 0, 0, 1)]
```

We compute the image of the winding element:

```
sage: m = ModularSymbols(37,sign=1)
sage: a = m[1]
sage: f = a.integral_period_mapping()
sage: e = m([0,oo])
sage: f(e)
(-2/3)
```

The input space must be cuspidal:

```
sage: m = ModularSymbols(37,2,sign=1)
sage: m.integral_period_mapping()
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: integral mapping only defined for cuspidal spaces
```

integral_structure()

Return the $\mathbf{Z}$-structure of this modular symbols spaces generated by all integral modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(11,4)
sage: M.integral_structure()
Free module of degree 6 and rank 6 over Integer Ring
Echelon basis matrix:
[    1     0     0     0     0     0]
[    0  1/14   1/7  5/14   1/2 13/14]
[    0     0   1/2     0     0   1/2]
[    0     0     0     1     0     0]
[    0     0     0     0     1     0]
[    0     0     0     0     0     1]
sage: M.cuspidal_submodule().integral_structure()
Free module of degree 6 and rank 4 over Integer Ring
Echelon basis matrix:
[     0   1/14    1/7   5/14    1/2 -15/14]
[     0      0    1/2      0      0   -1/2]
[     0      0      0      1      0     -1]
[     0      0      0      0      1     -1]
```

intersection_number(*M*)

Given modular symbols spaces `self` and `M` in some common ambient space, returns the intersection number of these two spaces.

This is the index in their saturation of the sum of their underlying integral structures.

If `self` and `M` are of weight two and defined over QQ, and correspond to newforms f and g, then this number equals the order of the intersection of the modular abelian varieties attached to f and g.

EXAMPLES:

```
sage: m = ModularSymbols(389,2)
sage: d = m.decomposition(2)
sage: eis = d[0]
sage: ell = d[1]
sage: af = d[-1]
sage: af.intersection_number(eis)
97
sage: af.intersection_number(ell)
400
```

is_ambient()

Return `True` if `self` is an ambient space of modular symbols.

EXAMPLES:

```
sage: ModularSymbols(21,4).is_ambient()
True
```

(continues on next page)

(continued from previous page)

```
sage: ModularSymbols(21,4).cuspidal_submodule().is_ambient()
False
```

is_cuspidal()

Return True if self is a cuspidal space of modular symbols.

Note: This should be overridden in all derived classes.

EXAMPLES:

```
sage: sage.modular.modsym.space.ModularSymbolsSpace(Gamma0(11),2,
↪DirichletGroup(11).gens()[0]**10,0,QQ).is_cuspidal()
Traceback (most recent call last):
...
NotImplementedError: computation of cuspidal subspace not yet implemented for␣
↪this class
sage: ModularSymbols(Gamma0(11),2).is_cuspidal()
False
```

is_simple()

Return whether this modular symbols space is simple as a module over the anemic Hecke algebra adjoin *.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma0(33),2,sign=1)
sage: m.is_simple()
False
sage: o = m.old_subspace()
sage: o.decomposition()
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 6␣
↪for Gamma_0(33) of weight 2 with sign 1 over Rational Field,
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 6␣
↪for Gamma_0(33) of weight 2 with sign 1 over Rational Field
]
sage: C = ModularSymbols(1,14,0,GF(5)).cuspidal_submodule(); C
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2␣
↪for Gamma_0(1) of weight 14 with sign 0 over Finite Field of size 5
sage: C.is_simple()
True
```

minus_submodule(*compute_dual=True*)

Return the subspace of self on which the star involution acts as -1.

INPUT:

- compute_dual - bool (default: True) also compute dual subspace. This are useful for many algorithms.

OUTPUT: subspace of modular symbols

EXAMPLES:

```
sage: ModularSymbols(14,4)
Modular Symbols space of dimension 12 for Gamma_0(14) of weight 4 with sign 0␣
↪over Rational Field
sage: ModularSymbols(14,4).minus_submodule()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension␣
↪12 for Gamma_0(14) of weight 4 with sign 0 over Rational Field
```

modular_symbols_of_sign(*sign*, *bound=None*)

Return a space of modular symbols with the same defining properties (weight, level, etc.) and Hecke eigenvalues as this space except with given sign.

INPUT:

- `self` - a cuspidal space of modular symbols
- `sign` - an integer, one of -1, 0, or 1
- `bound` - integer (default: None); if specified only use Hecke operators up to the given bound.

EXAMPLES:

```
sage: S = ModularSymbols(Gamma0(11),2,sign=0).cuspidal_subspace()
sage: S
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3␣
↪for Gamma_0(11) of weight 2 with sign 0 over Rational Field
sage: S.modular_symbols_of_sign(-1)
Modular Symbols space of dimension 1 for Gamma_0(11) of weight 2 with sign -1␣
↪over Rational Field
```

```
sage: S = ModularSymbols(43,2,sign=1)[2]; S
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4␣
↪for Gamma_0(43) of weight 2 with sign 1 over Rational Field
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3␣
↪for Gamma_0(43) of weight 2 with sign -1 over Rational Field
```

```
sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 7␣
↪for Gamma_0(43) of weight 2 with sign 0 over Rational Field
```

```
sage: S = ModularSymbols(389,sign=1)[3]; S
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension␣
↪33 for Gamma_0(389) of weight 2 with sign 1 over Rational Field
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension␣
↪32 for Gamma_0(389) of weight 2 with sign -1 over Rational Field
sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 6 of Modular Symbols space of dimension␣
↪65 for Gamma_0(389) of weight 2 with sign 0 over Rational Field
```

```
sage: S = ModularSymbols(23,sign=1,weight=4)[2]; S
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 7␣
↪for Gamma_0(23) of weight 4 with sign 1 over Rational Field
sage: S.modular_symbols_of_sign(1) is S
```

(continues on next page)

(continued from previous page)

```
True
sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension␣
↪12 for Gamma_0(23) of weight 4 with sign 0 over Rational Field
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5␣
↪for Gamma_0(23) of weight 4 with sign -1 over Rational Field
```

multiplicity(*S*, *check_simple=True*)

Return the multiplicity of the simple modular symbols space S in self. S must be a simple anemic Hecke module.

ASSUMPTION: self is an anemic Hecke module with the same weight and group as S, and S is simple.

EXAMPLES:

```
sage: M = ModularSymbols(11,2,sign=1)
sage: N1, N2 = M.decomposition()
sage: N1.multiplicity(N2)
0
sage: M.multiplicity(N1)
1
sage: M.multiplicity(ModularSymbols(14,2))
0
```

new_subspace(*p=None*)

Synonym for new_submodule.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma0(5),12); m.dimension()
12
sage: m.new_subspace().dimension()
6
sage: m = ModularSymbols(Gamma1(3),12); m.dimension()
8
sage: m.new_subspace().dimension()
2
```

ngens()

Return the number of generators of `self`.

INPUT:

- `ModularSymbols self` - arbitrary space of modular symbols.

OUTPUT:

- `int` - the number of generators, which is the same as the dimension of self.

ALGORITHM: Call the dimension function.

EXAMPLES:

```
sage: m = ModularSymbols(33)
sage: m.ngens()
9
```

(continues on next page)

(continued from previous page)

```
sage: m.rank()
9
sage: ModularSymbols(100, weight=2, sign=1).ngens()
18
```

old_subspace(*p=None*)

Synonym for old_submodule.

EXAMPLES:

```
sage: m = ModularSymbols(Gamma1(3),12); m.dimension()
8
sage: m.old_subspace().dimension()
6
```

plus_submodule(*compute_dual=True*)

Return the subspace of `self` on which the star involution acts as +1.

INPUT:

- `compute_dual` - bool (default: `True`) also compute dual subspace. This are useful for many algorithms.

OUTPUT: subspace of modular symbols

EXAMPLES:

```
sage: ModularSymbols(17,2)
Modular Symbols space of dimension 3 for Gamma_0(17) of weight 2 with sign 0␣
↪over Rational Field
sage: ModularSymbols(17,2).plus_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3␣
↪for Gamma_0(17) of weight 2 with sign 0 over Rational Field
```

q_eigenform(*prec*, *names=None*)

Return the q-expansion to precision `prec` of a new eigenform associated to `self`.

Here `self` must be new, cuspidal, and simple.

EXAMPLES:

```
sage: ModularSymbols(2, 8)[1].q_eigenform(5, 'a')
q - 8*q^2 + 12*q^3 + 64*q^4 + O(q^5)
sage: ModularSymbols(2, 8)[0].q_eigenform(5,'a')
Traceback (most recent call last):
...
ArithmeticError: self must be cuspidal.
```

q_eigenform_character(*names=None*)

Return the Dirichlet character associated to the specific choice of q-eigenform attached to this simple cuspidal modular symbols space.

INPUT:

- `names` – string, name of the variable.

OUTPUT:

- a Dirichlet character taking values in the Hecke eigenvalue field, where the indeterminate of that field is determined by the given variable name.

EXAMPLES:

```
sage: f = ModularSymbols(Gamma1(13),2,sign=1).cuspidal_subspace().
↪decomposition()[0]
sage: eps = f.q_eigenform_character('a'); eps
Dirichlet character modulo 13 of conductor 13 mapping 2 |--> -a - 1
sage: parent(eps)
Group of Dirichlet characters modulo 13 with values in Number Field in a with␣
↪defining polynomial x^2 + 3*x + 3
sage: eps(3)
a + 1
```

The modular symbols space must be simple.:

```
sage: ModularSymbols(Gamma1(17),2,sign=1).cuspidal_submodule().q_eigenform_
↪character('a')
Traceback (most recent call last):
...
ArithmeticError: self must be simple
```

If the character is specified when making the modular symbols space, then names need not be given and the returned character is just the character of the space.:

```
sage: f = ModularSymbols(kronecker_character(19),2,sign=1).cuspidal_subspace().
↪decomposition()[0]
sage: f
Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension␣
↪10 and level 76, weight 2, character [-1, -1], sign 1, over Rational Field
sage: f.q_eigenform_character()
Dirichlet character modulo 76 of conductor 76 mapping 39 |--> -1, 21 |--> -1
sage: f.q_eigenform_character() is f.character()
True
```

The input space need not be cuspidal:

```
sage: M = ModularSymbols(Gamma1(13),2,sign=1).eisenstein_submodule()[0]
sage: M.q_eigenform_character('a')
Dirichlet character modulo 13 of conductor 13 mapping 2 |--> -1
```

The modular symbols space does not have to come from a decomposition:

```
sage: ModularSymbols(Gamma1(16),2,sign=1).cuspidal_submodule().q_eigenform_
↪character('a')
Dirichlet character modulo 16 of conductor 16 mapping 15 |--> 1, 5 |--> -a - 1
```

q_expansion_basis(*prec=None*, *algorithm='default'*)

Return a basis of q-expansions (as power series) to precision prec of the space of modular forms associated to `self`.

The q-expansions are defined over the same base ring as `self`, and a put in echelon form.

INPUT:

- `self` - a space of CUSPIDAL modular symbols

- prec - an integer
- algorithm - string:
- 'default' (default) - decide which algorithm to use based on heuristics
- 'hecke' - compute basis by computing homomorphisms T - K, where T is the Hecke algebra
- 'eigen' - compute basis using eigenvectors for the Hecke action and Atkin-Lehner-Li theory to patch them together
- 'all' - compute using hecke_dual and eigen algorithms and verify that the results are the same.

The computed basis is *not* cached, though of course Hecke operators used in computing the basis are cached.

EXAMPLES:

```
sage: M = ModularSymbols(1, 12).cuspidal_submodule()
sage: M.q_expansion_basis(8)
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + O(q^8)
]
```

```
sage: M.q_expansion_basis(8, algorithm='eigen')
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + O(q^8)
]
```

```
sage: M = ModularSymbols(1, 24).cuspidal_submodule()
sage: M.q_expansion_basis(8, algorithm='eigen')
[
q + 195660*q^3 + 12080128*q^4 + 44656110*q^5 - 982499328*q^6 - 147247240*q^7 +
→O(q^8),
q^2 - 48*q^3 + 1080*q^4 - 15040*q^5 + 143820*q^6 - 985824*q^7 + O(q^8)
]
```

```
sage: M = ModularSymbols(11, 2, sign=-1).cuspidal_submodule()
sage: M.q_expansion_basis(8, algorithm='eigen')
[
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
]
```

```
sage: M = ModularSymbols(Gamma1(13), 2, sign=1).cuspidal_submodule()
sage: M.q_expansion_basis(8, algorithm='eigen')
[
q - 4*q^3 - q^4 + 3*q^5 + 6*q^6 + O(q^8),
q^2 - 2*q^3 - q^4 + 2*q^5 + 2*q^6 + O(q^8)
]
```

```
sage: M = ModularSymbols(Gamma1(5), 3, sign=-1).cuspidal_submodule()
sage: M.q_expansion_basis(8, algorithm='eigen')   # dimension is 0
[]
```

```
sage: M = ModularSymbols(Gamma1(7), 3, sign=-1).cuspidal_submodule()
sage: M.q_expansion_basis(8)
```

(continues on next page)

(continued from previous page)

```
[
q - 3*q^2 + 5*q^4 - 7*q^7 + O(q^8)
]
```

```
sage: M = ModularSymbols(43, 2, sign=0).cuspidal_submodule()
sage: M[0]
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7␣
↪for Gamma_0(43) of weight 2 with sign 0 over Rational Field
sage: M[0].q_expansion_basis()
[
q - 2*q^2 - 2*q^3 + 2*q^4 - 4*q^5 + 4*q^6 + O(q^8)
]
sage: M[1]
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 7␣
↪for Gamma_0(43) of weight 2 with sign 0 over Rational Field
sage: M[1].q_expansion_basis()
[
q + 2*q^5 - 2*q^6 - 2*q^7 + O(q^8),
q^2 - q^3 - q^5 + q^7 + O(q^8)
]
```

q_expansion_cuspforms(*prec=None*)

Return a function f(i,j) such that each value f(i,j) is the q-expansion, to the given precision, of an element of the corresponding space S of cusp forms.

Together these functions span S. Here i, j are integers with $0 \leq i, j < d$, where d is the dimension of `self`.

For a reduced echelon basis, use the function `q_expansion_basis` instead.

More precisely, this function returns the q-expansions obtained by taking the ij entry of the matrices of the Hecke operators T_n acting on the subspace of the linear dual of modular symbols corresponding to `self`.

EXAMPLES:

```
sage: S = ModularSymbols(11,2, sign=1).cuspidal_submodule()
sage: f = S.q_expansion_cuspforms(8)
sage: f(0,0)
q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8)
```

```
sage: S = ModularSymbols(37,2).cuspidal_submodule()
sage: f = S.q_expansion_cuspforms(8)
sage: f(0,0)
q + q^3 - 2*q^4 - q^7 + O(q^8)
sage: f(3,3)
q - 2*q^2 - 3*q^3 + 2*q^4 - 2*q^5 + 6*q^6 - q^7 + O(q^8)
sage: f(1,2)
q^2 + 2*q^3 - 2*q^4 + q^5 - 3*q^6 + O(q^8)
```

```
sage: S = ModularSymbols(Gamma1(13),2,sign=-1).cuspidal_submodule()
sage: f = S.q_expansion_cuspforms(8)
sage: f(0,0)
q - 2*q^2 + q^4 - q^5 + 2*q^6 + O(q^8)
```

(continues on next page)

(continued from previous page)

```
sage: f(0,1)
-q^2 + 2*q^3 + q^4 - 2*q^5 - 2*q^6 + O(q^8)
```

```
sage: S = ModularSymbols(1,12,sign=-1).cuspidal_submodule()
sage: f = S.q_expansion_cuspforms(8)
sage: f(0,0)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + O(q^8)
```

q_expansion_module(*prec=None*, *R=None*)

Return a basis over R for the space spanned by the coefficient vectors of the q-expansions corresponding to `self`.

If R is not the base ring of `self`, this returns the restriction of scalars down to R (for this, self must have base ring $\mathbf{Q}$ or a number field).

INPUT:

- `self` - must be cuspidal
- `prec` - an integer (default: self.default_prec())
- `R` - either ZZ, QQ, or the base_ring of self (which is the default)

OUTPUT: A free module over R.

Todo: extend to more general R (though that is fairly easy for the user to get by just doing base_extend or change_ring on the output of this function).

Note that the prec needed to distinguish elements of the restricted-down-to-R basis may be bigger than `self.hecke_bound()`, since one must use the Sturm bound for modular forms on $\Gamma_H(N)$.

EXAMPLES WITH SIGN 1 and R=QQ:

Basic example with sign 1:

```
sage: M = ModularSymbols(11, sign=1).cuspidal_submodule()
sage: M.q_expansion_module(5, QQ)
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0  1 -2 -1  2]
```

Same example with sign -1:

```
sage: M = ModularSymbols(11, sign=-1).cuspidal_submodule()
sage: M.q_expansion_module(5, QQ)
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0  1 -2 -1  2]
```

An example involving old forms:

```
sage: M = ModularSymbols(22, sign=1).cuspidal_submodule()
sage: M.q_expansion_module(5, QQ)
Vector space of degree 5 and dimension 2 over Rational Field
Basis matrix:
```

(continues on next page)

(continued from previous page)

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

An example that (somewhat spuriously) is over a number field:

```
sage: x = polygen(QQ)
sage: k = NumberField(x^2+1, 'a')
sage: M = ModularSymbols(11, base_ring=k, sign=1).cuspidal_submodule()
sage: M.q_expansion_module(5, QQ)
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0  1 -2 -1  2]
```

An example that involves an eigenform with coefficients in a number field:

```
sage: M = ModularSymbols(23, sign=1).cuspidal_submodule()
sage: M.q_eigenform(4, 'gamma')
q + gamma*q^2 + (-2*gamma - 1)*q^3 + O(q^4)
sage: M.q_expansion_module(11, QQ)
Vector space of degree 11 and dimension 2 over Rational Field
Basis matrix:
[ 0  1  0 -1 -1  0 -2  2 -1  2  2]
[ 0  0  1 -2 -1  2  1  2 -2  0 -2]
```

An example that is genuinely over a base field besides QQ.

```
sage: eps = DirichletGroup(11).0
sage: M = ModularSymbols(eps,3,sign=1).cuspidal_submodule(); M
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2␣
↪and level 11, weight 3, character [zeta10], sign 1, over Cyclotomic Field of␣
↪order 10 and degree 4
sage: M.q_eigenform(4, 'beta')
q + (-zeta10^3 + 2*zeta10^2 - 2*zeta10)*q^2 + (2*zeta10^3 - 3*zeta10^2 +␣
↪3*zeta10 - 2)*q^3 + O(q^4)
sage: M.q_expansion_module(7, QQ)
Vector space of degree 7 and dimension 4 over Rational Field
Basis matrix:
[  0   1   0   0   0 -40  64]
[  0   0   1   0   0 -24  41]
[  0   0   0   1   0 -12  21]
[  0   0   0   0   1  -4   4]
```

An example involving an eigenform rational over the base, but the base is not QQ.

```
sage: k.<a> = NumberField(x^2-5)
sage: M = ModularSymbols(23, base_ring=k, sign=1).cuspidal_submodule()
sage: D = M.decomposition(); D
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3␣
↪for Gamma_0(23) of weight 2 with sign 1 over Number Field in a with defining␣
↪polynomial x^2 - 5,
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3␣
↪for Gamma_0(23) of weight 2 with sign 1 over Number Field in a with defining␣
↪polynomial x^2 - 5
```

(continues on next page)

(continued from previous page)

```
]
sage: M.q_expansion_module(8, QQ)
Vector space of degree 8 and dimension 2 over Rational Field
Basis matrix:
[ 0  1  0 -1 -1  0 -2  2]
[ 0  0  1 -2 -1  2  1  2]
```

An example involving an eigenform not rational over the base and for which the base is not QQ.

```
sage: eps = DirichletGroup(25).0^2
sage: M = ModularSymbols(eps,2,sign=1).cuspidal_submodule(); M
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4␣
↪and level 25, weight 2, character [zeta10], sign 1, over Cyclotomic Field of␣
↪order 10 and degree 4
sage: D = M.decomposition(); D
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4␣
↪and level 25, weight 2, character [zeta10], sign 1, over Cyclotomic Field of␣
↪order 10 and degree 4
]
sage: D[0].q_eigenform(4, 'mu')
q + mu*q^2 + ((zeta10^3 + zeta10 - 1)*mu + zeta10^2 - 1)*q^3 + O(q^4)
sage: D[0].q_expansion_module(11, QQ)
Vector space of degree 11 and dimension 8 over Rational Field
Basis matrix:
[  0   1   0   0   0   0   0   0 -20  -3   0]
[  0   0   1   0   0   0   0   0 -16  -1   0]
[  0   0   0   1   0   0   0   0 -11  -2   0]
[  0   0   0   0   1   0   0   0  -8  -1   0]
[  0   0   0   0   0   1   0   0  -5  -1   0]
[  0   0   0   0   0   0   1   0  -3  -1   0]
[  0   0   0   0   0   0   0   1  -2   0   0]
[  0   0   0   0   0   0   0   0   0   0   1]
sage: D[0].q_expansion_module(11)
Vector space of degree 11 and dimension 2 over Cyclotomic Field of order 10 and␣
↪degree 4
Basis matrix:
[                                  0                                    1      ␣
↪                            0                        zeta10^2 - 1            ␣
↪          -zeta10^2 - 1               -zeta10^3 - zeta10^2                   ␣
↪zeta10^2 - zeta10           2*zeta10^3 + 2*zeta10 - 1    zeta10^3 - zeta10^2 -
↪ zeta10 + 1         zeta10^3 - zeta10^2 + zeta10   -2*zeta10^3 + 2*zeta10^2 -␣
↪zeta10]
[                                  0                                    0      ␣
↪                            1              zeta10^3 + zeta10 - 1              ␣
↪            -zeta10 - 1               -zeta10^3 - zeta10^2 -2*zeta10^3 +␣
↪zeta10^2 - zeta10 + 1                              zeta10^2                   ␣
↪              0                    zeta10^3 + 1  2*zeta10^3 - zeta10^2 +␣
↪zeta10 - 1]
```

EXAMPLES WITH SIGN 0 and R=QQ:

Todo: This doesn't work yet as it's not implemented!!

```
sage: M = ModularSymbols(11,2).cuspidal_submodule() #not tested
sage: M.q_expansion_module() #not tested
... boom ...
```

EXAMPLES WITH SIGN 1 and R=ZZ (computes saturation):

```
sage: M = ModularSymbols(43,2, sign=1).cuspidal_submodule()
sage: M.q_expansion_module(8, QQ)
Vector space of degree 8 and dimension 3 over Rational Field
Basis matrix:
[    0     1     0     0     0     2    -2    -2]
[    0     0     1     0  -1/2     1  -3/2     0]
[    0     0     0     1  -1/2     2  -3/2    -1]
sage: M.q_expansion_module(8, ZZ)
Free module of degree 8 and rank 3 over Integer Ring
Echelon basis matrix:
[ 0  1  0  0  0  2 -2 -2]
[ 0  0  1  1 -1  3 -3 -1]
[ 0  0  0  2 -1  4 -3 -2]
```

rational_period_mapping()

Return the rational period mapping associated to `self`.

This is a homomorphism to a vector space whose kernel is the same as the kernel of the period mapping associated to `self`. For this to exist, self must be Hecke equivariant.

Use *integral_period_mapping()* to obtain a homomorphism to a $\mathbf{Z}$-module, normalized so the image of integral modular symbols is exactly $\mathbf{Z}^n$.

EXAMPLES:

```
sage: M = ModularSymbols(37)
sage: A = M[1]; A
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5␣
↪for Gamma_0(37) of weight 2 with sign 0 over Rational Field
sage: r = A.rational_period_mapping(); r
Rational period mapping associated to Modular Symbols subspace of dimension 2␣
↪of Modular Symbols space of dimension 5 for Gamma_0(37) of weight 2 with sign␣
↪0 over Rational Field
sage: r(M.0)
(0, 0)
sage: r(M.1)
(1, 0)
sage: r.matrix()
[ 0  0]
[ 1  0]
[ 0  1]
[-1 -1]
[ 0  0]
sage: r.domain()
Modular Symbols space of dimension 5 for Gamma_0(37) of weight 2 with sign 0␣
↪over Rational Field
```

(continues on next page)

(continued from previous page)

```
sage: r.codomain()
Vector space of degree 2 and dimension 2 over Rational Field
Basis matrix:
[1 0]
[0 1]
```

set_default_prec(*prec*)

Set the default precision for computation of q-expansion associated to the ambient space of this space of modular symbols (and all subspaces).

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(13),2)
sage: M.set_default_prec(5)
sage: M.cuspidal_submodule().q_expansion_basis()
[
q - 4*q^3 - q^4 + O(q^5),
q^2 - 2*q^3 - q^4 + O(q^5)
]
```

set_precision(*prec*)

Same as self.set_default_prec(prec).

EXAMPLES:

```
sage: M = ModularSymbols(17,2)
sage: M.cuspidal_submodule().q_expansion_basis()
[
q - q^2 - q^4 - 2*q^5 + 4*q^7 + O(q^8)
]
sage: M.set_precision(10)
sage: M.cuspidal_submodule().q_expansion_basis()
[
q - q^2 - q^4 - 2*q^5 + 4*q^7 + 3*q^8 - 3*q^9 + O(q^10)
]
```

sign()

Return the sign of `self`.

For efficiency reasons, it is often useful to compute in the (largest) quotient of modular symbols where the * involution acts as +1, or where it acts as -1.

INPUT:

- `ModularSymbols self` - arbitrary space of modular symbols.

OUTPUT:

- `int` - the sign of `self`, either -1, 0, or 1.
- `-1` - if this is factor of quotient where * acts as -1,
- `+1` - if this is factor of quotient where * acts as +1,
- `0` - if this is full space of modular symbols (no quotient).

EXAMPLES:

```
sage: m = ModularSymbols(33)
sage: m.rank()
9
sage: m.sign()
0
sage: m = ModularSymbols(33, sign=0)
sage: m.sign()
0
sage: m.rank()
9
sage: m = ModularSymbols(33, sign=-1)
sage: m.sign()
-1
sage: m.rank()
3
```

sign_submodule(*sign*, *compute_dual=True*)

Return the subspace of `self` that is fixed under the star involution.

INPUT:

- `sign` - int (either -1, 0 or +1)
- `compute_dual` - bool (default: `True`) also compute dual subspace. This are useful for many algorithms.

OUTPUT: subspace of modular symbols

EXAMPLES:

```
sage: M = ModularSymbols(29,2)
sage: M.sign_submodule(1)
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 5␣
↪for Gamma_0(29) of weight 2 with sign 0 over Rational Field
sage: M.sign_submodule(-1)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5␣
↪for Gamma_0(29) of weight 2 with sign 0 over Rational Field
sage: M.sign_submodule(-1).sign()
-1
```

simple_factors()

Return a list modular symbols spaces S where S is simple spaces of modular symbols (for the anemic Hecke algebra) and `self` is isomorphic to the direct sum of the S with some multiplicities, as a module over the *anemic* Hecke algebra.

For the multiplicities use factorization() instead.

ASSUMPTION: self is a module over the anemic Hecke algebra.

EXAMPLES:

```
sage: ModularSymbols(1,100,sign=-1).simple_factors()
[Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension␣
↪8 for Gamma_0(1) of weight 100 with sign -1 over Rational Field]
sage: ModularSymbols(1,16,0,GF(5)).simple_factors()
[Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(1) of weight 16 with sign 0 over Finite Field of size 5,
```

(continues on next page)

(continued from previous page)

```
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3␣
↪for Gamma_0(1) of weight 16 with sign 0 over Finite Field of size 5,
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3␣
↪for Gamma_0(1) of weight 16 with sign 0 over Finite Field of size 5]
```

star_decomposition()

Decompose `self` into subspaces which are eigenspaces for the star involution.

EXAMPLES:

```
sage: ModularSymbols(Gamma1(19), 2).cuspidal_submodule().star_decomposition()
[
Modular Symbols subspace of dimension 7 of Modular Symbols space of dimension␣
↪31 for Gamma_1(19) of weight 2 with sign 0 over Rational Field,
Modular Symbols subspace of dimension 7 of Modular Symbols space of dimension␣
↪31 for Gamma_1(19) of weight 2 with sign 0 over Rational Field
]
```

star_eigenvalues()

Return the eigenvalues of the star involution acting on `self`.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: D = M.decomposition()
sage: M.star_eigenvalues()
[1, -1]
sage: D[0].star_eigenvalues()
[1]
sage: D[1].star_eigenvalues()
[1, -1]
sage: D[1].plus_submodule().star_eigenvalues()
[1]
sage: D[1].minus_submodule().star_eigenvalues()
[-1]
```

star_involution()

Return the star involution on `self`, which is induced by complex conjugation on modular symbols.

Not implemented in this abstract base class.

EXAMPLES:

```
sage: M = ModularSymbols(11, 2); sage.modular.modsym.space.ModularSymbolsSpace.
↪star_involution(M)
Traceback (most recent call last):
...
NotImplementedError
```

sturm_bound()

Return the Sturm bound for this space of modular symbols.

Type `sturm_bound?` for more details.

EXAMPLES:

```
sage: ModularSymbols(11,2).sturm_bound()
2
sage: ModularSymbols(389,2).sturm_bound()
65
sage: ModularSymbols(1,12).sturm_bound()
1
sage: ModularSymbols(1,36).sturm_bound()
3
sage: ModularSymbols(DirichletGroup(31).0^2).sturm_bound()
6
sage: ModularSymbols(Gamma1(31)).sturm_bound()
160
```

class sage.modular.modsym.space.**PeriodMapping**(*modsym*, *A*)

Bases: sage.structure.sage_object.SageObject

Base class for representing a period mapping attached to a space of modular symbols.

To be used via the derived classes *RationalPeriodMapping* and *IntegralPeriodMapping*.

codomain()

Return the codomain of this mapping.

EXAMPLES:

Note that this presently returns the wrong answer, as a consequence of various bugs in the free module routines:

```
sage: ModularSymbols(11, 2).cuspidal_submodule().integral_period_mapping().
↪codomain()
Vector space of degree 2 and dimension 2 over Rational Field
Basis matrix:
[1 0]
[0 1]
```

domain()

Return the domain of this mapping (which is the ambient space of the corresponding modular symbols space).

EXAMPLES:

```
sage: ModularSymbols(17, 2).cuspidal_submodule().integral_period_mapping().
↪domain()
Modular Symbols space of dimension 3 for Gamma_0(17) of weight 2 with sign 0␣
↪over Rational Field
```

matrix()

Return the matrix of this period mapping.

EXAMPLES:

```
sage: ModularSymbols(11, 2).cuspidal_submodule().integral_period_mapping().
↪matrix()
[  0 1/5]
[  1   0]
[  0   1]
```

modular_symbols_space()

Return the space of modular symbols to which this period mapping corresponds.

EXAMPLES:

```
sage: ModularSymbols(17, 2).rational_period_mapping().modular_symbols_space()
Modular Symbols space of dimension 3 for Gamma_0(17) of weight 2 with sign 0␣
↪over Rational Field
```

class sage.modular.modsym.space.**RationalPeriodMapping**(*modsym*, *A*)

Bases: *sage.modular.modsym.space.PeriodMapping*

sage.modular.modsym.space.**is_ModularSymbolsSpace**(*x*)

Return True if x is a space of modular symbols.

EXAMPLES:

```
sage: M = ModularForms(3, 2)
sage: sage.modular.modsym.space.is_ModularSymbolsSpace(M)
False
sage: sage.modular.modsym.space.is_ModularSymbolsSpace(M.modular_symbols(sign=1))
True
```

CHAPTER

THREE

AMBIENT SPACES OF MODULAR SYMBOLS

This module defines the following classes. There is an abstract base class `ModularSymbolsAmbient`, derived from `space.ModularSymbolsSpace` and `hecke.AmbientHeckeModule`. As this is an abstract base class, only derived classes should be instantiated. There are five derived classes:

- `ModularSymbolsAmbient_wtk_g0`, for modular symbols of general weight k for $\Gamma_0(N)$;
- `ModularSymbolsAmbient_wt2_g0` (derived from `ModularSymbolsAmbient_wtk_g0`), for modular symbols of weight 2 for $\Gamma_0(N)$;
- `ModularSymbolsAmbient_wtk_g1`, for modular symbols of general weight k for $\Gamma_1(N)$;
- `ModularSymbolsAmbient_wtk_gamma_h`, for modular symbols of general weight k for Γ_H, where H is a subgroup of $\mathbf{Z}/N\mathbf{Z}$;
- `ModularSymbolsAmbient_wtk_eps`, for modular symbols of general weight k and character ϵ.

EXAMPLES:

We compute a space of modular symbols modulo 2. The dimension is different from that of the corresponding space in characteristic 0:

```
sage: M = ModularSymbols(11,4,base_ring=GF(2)); M
Modular Symbols space of dimension 7 for Gamma_0(11) of weight 4
with sign 0 over Finite Field of size 2
sage: M.basis()
([X*Y,(1,0)], [X*Y,(1,8)], [X*Y,(1,9)], [X^2,(0,1)], [X^2,(1,8)], [X^2,(1,9)], [X^2,(1,
↪10)])
sage: M0 =  ModularSymbols(11,4,base_ring=QQ); M0
Modular Symbols space of dimension 6 for Gamma_0(11) of weight 4
with sign 0 over Rational Field
sage: M0.basis()
([X^2,(0,1)], [X^2,(1,6)], [X^2,(1,7)], [X^2,(1,8)], [X^2,(1,9)], [X^2,(1,10)])
```

The characteristic polynomial of the Hecke operator T_2 has an extra factor x.

```
sage: M.T(2).matrix().fcp('x')
(x + 1)^2 * x^5
sage: M0.T(2).matrix().fcp('x')
(x - 9)^2 * (x^2 - 2*x - 2)^2
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient**(*group*, *weight*, *sign*, *base_ring*, *character=None*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.space.ModularSymbolsSpace*, sage.modular.hecke.ambient_module.AmbientHeckeModule

An ambient space of modular symbols for a congruence subgroup of $SL_2(\mathbf{Z})$.

This class is an abstract base class, so only derived classes should be instantiated.

INPUT:

- `weight` - an integer
- `group` - a congruence subgroup.
- `sign` - an integer, either -1, 0, or 1
- `base_ring` - a commutative ring
- `custom_init` - a function that is called with self as input before any computations are done using self; this could be used to set a custom modular symbols presentation.

boundary_map()

Return the boundary map to the corresponding space of boundary modular symbols.

EXAMPLES:

```
sage: ModularSymbols(20,2).boundary_map()
Hecke module morphism boundary map defined by the matrix
[ 1 -1  0  0  0  0]
[ 0  1 -1  0  0  0]
[ 0  1  0 -1  0  0]
[ 0  0  0 -1  1  0]
[ 0  1  0 -1  0  0]
[ 0  0  1 -1  0  0]
[ 0  1  0  0  0 -1]
Domain: Modular Symbols space of dimension 7 for Gamma_0(20) of weight ...
Codomain: Space of Boundary Modular Symbols for Congruence Subgroup Gamma0(20) .
↪..
sage: type(ModularSymbols(20,2).boundary_map())
<class 'sage.modular.hecke.morphism.HeckeModuleMorphism_matrix'>
```

boundary_space()

Return the subspace of boundary modular symbols of this modular symbols ambient space.

EXAMPLES:

```
sage: M = ModularSymbols(20, 2)
sage: B = M.boundary_space(); B
Space of Boundary Modular Symbols for Congruence Subgroup Gamma0(20) of weight␣
↪2 over Rational Field
sage: M.cusps()
[Infinity, 0, -1/4, 1/5, -1/2, 1/10]
sage: M.dimension()
7
sage: B.dimension()
6
```

change_ring(*R*)

Change the base ring to R.

EXAMPLES:

```
sage: ModularSymbols(Gamma1(13), 2).change_ring(GF(17))
Modular Symbols space of dimension 15 for Gamma_1(13) of weight 2 with sign 0␣
↪over Finite Field of size 17
sage: M = ModularSymbols(DirichletGroup(5).0, 7); MM=M.change_
↪ring(CyclotomicField(8)); MM
Modular Symbols space of dimension 6 and level 5, weight 7, character [zeta8^2],
↪ sign 0, over Cyclotomic Field of order 8 and degree 4
sage: MM.change_ring(CyclotomicField(4)) == M
True
sage: M.change_ring(QQ)
Traceback (most recent call last):
...
TypeError: Unable to coerce zeta4 to a rational
```

Similarly with base_extend():

```
sage: M = ModularSymbols(DirichletGroup(5).0, 7); MM = M.base_
↪extend(CyclotomicField(8)); MM
Modular Symbols space of dimension 6 and level 5, weight 7, character [zeta8^2],
↪ sign 0, over Cyclotomic Field of order 8 and degree 4
sage: MM.base_extend(CyclotomicField(4))
Traceback (most recent call last):
...
TypeError: Base extension of self (over 'Cyclotomic Field of order 8 and degree␣
↪4') to ring 'Cyclotomic Field of order 4 and degree 2' not defined.
```

compact_newform_eigenvalues(*v*, *names='alpha'*)

Return compact systems of eigenvalues for each Galois conjugacy class of cuspidal newforms in this ambient space.

INPUT:

- v - list of positive integers

OUTPUT:

- list - of pairs (E, x), where E*x is a vector with entries the eigenvalues a_n for $n \in v$.

EXAMPLES:

```
sage: M = ModularSymbols(43,2,1)
sage: X = M.compact_newform_eigenvalues(prime_range(10))
sage: X[0][0] * X[0][1]
(-2, -2, -4, 0)
sage: X[1][0] * X[1][1]
(alpha1, -alpha1, -alpha1 + 2, alpha1 - 2)
```

```
sage: M = ModularSymbols(DirichletGroup(24,QQ).1,2,sign=1)
sage: M.compact_newform_eigenvalues(prime_range(10),'a')
[(
[-1/2 -1/2]
[ 1/2 -1/2]
[  -1    1]
[  -2    0], (1, -2*a0 - 1)
)]
```

(continues on next page)

(continued from previous page)

```
sage: a = M.compact_newform_eigenvalues([1..10],'a')[0]
sage: a[0]*a[1]
(1, a0, a0 + 1, -2*a0 - 2, -2*a0 - 2, -a0 - 2, -2, 2*a0 + 4, -1, 2*a0 + 4)
sage: M = ModularSymbols(DirichletGroup(13).0^2,2,sign=1)
sage: M.compact_newform_eigenvalues(prime_range(10),'a')
[(
[  -zeta6 - 1]
[ 2*zeta6 - 2]
[-2*zeta6 + 1]
[           0], (1)
)]
sage: a = M.compact_newform_eigenvalues([1..10],'a')[0]
sage: a[0]*a[1]
(1, -zeta6 - 1, 2*zeta6 - 2, zeta6, -2*zeta6 + 1, -2*zeta6 + 4, 0, 2*zeta6 - 1,
↪-zeta6, 3*zeta6 - 3)
```

compute_presentation()

Compute and cache the presentation of this space.

EXAMPLES:

```
sage: ModularSymbols(11,2).compute_presentation() # no output
```

cuspidal_submodule()

The cuspidal submodule of this modular symbols ambient space.

EXAMPLES:

```
sage: M = ModularSymbols(12,2,0,GF(5)) ; M
Modular Symbols space of dimension 5 for Gamma_0(12) of weight 2 with sign 0
↪over Finite Field of size 5
sage: M.cuspidal_submodule()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 5
↪for Gamma_0(12) of weight 2 with sign 0 over Finite Field of size 5
sage: ModularSymbols(1,24,-1).cuspidal_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 2
↪for Gamma_0(1) of weight 24 with sign -1 over Rational Field
```

The cuspidal submodule of the cuspidal submodule is itself:

```
sage: M = ModularSymbols(389)
sage: S = M.cuspidal_submodule()
sage: S.cuspidal_submodule() is S
True
```

cusps()

Return the set of cusps for this modular symbols space.

EXAMPLES:

```
sage: ModularSymbols(20,2).cusps()
[Infinity, 0, -1/4, 1/5, -1/2, 1/10]
```

dual_star_involution_matrix()

Return the matrix of the dual star involution, which is induced by complex conjugation on the linear dual of modular symbols.

EXAMPLES:

```
sage: ModularSymbols(20,2).dual_star_involution_matrix()
[1 0 0 0 0 0 0]
[0 1 0 0 0 0 0]
[0 0 0 0 1 0 0]
[0 0 0 1 0 0 0]
[0 0 1 0 0 0 0]
[0 0 0 0 0 1 0]
[0 0 0 0 0 0 1]
```

eisenstein_submodule()

Return the Eisenstein submodule of this space of modular symbols.

EXAMPLES:

```
sage: ModularSymbols(20,2).eisenstein_submodule()
Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension 7␣
↪for Gamma_0(20) of weight 2 with sign 0 over Rational Field
```

element(x)

Creates and returns an element of self from a modular symbol, if possible.

INPUT:

- `x` - an object of one of the following types: ModularSymbol, ManinSymbol.

OUTPUT:

ModularSymbol - a modular symbol with parent self.

EXAMPLES:

```
sage: M = ModularSymbols(11,4,1)
sage: M.T(3)
Hecke operator T_3 on Modular Symbols space of dimension 4 for Gamma_0(11) of␣
↪weight 4 with sign 1 over Rational Field
sage: M.T(3)(M.0)
28*[X^2,(0,1)] + 2*[X^2,(1,4)] + 2/3*[X^2,(1,6)] - 8/3*[X^2,(1,9)]
sage: M.T(3)(M.0).element()
(28, 2, 2/3, -8/3)
```

factor()

Return a list of pairs (S, e) where S is spaces of modular symbols and self is isomorphic to the direct sum of the S^e as a module over the *anemic* Hecke algebra adjoin the star involution. The cuspidal S are all simple, but the Eisenstein factors need not be simple.

EXAMPLES:

```
sage: ModularSymbols(Gamma0(22), 2).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field)^2 *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field)^2 *
(Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension␣
↪7 for Gamma_0(22) of weight 2 with sign 0 over Rational Field)
```

```
sage: ModularSymbols(1,6,0,GF(2)).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪2 for Gamma_0(1) of weight 6 with sign 0 over Finite Field of size 2) *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪2 for Gamma_0(1) of weight 6 with sign 0 over Finite Field of size 2)
```

```
sage: ModularSymbols(18,2).factorization()
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 for Gamma_0(18) of weight 2 with sign 0 over Rational Field) *
(Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension␣
↪7 for Gamma_0(18) of weight 2 with sign 0 over Rational Field)
```

```
sage: M = ModularSymbols(DirichletGroup(38,CyclotomicField(3)).0^2,  2, +1); M
Modular Symbols space of dimension 7 and level 38, weight 2, character [zeta3],␣
↪sign 1, over Cyclotomic Field of order 3 and degree 2
sage: M.factorization()                    # long time (about 8 seconds)
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2)
```

factorization()

Return a list of pairs (S, e) where S is spaces of modular symbols and self is isomorphic to the direct sum of the S^e as a module over the *anemic* Hecke algebra adjoin the star involution. The cuspidal S are all simple, but the Eisenstein factors need not be simple.

EXAMPLES:

```
sage: ModularSymbols(Gamma0(22), 2).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field)^2 *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field)^2 *
(Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension␣
↪7 for Gamma_0(22) of weight 2 with sign 0 over Rational Field)
```

```
sage: ModularSymbols(1,6,0,GF(2)).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪2 for Gamma_0(1) of weight 6 with sign 0 over Finite Field of size 2) *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪2 for Gamma_0(1) of weight 6 with sign 0 over Finite Field of size 2)
```

```
sage: ModularSymbols(18,2).factorization()
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 for Gamma_0(18) of weight 2 with sign 0 over Rational Field) *
```

(continues on next page)

(continued from previous page)

```
(Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension␣
↪7 for Gamma_0(18) of weight 2 with sign 0 over Rational Field)
```

```
sage: M = ModularSymbols(DirichletGroup(38,CyclotomicField(3)).0^2,  2, +1); M
Modular Symbols space of dimension 7 and level 38, weight 2, character [zeta3],␣
↪sign 1, over Cyclotomic Field of order 3 and degree 2
sage: M.factorization()                    # long time (about 8 seconds)
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2) *
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪7 and level 38, weight 2, character [zeta3], sign 1, over Cyclotomic Field of␣
↪order 3 and degree 2)
```

integral_structure(*algorithm='default'*)

Return the $\mathbf{Z}$-structure of this modular symbols space, generated by all integral modular symbols.

INPUT:

- `algorithm` - string (default: 'default' - choose heuristically)
  - `'pari'` - use pari for the HNF computation
  - `'padic'` - use p-adic algorithm (only good for dense case)

ALGORITHM: It suffices to consider lattice generated by the free generating symbols $X^iY^{k-2-i}.(u,v)$ after quotienting out by the S (and I) relations, since the quotient by these relations is the same over any ring.

EXAMPLES: In weight 2 the rational basis is often integral.

```
sage: M = ModularSymbols(11,2)
sage: M.integral_structure()
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[1 0 0]
[0 1 0]
[0 0 1]
```

This is rarely the case in higher weight:

```
sage: M = ModularSymbols(6,4)
sage: M.integral_structure()
Free module of degree 6 and rank 6 over Integer Ring
Echelon basis matrix:
[  1   0   0   0   0   0]
[  0   1   0   0   0   0]
[  0   0 1/2 1/2 1/2 1/2]
[  0   0   0   1   0   0]
```

(continues on next page)

(continued from previous page)

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

Here is an example involving $\Gamma_1(N)$.

```
sage: M = ModularSymbols(Gamma1(5),6)
sage: M.integral_structure()
Free module of degree 10 and rank 10 over Integer Ring
Echelon basis matrix:
[    1     0     0     0     0     0     0     0     0     0]
[    0     1     0     0     0     0     0     0     0     0]
[    0     0  1/96  1/32 23/24     0  1/96     0  7/24 67/96]
[    0     0     0  1/24 23/24     0     0  1/24   1/4 17/24]
[    0     0     0     0     1     0     0     0     0     0]
[    0     0     0     0     0   1/6     0  1/48 23/48   1/3]
[    0     0     0     0     0     0  1/24  1/24 11/24 11/24]
[    0     0     0     0     0     0     0  1/16  7/16   1/2]
[    0     0     0     0     0     0     0     0   1/2   1/2]
[    0     0     0     0     0     0     0     0     0     1]
```

is_cuspidal()

Return True if this space is cuspidal, else False.

EXAMPLES:

```
sage: M = ModularSymbols(20,2)
sage: M.is_cuspidal()
False
sage: S = M.cuspidal_subspace()
sage: S.is_cuspidal()
True
sage: S = M.eisenstein_subspace()
sage: S.is_cuspidal()
False
```

is_eisenstein()

Return True if this space is Eisenstein, else False.

EXAMPLES:

```
sage: M = ModularSymbols(20,2)
sage: M.is_eisenstein()
False
sage: S = M.eisenstein_submodule()
sage: S.is_eisenstein()
True
sage: S = M.cuspidal_subspace()
sage: S.is_eisenstein()
False
```

manin_basis()

Return a list of indices into the list of Manin generators (see `self.manin_generators()`) such that those symbols form a basis for the quotient of the $\mathbf{Q}$-vector space spanned by Manin symbols modulo the relations.

EXAMPLES:

```
sage: M = ModularSymbols(2,2)
sage: M.manin_basis()
[1]
sage: [M.manin_generators()[i] for i in M.manin_basis()]
[(1,0)]
sage: M = ModularSymbols(6,2)
sage: M.manin_basis()
[1, 10, 11]
sage: [M.manin_generators()[i] for i in M.manin_basis()]
[(1,0), (3,1), (3,2)]
```

manin_generators()

Return list of all Manin symbols for this space. These are the generators in the presentation of this space by Manin symbols.

EXAMPLES:

```
sage: M = ModularSymbols(2,2)
sage: M.manin_generators()
[(0,1), (1,0), (1,1)]
```

```
sage: M = ModularSymbols(1,6)
sage: M.manin_generators()
[[Y^4,(0,0)], [X*Y^3,(0,0)], [X^2*Y^2,(0,0)], [X^3*Y,(0,0)], [X^4,(0,0)]]
```

manin_gens_to_basis()

Return the matrix expressing the manin symbol generators in terms of the basis.

EXAMPLES:

```
sage: ModularSymbols(11,2).manin_gens_to_basis()
[-1  0  0]
[ 1  0  0]
[ 0  0  0]
[ 0  0  1]
[ 0 -1  1]
[ 0 -1  0]
[ 0  0 -1]
[ 0  0 -1]
[ 0  1 -1]
[ 0  1  0]
[ 0  0  1]
[ 0  0  0]
```

manin_symbol(*x*, *check=True*)

Construct a Manin Symbol from the given data.

INPUT:

- **x** (list) – either $[u, v]$ or $[i, u, v]$, where $0 \le i \le k - 2$ where k is the weight, and u,`v` are integers defining a valid element of $\mathbb{P}^1(N)$, where N is the level.

OUTPUT:

(ManinSymbol) the Manin Symbol associated to $[i; (u, v)]$, with $i = 0$ if not supplied, corresponding to the monomial symbol $[X^i * Y^{k-2-i}, (u, v)]$.

EXAMPLES:

```
sage: M = ModularSymbols(11,4,1)
sage: M.manin_symbol([2,5,6])
-2/3*[X^2,(1,6)] + 5/3*[X^2,(1,9)]
```

manin_symbols()

Return the list of Manin symbols for this modular symbols ambient space.

EXAMPLES:

```
sage: ModularSymbols(11,2).manin_symbols()
Manin Symbol List of weight 2 for Gamma0(11)
```

manin_symbols_basis()

A list of Manin symbols that form a basis for the ambient space `self`.

OUTPUT:

- `list` - a list of 2-tuples (if the weight is 2) or 3-tuples, which represent the Manin symbols basis for self.

EXAMPLES:

```
sage: m = ModularSymbols(23)
sage: m.manin_symbols_basis()
[(1,0), (1,17), (1,19), (1,20), (1,21)]
sage: m = ModularSymbols(6, weight=4, sign=-1)
sage: m.manin_symbols_basis()
[[X^2,(2,1)]]
```

modular_symbol(*x*, *check=True*)

Create a modular symbol in this space.

INPUT:

- `x` (list) – a list of either 2 or 3 entries:
  - 2 entries: $[\alpha, \beta]$ where α and β are cusps;
  - 3 entries: $[i, \alpha, \beta]$ where $0 \le i \le k-2$ and α and β are cusps;
- `check` (bool, default True) – flag that determines whether the input `x` needs processing: use check=False for efficiency if the input `x` is a list of length 3 whose first entry is an Integer, and whose second and third entries are Cusps (see examples).

OUTPUT:

(Modular Symbol) The modular symbol $Y^{k-2}\{\alpha, \beta\}$. or $X^i Y^{k-2-i}\{\alpha, \beta\}$.

EXAMPLES:

```
sage: set_modsym_print_mode('modular')
sage: M = ModularSymbols(11)
sage: M.modular_symbol([2/11, oo])
-{-1/9, 0}
sage: M.1
{-1/8, 0}
sage: M.modular_symbol([-1/8, 0])
{-1/8, 0}
```

(continues on next page)

(continued from previous page)

```
sage: M.modular_symbol([0, -1/8, 0])
{-1/8, 0}
sage: M.modular_symbol([10, -1/8, 0])
Traceback (most recent call last):
...
ValueError: The first entry of the tuple (=[10, -1/8, 0]) must be an integer␣
↪between 0 and k-2 (=0).
```

```
sage: N = ModularSymbols(6,4)
sage: set_modsym_print_mode('manin')
sage: N([1,Cusp(-1/4),Cusp(0)])
17/2*[X^2,(2,3)] - 9/2*[X^2,(2,5)] + 15/2*[X^2,(3,1)] - 15/2*[X^2,(3,2)]
sage: N([1,Cusp(-1/2),Cusp(0)])
1/2*[X^2,(2,3)] + 3/2*[X^2,(2,5)] + 3/2*[X^2,(3,1)] - 3/2*[X^2,(3,2)]
```

Use check=False for efficiency if the input x is a list of length 3 whose first entry is an Integer, and whose second and third entries are cusps:

```
sage: M.modular_symbol([0, Cusp(2/11), Cusp(oo)], check=False)
-(1,9)
```

```
sage: set_modsym_print_mode()   # return to default.
```

modular_symbol_sum(*x*, *check=True*)

Construct a modular symbol sum.

INPUT:

- `x` (list) – $[f, \alpha, \beta]$ where $f = \sum_{i=0}^{k-2} a_i X^i Y^{k-2-i}$ is a homogeneous polynomial over $\mathbf{Z}$ of degree k and α and β are cusps.
- `check` (bool, default True) – if True check the validity of the input tuple `x`

OUTPUT:

The sum $\sum_{i=0}^{k-2} a_i[i, \alpha, \beta]$ as an element of this modular symbol space.

EXAMPLES:

```
sage: M = ModularSymbols(11,4)
sage: R.<X,Y>=QQ[]
sage: M.modular_symbol_sum([X*Y,Cusp(0),Cusp(Infinity)])
-3/14*[X^2,(1,6)] + 1/14*[X^2,(1,7)] - 1/14*[X^2,(1,8)] + 1/2*[X^2,(1,9)] - 2/
↪7*[X^2,(1,10)]
```

modular_symbols_of_level(*G*)

Return a space of modular symbols with the same parameters as this space, except the congruence subgroup is changed to G.

INPUT:

- `G` – either a congruence subgroup or an integer to use as the level of such a group. The given group must either contain or be contained in the group defining `self`.

modular_symbols_of_sign(*sign*)

Return a space of modular symbols with the same defining properties (weight, level, etc.) as this space except with given sign.

INPUT:

- sign (int) – A sign ($+1$, -1 or 0).

OUTPUT:

(ModularSymbolsAmbient) A space of modular symbols with the same defining properties (weight, level, etc.) as this space except with given sign.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma0(11),2,sign=0)
sage: M
Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign 0␣
↪over Rational Field
sage: M.modular_symbols_of_sign(-1)
Modular Symbols space of dimension 1 for Gamma_0(11) of weight 2 with sign -1␣
↪over Rational Field
sage: M = ModularSymbols(Gamma1(11),2,sign=0)
sage: M.modular_symbols_of_sign(-1)
Modular Symbols space of dimension 1 for Gamma_1(11) of weight 2 with sign -1␣
↪over Rational Field
```

modular_symbols_of_weight(*k*)

Return a space of modular symbols with the same defining properties (weight, sign, etc.) as this space except with weight k.

INPUT:

- k (int) – A positive integer.

OUTPUT:

(ModularSymbolsAmbient) A space of modular symbols with the same defining properties (level, sign) as this space except with given weight.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(6),2,sign=0)
sage: M.modular_symbols_of_weight(3)
Modular Symbols space of dimension 4 for Gamma_1(6) of weight 3 with sign 0␣
↪over Rational Field
```

new_submodule(*p=None*)

Return the new or p-new submodule of this modular symbols ambient space.

INPUT:

- p - (default: None); if not None, return only the p-new submodule.

OUTPUT:

The new or p-new submodule of this modular symbols ambient space.

EXAMPLES:

```
sage: ModularSymbols(100).new_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension␣
↪31 for Gamma_0(100) of weight 2 with sign 0 over Rational Field
sage: ModularSymbols(389).new_submodule()
Modular Symbols space of dimension 65 for Gamma_0(389) of weight 2 with sign 0␣
↪over Rational Field
```

(continues on next page)

(continued from previous page)

```
```

p1list()

Return a P1list of the level of this modular symbol space.

EXAMPLES:

```
sage: ModularSymbols(11,2).p1list()
The projective line over the integers modulo 11
```

rank()

Return the rank of this modular symbols ambient space.

OUTPUT:

(int) The rank of this space of modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(389)
sage: M.rank()
65
```

```
sage: ModularSymbols(11,sign=0).rank()
3
sage: ModularSymbols(100,sign=0).rank()
31
sage: ModularSymbols(22,sign=1).rank()
5
sage: ModularSymbols(1,12).rank()
3
sage: ModularSymbols(3,4).rank()
2
sage: ModularSymbols(8,6,sign=-1).rank()
3
```

star_involution()

Return the star involution on this modular symbols space.

OUTPUT:

(matrix) The matrix of the star involution on this space, which is induced by complex conjugation on modular symbols, with respect to the standard basis.

EXAMPLES:

```
sage: ModularSymbols(20,2).star_involution()
Hecke module morphism Star involution on Modular Symbols space of dimension 7␣
↪for Gamma_0(20) of weight 2 with sign 0 over Rational Field defined by the␣
↪matrix
[1 0 0 0 0 0 0]
[0 1 0 0 0 0 0]
[0 0 0 0 1 0 0]
[0 0 0 1 0 0 0]
[0 0 1 0 0 0 0]
[0 0 0 0 0 1 0]
```

(continues on next page)

(continued from previous page)

```
[0 0 0 0 0 0 1]
Domain: Modular Symbols space of dimension 7 for Gamma_0(20) of weight ...
Codomain: Modular Symbols space of dimension 7 for Gamma_0(20) of weight ...
```

submodule(*M*, *dual_free_module=None*, *check=True*)

Return the submodule with given generators or free module M.

INPUT:

- M - either a submodule of this ambient free module, or generators for a submodule;
- **dual_free_module (bool, default None) – this may be** useful to speed up certain calculations; it is the corresponding submodule of the ambient dual module;
- **check (bool, default True) – if True, check that** M **is** a submodule, i.e. is invariant under all Hecke operators.

OUTPUT:

A subspace of this modular symbol space.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: M.submodule([M.0])
Traceback (most recent call last):
...
ValueError: The submodule must be invariant under all Hecke operators.
sage: M.eisenstein_submodule().basis()
((1,0) - 1/5*(1,9),)
sage: M.basis()
((1,0), (1,8), (1,9))
sage: M.submodule([M.0 - 1/5*M.2])
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3␣
↪for Gamma_0(11) of weight 2 with sign 0 over Rational Field
```

Note: It would make more sense to only check that M is invariant under the Hecke operators with index coprime to the level. Unfortunately, I do not know a reasonable algorithm for determining whether a module is invariant under just the anemic Hecke algebra, since I do not know an analogue of the Sturm bound for the anemic Hecke algebra. - William Stein, 2007-07-27

twisted_winding_element(*i*, *eps*)

Return the twisted winding element of given degree and character.

INPUT:

- i (int) – an integer, $0 \leq i \leq k-2$ where k is the weight.
- eps (character) – a Dirichlet character

OUTPUT:

(modular symbol) The so-called 'twisted winding element':

$$\sum_{a \in (\mathbf{Z}/m\mathbf{Z})^\times} \varepsilon(a) * [i, 0, a/m].$$

Note: This will only work if the base ring of the modular symbol space contains the character values.

EXAMPLES:

```
sage: eps = DirichletGroup(5)[2]
sage: K = eps.base_ring()
sage: M = ModularSymbols(37,2,0,K)
sage: M.twisted_winding_element(0,eps)
2*(1,23) - 2*(1,32) + 2*(1,34)
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient_wt2_g0**(*N*, *sign*, *F*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g0*

Modular symbols for $\Gamma_0(N)$ of integer weight 2 over the field F.

INPUT:

- N - int, the level
- sign - int, either -1, 0, or 1

OUTPUT:

The space of modular symbols of weight 2, trivial character, level N and given sign.

EXAMPLES:

```
sage: ModularSymbols(Gamma0(12),2)
Modular Symbols space of dimension 5 for Gamma_0(12) of weight 2 with sign 0 over␣
↪Rational Field
```

boundary_space()

Return the space of boundary modular symbols for this space.

EXAMPLES:

```
sage: M = ModularSymbols(100,2)
sage: M.boundary_space()
Space of Boundary Modular Symbols for Congruence Subgroup Gamma0(100) of weight␣
↪2 over Rational Field
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient_wtk_eps**(*eps*, *weight*, *sign*, *base_ring*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.ambient.ModularSymbolsAmbient*

Space of modular symbols with given weight, character, base ring and sign.

INPUT:

- eps - dirichlet.DirichletCharacter, the "Nebentypus" character.
- weight - int, the weight = 2
- sign - int, either -1, 0, or 1
- base_ring - the base ring. It must be possible to change the ring of the character to this base ring (not always canonically).

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: eps.order()
2
sage: ModularSymbols(eps, 2)
Modular Symbols space of dimension 0 and level 4, weight 2, character [-1], sign 0,␣
↪over Rational Field
sage: ModularSymbols(eps, 3)
Modular Symbols space of dimension 2 and level 4, weight 3, character [-1], sign 0,␣
↪over Rational Field
```

We next create a space with character of order bigger than 2.

```
sage: eps = DirichletGroup(5).gen(0)
sage: eps     # has order 4
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> zeta4
sage: ModularSymbols(eps, 2).dimension()
0
sage: ModularSymbols(eps, 3).dimension()
2
```

Here is another example:

```
sage: G.<e> = DirichletGroup(5)
sage: M = ModularSymbols(e,3)
sage: loads(M.dumps()) == M
True
```

boundary_space()

Return the space of boundary modular symbols for this space.

EXAMPLES:

```
sage: eps = DirichletGroup(5).gen(0)
sage: M = ModularSymbols(eps, 2)
sage: M.boundary_space()
Boundary Modular Symbols space of level 5, weight 2, character [zeta4] and␣
↪dimension 0 over Cyclotomic Field of order 4 and degree 2
```

manin_symbols()

Return the Manin symbol list of this modular symbol space.

EXAMPLES:

```
sage: eps = DirichletGroup(5).gen(0)
sage: M = ModularSymbols(eps, 2)
sage: M.manin_symbols()
Manin Symbol List of weight 2 for Gamma1(5) with character [zeta4]
sage: len(M.manin_symbols())
6
```

modular_symbols_of_level(N)

Return a space of modular symbols with the same parameters as this space except with level N.

INPUT:

- `N` (int) – a positive integer.

OUTPUT:

(Modular Symbol space) A space of modular symbols with the same defining properties (weight, sign, etc.) as this space except with level N.

EXAMPLES:

```
sage: eps = DirichletGroup(5).gen(0)
sage: M = ModularSymbols(eps, 2); M
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4],␣
↪sign 0, over Cyclotomic Field of order 4 and degree 2
sage: M.modular_symbols_of_level(15)
Modular Symbols space of dimension 0 and level 15, weight 2, character [1,␣
↪zeta4], sign 0, over Cyclotomic Field of order 4 and degree 2
```

modular_symbols_of_sign(*sign*)

Return a space of modular symbols with the same defining properties (weight, level, etc.) as this space except with given sign.

INPUT:

- `sign` (int) – A sign ($+1$, -1 or 0).

OUTPUT:

(ModularSymbolsAmbient) A space of modular symbols with the same defining properties (weight, level, etc.) as this space except with given sign.

EXAMPLES:

```
sage: eps = DirichletGroup(5).gen(0)
sage: M = ModularSymbols(eps, 2); M
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4],␣
↪sign 0, over Cyclotomic Field of order 4 and degree 2
sage: M.modular_symbols_of_sign(0) == M
True
sage: M.modular_symbols_of_sign(+1)
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4],␣
↪sign 1, over Cyclotomic Field of order 4 and degree 2
sage: M.modular_symbols_of_sign(-1)
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4],␣
↪sign -1, over Cyclotomic Field of order 4 and degree 2
```

modular_symbols_of_weight(k)

Return a space of modular symbols with the same defining properties (weight, sign, etc.) as this space except with weight k.

INPUT:

- `k` (int) – A positive integer.

OUTPUT:

(ModularSymbolsAmbient) A space of modular symbols with the same defining properties (level, sign) as this space except with given weight.

EXAMPLES:

```
sage: eps = DirichletGroup(5).gen(0)
sage: M = ModularSymbols(eps, 2); M
```

(continues on next page)

(continued from previous page)

```
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4],␣
↪sign 0, over Cyclotomic Field of order 4 and degree 2
sage: M.modular_symbols_of_weight(3)
Modular Symbols space of dimension 2 and level 5, weight 3, character [zeta4],␣
↪sign 0, over Cyclotomic Field of order 4 and degree 2
sage: M.modular_symbols_of_weight(2) == M
True
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient_wtk_g0**(*N*, *k*, *sign*, *F*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.ambient.ModularSymbolsAmbient*

Modular symbols for $\Gamma_0(N)$ of integer weight $k > 2$ over the field F.

For weight 2, it is faster to use `ModularSymbols_wt2_g0`.

INPUT:

- `N` - int, the level
- `k` - integer weight = 2.
- `sign` - int, either -1, 0, or 1
- `F` - field

EXAMPLES:

```
sage: ModularSymbols(1,12)
Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with sign 0 over␣
↪Rational Field
sage: ModularSymbols(1,12, sign=1).dimension()
2
sage: ModularSymbols(15,4, sign=-1).dimension()
4
sage: ModularSymbols(6,6).dimension()
10
sage: ModularSymbols(36,4).dimension()
36
```

boundary_space()

Return the space of boundary modular symbols for this space.

EXAMPLES:

```
sage: M = ModularSymbols(100,2)
sage: M.boundary_space()
Space of Boundary Modular Symbols for Congruence Subgroup Gamma0(100) of weight␣
↪2 over Rational Field
```

manin_symbols()

Return the Manin symbol list of this modular symbol space.

EXAMPLES:

```
sage: M = ModularSymbols(100,4)
sage: M.manin_symbols()
```

(continues on next page)

(continued from previous page)

```
Manin Symbol List of weight 4 for Gamma0(100)
sage: len(M.manin_symbols())
540
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient_wtk_g1**(*level*, *weight*, *sign*, *F*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.ambient.ModularSymbolsAmbient*

INPUT:

- `level` - int, the level
- `weight` - int, the weight = 2
- `sign` - int, either -1, 0, or 1
- `F` - field

EXAMPLES:

```
sage: ModularSymbols(Gamma1(17),2)
Modular Symbols space of dimension 25 for Gamma_1(17) of weight 2 with sign 0 over
↪Rational Field
sage: [ModularSymbols(Gamma1(7),k).dimension() for k in [2,3,4,5]]
[5, 8, 12, 16]
```

```
sage: ModularSymbols(Gamma1(7),3)
Modular Symbols space of dimension 8 for Gamma_1(7) of weight 3 with sign 0 over
↪Rational Field
```

boundary_space()

Return the space of boundary modular symbols for this space.

EXAMPLES:

```
sage: M = ModularSymbols(100,2)
sage: M.boundary_space()
Space of Boundary Modular Symbols for Congruence Subgroup Gamma0(100) of weight
↪2 over Rational Field
```

manin_symbols()

Return the Manin symbol list of this modular symbol space.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(30),4)
sage: M.manin_symbols()
Manin Symbol List of weight 4 for Gamma1(30)
sage: len(M.manin_symbols())
1728
```

class sage.modular.modsym.ambient.**ModularSymbolsAmbient_wtk_gamma_h**(*group*, *weight*, *sign*, *F*, *custom_init=None*, *category=None*)

Bases: *sage.modular.modsym.ambient.ModularSymbolsAmbient*

Initialize a space of modular symbols for $\Gamma_H(N)$.

INPUT:

- group - a congruence subgroup $\Gamma_H(N)$.
- weight - int, the weight = 2
- sign - int, either -1, 0, or 1
- F - field

EXAMPLES:

```
sage: ModularSymbols(GammaH(15,[4]),2)
Modular Symbols space of dimension 9 for Congruence Subgroup Gamma_H(15) with H␣
↪generated by [4] of weight 2 with sign 0 over Rational Field
```

boundary_space()

Return the space of boundary modular symbols for this space.

EXAMPLES:

```
sage: M = ModularSymbols(GammaH(15,[4]),2)
sage: M.boundary_space()
Boundary Modular Symbols space for Congruence Subgroup Gamma_H(15) with H␣
↪generated by [4] of weight 2 over Rational Field
```

manin_symbols()

Return the Manin symbol list of this modular symbol space.

EXAMPLES:

```
sage: M = ModularSymbols(GammaH(15,[4]),2)
sage: M.manin_symbols()
Manin Symbol List of weight 2 for Congruence Subgroup Gamma_H(15) with H␣
↪generated by [4]
sage: len(M.manin_symbols())
96
```

CHAPTER

FOUR

SUBSPACE OF AMBIENT SPACES OF MODULAR SYMBOLS

class sage.modular.modsym.subspace.**ModularSymbolsSubspace**(*ambient_hecke_module*, *submodule*, *dual_free_module=None*, *check=False*)

Bases: *sage.modular.modsym.space.ModularSymbolsSpace*, sage.modular.hecke.submodule.HeckeSubmodule

Subspace of ambient space of modular symbols

boundary_map()

The boundary map to the corresponding space of boundary modular symbols. (This is the restriction of the map on the ambient space.)

EXAMPLES:

```
sage: M = ModularSymbols(1, 24, sign=1) ; M
Modular Symbols space of dimension 3 for Gamma_0(1) of weight 24 with sign 1␣
↪over Rational Field
sage: M.basis()
([X^18*Y^4,(0,0)], [X^20*Y^2,(0,0)], [X^22,(0,0)])
sage: M.cuspidal_submodule().basis()
([X^18*Y^4,(0,0)], [X^20*Y^2,(0,0)])
sage: M.eisenstein_submodule().basis()
([X^18*Y^4,(0,0)] + 166747/324330*[X^20*Y^2,(0,0)] + 236364091/6742820700*[X^22,
↪(0,0)],)
sage: M.boundary_map()
Hecke module morphism boundary map defined by the matrix
[ 0]
[ 0]
[-1]
Domain: Modular Symbols space of dimension 3 for Gamma_0(1) of weight ...
Codomain: Space of Boundary Modular Symbols for Modular Group SL(2,Z) ...
sage: M.cuspidal_subspace().boundary_map()
Hecke module morphism defined by the matrix
[0]
[0]
Domain: Modular Symbols subspace of dimension 2 of Modular Symbols space ...
Codomain: Space of Boundary Modular Symbols for Modular Group SL(2,Z) ...
sage: M.eisenstein_submodule().boundary_map()
Hecke module morphism defined by the matrix
[-236364091/6742820700]
Domain: Modular Symbols subspace of dimension 1 of Modular Symbols space ...
Codomain: Space of Boundary Modular Symbols for Modular Group SL(2,Z) ...
```

cuspidal_submodule()

Return the cuspidal subspace of this subspace of modular symbols.

EXAMPLES:

```
sage: S = ModularSymbols(42,4).cuspidal_submodule() ; S
Modular Symbols subspace of dimension 40 of Modular Symbols space of dimension␣
↪48 for Gamma_0(42) of weight 4 with sign 0 over Rational Field
sage: S.is_cuspidal()
True
sage: S.cuspidal_submodule()
Modular Symbols subspace of dimension 40 of Modular Symbols space of dimension␣
↪48 for Gamma_0(42) of weight 4 with sign 0 over Rational Field
```

The cuspidal submodule of the cuspidal submodule is just itself:

```
sage: S.cuspidal_submodule() is S
True
sage: S.cuspidal_submodule() == S
True
```

An example where we abuse the _set_is_cuspidal function:

```
sage: M = ModularSymbols(389)
sage: S = M.eisenstein_submodule()
sage: S._set_is_cuspidal(True)
sage: S.cuspidal_submodule()
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪65 for Gamma_0(389) of weight 2 with sign 0 over Rational Field
```

dual_star_involution_matrix()

Return the matrix of the dual star involution, which is induced by complex conjugation on the linear dual of modular symbols.

EXAMPLES:

```
sage: S = ModularSymbols(6,4) ; S.dual_star_involution_matrix()
[ 1  0  0  0  0  0]
[ 0  1  0  0  0  0]
[ 0 -2  1  2  0  0]
[ 0  2  0 -1  0  0]
[ 0 -2  0  2  1  0]
[ 0  2  0 -2  0  1]
sage: S.star_involution().matrix().transpose() == S.dual_star_involution_
↪matrix()
True
```

eisenstein_subspace()

Return the Eisenstein subspace of this space of modular symbols.

EXAMPLES:

```
sage: ModularSymbols(24,4).eisenstein_subspace()
Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension␣
↪24 for Gamma_0(24) of weight 4 with sign 0 over Rational Field
sage: ModularSymbols(20,2).cuspidal_subspace().eisenstein_subspace()
```

(continues on next page)

(continued from previous page)

```
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 7␣
↪for Gamma_0(20) of weight 2 with sign 0 over Rational Field
```

factorization()

Return a list of pairs (S, e) where S is simple spaces of modular symbols and self is isomorphic to the direct sum of the S^e as a module over the *anemic* Hecke algebra adjoin the star involution.

The cuspidal S are all simple, but the Eisenstein factors need not be simple.

The factors are sorted by dimension - don't depend on much more for now.

ASSUMPTION: self is a module over the anemic Hecke algebra.

EXAMPLES: Note that if the sign is 1 then the cuspidal factors occur twice, one with each star eigenvalue.

```
sage: M = ModularSymbols(11)
sage: D = M.factorization(); D
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field) *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field) *
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field)
sage: [A.T(2).matrix() for A, _ in D]
[[-2], [3], [-2]]
sage: [A.star_eigenvalues() for A, _ in D]
[[-1], [1], [1]]
```

In this example there is one old factor squared.

```
sage: M = ModularSymbols(22,sign=1)
sage: S = M.cuspidal_submodule()
sage: S.factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension␣
↪2 for Gamma_0(11) of weight 2 with sign 1 over Rational Field)^2
```

```
sage: M = ModularSymbols(Gamma0(22), 2, sign=1)
sage: M1 = M.decomposition()[1]
sage: M1.factorization()
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 5␣
↪for Gamma_0(22) of weight 2 with sign 1 over Rational Field
```

is_cuspidal()

Return True if self is cuspidal.

EXAMPLES:

```
sage: ModularSymbols(42,4).cuspidal_submodule().is_cuspidal()
True
sage: ModularSymbols(12,6).eisenstein_submodule().is_cuspidal()
False
```

is_eisenstein()

Return True if self is an Eisenstein subspace.

EXAMPLES:

```
sage: ModularSymbols(22,6).cuspidal_submodule().is_eisenstein()
False
sage: ModularSymbols(22,6).eisenstein_submodule().is_eisenstein()
True
```

star_involution()

Return the star involution on self, which is induced by complex conjugation on modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(1,24)
sage: M.star_involution()
Hecke module morphism Star involution on Modular Symbols space of dimension 5␣
↪for Gamma_0(1) of weight 24 with sign 0 over Rational Field defined by the␣
↪matrix
[ 1  0  0  0  0]
[ 0 -1  0  0  0]
[ 0  0  1  0  0]
[ 0  0  0 -1  0]
[ 0  0  0  0  1]
Domain: Modular Symbols space of dimension 5 for Gamma_0(1) of weight ...
Codomain: Modular Symbols space of dimension 5 for Gamma_0(1) of weight ...
sage: M.cuspidal_subspace().star_involution()
Hecke module morphism defined by the matrix
[ 1  0  0  0]
[ 0 -1  0  0]
[ 0  0  1  0]
[ 0  0  0 -1]
Domain: Modular Symbols subspace of dimension 4 of Modular Symbols space ...
Codomain: Modular Symbols subspace of dimension 4 of Modular Symbols space ...
sage: M.plus_submodule().star_involution()
Hecke module morphism defined by the matrix
[1 0 0]
[0 1 0]
[0 0 1]
Domain: Modular Symbols subspace of dimension 3 of Modular Symbols space ...
Codomain: Modular Symbols subspace of dimension 3 of Modular Symbols space ...
sage: M.minus_submodule().star_involution()
Hecke module morphism defined by the matrix
[-1  0]
[ 0 -1]
Domain: Modular Symbols subspace of dimension 2 of Modular Symbols space ...
Codomain: Modular Symbols subspace of dimension 2 of Modular Symbols space ...
```

CHAPTER

FIVE

A SINGLE ELEMENT OF AN AMBIENT SPACE OF MODULAR SYMBOLS

class sage.modular.modsym.element.**ModularSymbolsElement**(*parent*, *x*, *check=True*)

Bases: sage.modular.hecke.element.HeckeModuleElement

An element of a space of modular symbols.

list()

Return a list of the coordinates of self in terms of a basis for the ambient space.

EXAMPLES:

```
sage: ModularSymbols(37, 2).0.list()
[1, 0, 0, 0, 0]
```

manin_symbol_rep()

Return a representation of self as a formal sum of Manin symbols.

EXAMPLES:

```
sage: x = ModularSymbols(37, 4).0
sage: x.manin_symbol_rep()
[X^2,(0,1)]
```

The result is cached:

```
sage: x.manin_symbol_rep() is x.manin_symbol_rep()
True
```

modular_symbol_rep()

Return a representation of `self` as a formal sum of modular symbols.

EXAMPLES:

```
sage: x = ModularSymbols(37, 4).0
sage: x.modular_symbol_rep()
X^2*{0, Infinity}
```

The result is cached:

```
sage: x.modular_symbol_rep() is x.modular_symbol_rep()
True
```

sage.modular.modsym.element.**is_ModularSymbolsElement**(*x*)

Return True if x is an element of a modular symbols space.

EXAMPLES:

```
sage: sage.modular.modsym.element.is_ModularSymbolsElement(ModularSymbols(11, 2).0)
True
sage: sage.modular.modsym.element.is_ModularSymbolsElement(13)
False
```

sage.modular.modsym.element.**set_modsym_print_mode**(*mode='manin'*)

Set the mode for printing of elements of modular symbols spaces.

INPUT:

- `mode` - a string. The possibilities are as follows:
- `'manin'` - (the default) formal sums of Manin symbols [P(X,Y),(u,v)]
- `'modular'` - formal sums of Modular symbols P(X,Y)*alpha,beta, where alpha and beta are cusps
- `'vector'` - as vectors on the basis for the ambient space

OUTPUT: none

EXAMPLES:

```
sage: M = ModularSymbols(13, 8)
sage: x = M.0 + M.1 + M.14
sage: set_modsym_print_mode('manin'); x
[X^5*Y,(1,11)] + [X^5*Y,(1,12)] + [X^6,(1,11)]
sage: set_modsym_print_mode('modular'); x
1610510*X^6*{-1/11, 0} + 893101*X^5*Y*{-1/11, 0} + 206305*X^4*Y^2*{-1/11, 0} +
↪25410*X^3*Y^3*{-1/11, 0} + 1760*X^2*Y^4*{-1/11, 0} + 65*X*Y^5*{-1/11, 0} -
↪248832*X^6*{-1/12, 0} - 103680*X^5*Y*{-1/12, 0} - 17280*X^4*Y^2*{-1/12, 0} -
↪1440*X^3*Y^3*{-1/12, 0} - 60*X^2*Y^4*{-1/12, 0} - X*Y^5*{-1/12, 0} + Y^6*{-1/11,
↪0}
sage: set_modsym_print_mode('vector'); x
(1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
sage: set_modsym_print_mode()
```

CHAPTER SIX

MODULAR SYMBOLS {ALPHA, BETA}

The ModularSymbol class represents a single modular symbol $X^i Y^{k-2-i}\{\alpha, \beta\}$.

AUTHOR:

- William Stein (2005, 2009)

class sage.modular.modsym.modular_symbols.**ModularSymbol**(*space*, *i*, *alpha*, *beta*)

Bases: sage.structure.sage_object.SageObject

The modular symbol $X^i \cdot Y^{k-2-i} \cdot \{\alpha, \beta\}$.

alpha()

For a symbol of the form $X^i Y^{k-2-i}\{\alpha, \beta\}$, return α.

EXAMPLES:

```
sage: s = ModularSymbols(11,4).1.modular_symbol_rep()[0][1]; s
X^2*{-1/6, 0}
sage: s.alpha()
-1/6
sage: type(s.alpha())
<class 'sage.modular.cusps.Cusp'>
```

apply(g)

Act on this symbol by the element $g \in \mathrm{GL}_2(\mathbf{Q})$.

INPUT:

- g – a list [a,b,c,d], corresponding to the 2x2 matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{GL}_2(\mathbf{Q})$.

OUTPUT:

- FormalSum – a formal sum $\sum_i c_i x_i$, where c_i are scalars and x_i are ModularSymbol objects, such that the sum $\sum_i c_i x_i$ is the image of this symbol under the action of g. No reduction is performed modulo the relations that hold in self.space().

The action of g on symbols is by

$$P(X, Y)\{\alpha, \beta\} \mapsto P(dX - bY, -cx + aY)\{g(\alpha), g(\beta)\}.$$

Note that for us we have $P = X^i Y^{k-2-i}$, which simplifies computation of the polynomial part slightly.

EXAMPLES:

```
sage: s = ModularSymbols(11,2).1.modular_symbol_rep()[0][1]; s
{-1/8, 0}
sage: a = 1; b = 2; c = 3; d = 4; s.apply([a,b,c,d])
{15/29, 1/2}
sage: x = -1/8;  (a*x+b)/(c*x+d)
15/29
sage: x = 0;  (a*x+b)/(c*x+d)
1/2
sage: s = ModularSymbols(11,4).1.modular_symbol_rep()[0][1]; s
X^2*{-1/6, 0}
sage: s.apply([a,b,c,d])
16*X^2*{11/21, 1/2} - 16*X*Y*{11/21, 1/2} + 4*Y^2*{11/21, 1/2}
sage: P = s.polynomial_part()
sage: X, Y = P.parent().gens()
sage: P(d*X-b*Y, -c*X+a*Y)
16*X^2 - 16*X*Y + 4*Y^2
sage: x = -1/6; (a*x+b)/(c*x+d)
11/21
sage: x = 0; (a*x+b)/(c*x+d)
1/2
sage: type(s.apply([a,b,c,d]))
<class 'sage.structure.formal_sum.FormalSum'>
```

beta()

For a symbol of the form $X^i Y^{k-2-i}\{\alpha, \beta\}$, return β.

EXAMPLES:

```
sage: s = ModularSymbols(11,4).1.modular_symbol_rep()[0][1]; s
X^2*{-1/6, 0}
sage: s.beta()
0
sage: type(s.beta())
<class 'sage.modular.cusps.Cusp'>
```

i()

For a symbol of the form $X^i Y^{k-2-i}\{\alpha, \beta\}$, return i.

EXAMPLES:

```
sage: s = ModularSymbols(11).2.modular_symbol_rep()[0][1]
sage: s.i()
0
sage: s = ModularSymbols(1,28).0.modular_symbol_rep()[0][1]; s
X^22*Y^4*{0, Infinity}
sage: s.i()
22
```

manin_symbol_rep()

Return a representation of `self` as a formal sum of Manin symbols.

The result is not cached.

EXAMPLES:

```
sage: M = ModularSymbols(11,4)
sage: s = M.1.modular_symbol_rep()[0][1]; s
X^2*{-1/6, 0}
sage: s.manin_symbol_rep()
-2*[X*Y,(-1,0)] - [X^2,(-1,0)] - [Y^2,(1,1)] - [X^2,(-6,1)]
sage: M(s.manin_symbol_rep()) == M([2,-1/6,0])
True
```

polynomial_part()

Return the polynomial part of this symbol, i.e. for a symbol of the form $X^iY^{k-2-i}\{\alpha,\beta\}$, return X^iY^{k-2-i}.

EXAMPLES:

```
sage: s = ModularSymbols(11).2.modular_symbol_rep()[0][1]
sage: s.polynomial_part()
1
sage: s = ModularSymbols(1,28).0.modular_symbol_rep()[0][1]; s
X^22*Y^4*{0, Infinity}
sage: s.polynomial_part()
X^22*Y^4
```

space()

The list of Manin symbols to which this symbol belongs.

EXAMPLES:

```
sage: s = ModularSymbols(11).2.modular_symbol_rep()[0][1]
sage: s.space()
Manin Symbol List of weight 2 for Gamma0(11)
```

weight()

Return the weight of the modular symbols space to which this symbol belongs; i.e. for a symbol of the form $X^iY^{k-2-i}\{\alpha,\beta\}$, return k.

EXAMPLES:

```
sage: s = ModularSymbols(1,28).0.modular_symbol_rep()[0][1]
sage: s.weight()
28
```

CHAPTER

SEVEN

MANIN SYMBOLS

This module defines the class ManinSymbol. A Manin symbol of weight k, level N has the form $[P(X,Y),(u:v)]$ where $P(X,Y) \in \mathbf{Z}[X,Y]$ is homogeneous of weight $k-2$ and $(u:v) \in \mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$. The ManinSymbol class holds a "monomial Manin symbol" of the simpler form $[X^iY^{k-2-i},(u:v)]$, which is stored as a triple (i,u,v); the weight and level are obtained from the parent structure, which is a *sage.modular.modsym.manin_symbol_list.ManinSymbolList*.

Integer matrices $[a,b;c,d]$ act on Manin symbols on the right, sending $[P(X,Y),(u,v)]$ to $[P(aX+bY,cX+dY),(u,v)g]$. Diagonal matrices (with $b=c=0$, such as $I=[-1,0;0,1]$ and $J=[-1,0;0,-1]$) and anti-diagonal matrices (with $a=d=0$, such as $S=[0,-1;1,0]$) map monomial Manin symbols to monomial Manin symbols, up to a scalar factor. For general matrices (such as $T=[0,1,-1,-1]$ and $T^2=[-1,-1;0,1]$) the image of a monomial Manin symbol is expressed as a formal sum of monomial Manin symbols, with integer coefficients.

class sage.modular.modsym.manin_symbol.**ManinSymbol**

Bases: sage.structure.element.Element

A Manin symbol $[X^iY^{k-2-i},(u,v)]$.

INPUT:

- parent – *ManinSymbolList*
- t – a triple (i,u,v) of integers

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,2)
sage: s = ManinSymbol(m,(2,2,3)); s
(2,3)
sage: s == loads(dumps(s))
True
```

```
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3)); s
[X^2*Y^4,(2,3)]
```

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s.parent()
Manin Symbol List of weight 8 for Gamma0(5)
```

apply(a, b, c, d)

Return the image of self under the matrix $[a, b; c, d]$.

Not implemented for raw ManinSymbol objects, only for members of ManinSymbolLists.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,2)
sage: m.apply(10,[1,0,0,1]) # not implemented for base class
```

endpoints(*N=None*)

Return cusps $alpha$, $beta$ such that this Manin symbol, viewed as a symbol for level N, is $X^i * Y^{k-2-i}\{alpha, beta\}$.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3)); s
[X^2*Y^4,(2,3)]
sage: s.endpoints()
(1/3, 1/2)
```

i

level()

Return the level of this Manin symbol.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s.level()
5
```

lift_to_sl2z(*N=None*)

Return a lift of this Manin symbol to $SL_2(\mathbf{Z})$.

If this Manin symbol is (c, d) and N is its level, this function returns a list $[a, b, c', d']$ that defines a 2x2 matrix with determinant 1 and integer entries, such that $c = c' \pmod N$ and $d = d' \pmod N$.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s
[X^2*Y^4,(2,3)]
sage: s.lift_to_sl2z()
[1, 1, 2, 3]
```

modular_symbol_rep()

Return a representation of `self` as a formal sum of modular symbols.

The result is not cached.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s.modular_symbol_rep()
 144*X^6*{1/3, 1/2} - 384*X^5*Y*{1/3, 1/2} + 424*X^4*Y^2*{1/3, 1/2} - 248*X^3*Y^
↪3*{1/3, 1/2} + 81*X^2*Y^4*{1/3, 1/2} - 14*X*Y^5*{1/3, 1/2} + Y^6*{1/3, 1/2}
```

tuple()

Return the 3-tuple (i, u, v) of this Manin symbol.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s.tuple()
(2, 2, 3)
```

u

v

weight()

Return the weight of this Manin symbol.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: s = ManinSymbol(m,(2,2,3))
sage: s.weight()
8
```

sage.modular.modsym.manin_symbol.**is_ManinSymbol**(x)

Return `True` if `x` is a *ManinSymbol*.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol import ManinSymbol, is_ManinSymbol
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(6, 4)
sage: s = ManinSymbol(m, m.symbol_list()[3])
sage: s
[Y^2,(1,2)]
sage: is_ManinSymbol(s)
True
sage: is_ManinSymbol(m[3])
True
```

CHAPTER

EIGHT

MANIN SYMBOL LISTS

There are various different classes holding lists of Manin symbols of different types. The hierarchy is as follows:

- *ManinSymbolList*
  - *ManinSymbolList_group*
    - *ManinSymbolList_gamma0*
    - *ManinSymbolList_gamma1*
    - *ManinSymbolList_gamma_h*
  - *ManinSymbolList_character*

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList**(*weight*, *lst*)

Bases: sage.structure.parent.Parent

Base class for lists of all Manin symbols for a given weight, group or character.

Element

alias of *sage.modular.modsym.manin_symbol.ManinSymbol*

apply(*j*, *X*)

Apply the matrix $X = [a, b; c, d]$ to the j-th Manin symbol.

Implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.apply(10, [1,2,0,1])
Traceback (most recent call last):
...
NotImplementedError: Only implemented in derived classes
```

apply_I(*j*)

Apply the matrix $I = [-1, 0; 0, 1]$ to the j-th Manin symbol.

Implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.apply_I(10)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
NotImplementedError: Only implemented in derived classes
```

apply_S(j)

Apply the matrix $S = [0, -1; 1, 0]$ to the j-th Manin symbol.

Implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.apply_S(10)
Traceback (most recent call last):
...
NotImplementedError: Only implemented in derived classes
```

apply_T(j)

Apply the matrix $T = [0, 1; -1, -1]$ to the j-th Manin symbol.

Implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.apply_T(10)
Traceback (most recent call last):
...
NotImplementedError: Only implemented in derived classes
```

apply_TT(j)

Apply the matrix $TT = T^2 = [-1, -1; 0, 1]$ to the j-th Manin symbol.

Implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.apply_TT(10)
Traceback (most recent call last):
...
NotImplementedError: Only implemented in derived classes
```

index(x)

Return the index of `x` in the list of Manin symbols.

INPUT:

- `x` – a triple of integers (i, u, v) defining a valid Manin symbol, which need not be normalized

OUTPUT:

integer – the index of the normalized Manin symbol equivalent to (i, u, v). If `x` is not in `self`, -1 is returned.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.index(m.symbol_list()[2])
2
sage: S = m.symbol_list()
sage: all(i == m.index(S[i]) for i in range(len(S)))
True
```

list()

Return all the Manin symbols in `self` as a list.

Cached for subsequent calls.

OUTPUT:

A list of *ManinSymbol* objects, which is a copy of the complete list of Manin symbols.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.manin_symbol_list() # not implemented for the base class
```

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(6, 4)
sage: m.manin_symbol_list()
[[Y^2,(0,1)],
[Y^2,(1,0)],
[Y^2,(1,1)],
...
[X^2,(3,1)],
[X^2,(3,2)]]
```

manin_symbol(*i*)

Return the `i`-th Manin symbol in this *ManinSymbolList*.

INPUT:

- `i` – integer, a valid index of a symbol in this list

OUTPUT:

ManinSymbol – the i'th Manin symbol in the list.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.manin_symbol(3) # not implemented for base class
```

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(6, 4)
sage: s = m.manin_symbol(3); s
[Y^2,(1,2)]
sage: type(s)
<class 'sage.modular.modsym.manin_symbol.ManinSymbol'>
```

manin_symbol_list()

Return all the Manin symbols in `self` as a list.

Cached for subsequent calls.

OUTPUT:

A list of *ManinSymbol* objects, which is a copy of the complete list of Manin symbols.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.manin_symbol_list() # not implemented for the base class
```

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(6, 4)
sage: m.manin_symbol_list()
[[Y^2,(0,1)],
[Y^2,(1,0)],
[Y^2,(1,1)],
...
[X^2,(3,1)],
[X^2,(3,2)]]
```

normalize(*x*)

Return a normalized Manin symbol from `x`.

To be implemented in derived classes.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6,P1List(11))
sage: m.normalize((0,6,7)) # not implemented in base class
```

symbol_list()

Return the list of symbols of `self`.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList
sage: m = ManinSymbolList(6, P1List(11))
```

weight()

Return the weight of the Manin symbols in this *ManinSymbolList*.

OUTPUT:

integer – the weight of the Manin symbols in the list.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(6, 4)
sage: m.weight()
4
```

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList_character**(*character*, *weight*)

Bases: *sage.modular.modsym.manin_symbol_list.ManinSymbolList*

List of Manin symbols with character.

INPUT:

- `character` – (DirichletCharacter) the Dirichlet character
- `weight` – (integer) the weight

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.manin_symbol_list()
[(0,1), (1,0), (1,1), (1,2), (1,3), (2,1)]
sage: m == loads(dumps(m))
True
```

apply(*j*, *m*)

Apply the integer matrix $m = [a, b; c, d]$ to the j-th Manin symbol.

INPUT:

- `j` (integer): the index of the symbol to act on.
- `m` (list of ints): $[a, b, c, d]$ where $m = [a, b; c, d]$ is the matrix to be applied.

OUTPUT:

A list of pairs (j, c_i), where each c_i is an integer, j is an integer (the j-th Manin symbol), and the sum $c_i * x_i$ is the image of self under the right action of the matrix $[a, b; c, d]$. Here the right action of $g = [a, b; c, d]$ on a Manin symbol $[P(X, Y), (u, v)]$ is by definition $[P(aX + bY, cX + dY), (u, v) * g]$.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,4)
sage: m[6]
[X*Y,(0,1)]
sage: m.apply(4, [1,0,0,1])
[(4, 1)]
sage: m.apply(1, [-1,0,0,1])
[(1, -1)]
```

apply_I(*j*)

Apply the matrix $I = [-1, 0, 0, 1]$ to the j-th Manin symbol.

INPUT:

- `j` - (integer) a symbol index

OUTPUT:

`(k, s)` where k is the index of the symbol obtained by acting on the j'th symbol with I, and s is the parity of the j'th symbol.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.apply_I(4)
(2, -1)
sage: [m.apply_I(i) for i in range(len(m))]
[(0, 1), (1, -1), (4, -1), (3, -1), (2, -1), (5, 1)]
```

apply_S(*j*)

Apply the matrix $S = [0, 1; -1, 0]$ to the j-th Manin symbol.

INPUT:

- j - (integer) a symbol index.

OUTPUT:

(k, s) where k is the index of the symbol obtained by acting on the j'th symbol with S, and s is the parity of the j'th symbol.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.apply_S(4)
(2, -1)
sage: [m.apply_S(i) for i in range(len(m))]
[(1, 1), (0, -1), (4, 1), (5, -1), (2, -1), (3, 1)]
```

apply_T(*j*)

Apply the matrix $T = [0, 1, -1, -1]$ to the j-th Manin symbol.

INPUT:

- j - (integer) a symbol index.

OUTPUT:

A list of pairs (j, c_i), where each c_i is an integer, j is an integer (the j-th Manin symbol), and the sum $c_i * x_i$ is the image of self under the right action of the matrix T.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.apply_T(4)
[(1, -1)]
sage: [m.apply_T(i) for i in range(len(m))]
[[(4, 1)], [(0, -1)], [(3, 1)], [(5, 1)], [(1, -1)], [(2, 1)]]
```

apply_TT(j)

Apply the matrix $TT = [-1, -1, 0, 1]$ to the j-th Manin symbol.

INPUT:

- j - (integer) a symbol index

OUTPUT:

A list of pairs (j, c_i), where each c_i is an integer, j is an integer (the j-th Manin symbol), and the sum $c_i * x_i$ is the image of self under the right action of the matrix T^2.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.apply_TT(4)
[(0, 1)]
sage: [m.apply_TT(i) for i in range(len(m))]
[[(1, -1)], [(4, -1)], [(5, 1)], [(2, 1)], [(0, 1)], [(3, 1)]]
```

character()

Return the character of this *ManinSymbolList_character* object.

OUTPUT:

The Dirichlet character of this Manin symbol list.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,2); m
Manin Symbol List of weight 2 for Gamma1(4) with character [-1]
sage: m.character()
Dirichlet character modulo 4 of conductor 4 mapping 3 |--> -1
```

index(x)

Return the index of a standard Manin symbol equivalent to x, together with a scaling factor.

INPUT:

- x – 3-tuple of integers defining an element of this list of Manin symbols, which need not be normalized

OUTPUT:

A pair (i, s) where i is the index of the Manin symbol equivalent to x and s is the scalar (an element of the base field). If there is no Manin symbol equivalent to x in the list, then (-1, 0) is returned.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,4); m
Manin Symbol List of weight 4 for Gamma1(4) with character [-1]
```

(continues on next page)

(continued from previous page)

```
sage: [m.index(s.tuple()) for s in m.manin_symbol_list()]
[(0, 1),
(1, 1),
(2, 1),
(3, 1),
...
(16, 1),
(17, 1)]
```

level()

Return the level of this *ManinSymbolList*.

OUTPUT:

`integer` - the level of the symbols in this list.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: ManinSymbolList_character(eps,4).level()
4
```

normalize(*x*)

Return the normalization of the Manin Symbol `x` with respect to this list, together with the normalizing scalar.

INPUT:

- `x` - 3-tuple of integers `(i,u,v)`, defining an element of this list of Manin symbols, which need not be normalized.

OUTPUT:

`((i,u,v),s)`, where `(i,u,v)` is the normalized Manin symbol equivalent to `x`, and `s` is the normalizing scalar.

EXAMPLES:

```
sage: eps = DirichletGroup(4).gen(0)
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_
↪character
sage: m = ManinSymbolList_character(eps,4); m
Manin Symbol List of weight 4 for Gamma1(4) with character [-1]
sage: [m.normalize(s.tuple()) for s in m.manin_symbol_list()]
[((0, 0, 1), 1),
 ((0, 1, 0), 1),
 ((0, 1, 1), 1),
 ...
 ((2, 1, 3), 1),
 ((2, 2, 1), 1)]
```

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList_gamma0**(*level*, *weight*)

Bases: *sage.modular.modsym.manin_symbol_list.ManinSymbolList_group*

Class for Manin symbols for $\Gamma_0(N)$.

INPUT:

- `level` - (integer): the level.
- `weight` - (integer): the weight.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,2); m
Manin Symbol List of weight 2 for Gamma0(5)
sage: m.manin_symbol_list()
[(0,1), (1,0), (1,1), (1,2), (1,3), (1,4)]
sage: m = ManinSymbolList_gamma0(6,4); m
Manin Symbol List of weight 4 for Gamma0(6)
sage: len(m)
36
```

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList_gamma1**(*level*, *weight*)

Bases: *sage.modular.modsym.manin_symbol_list.ManinSymbolList_group*

Class for Manin symbols for $\Gamma_1(N)$.

INPUT:

- `level` - (integer): the level.
- `weight` - (integer): the weight.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma1
sage: m = ManinSymbolList_gamma1(5,2); m
Manin Symbol List of weight 2 for Gamma1(5)
sage: m.manin_symbol_list()
[(0,1),
(0,2),
(0,3),
...
(4,3),
(4,4)]
sage: m = ManinSymbolList_gamma1(6,4); m
Manin Symbol List of weight 4 for Gamma1(6)
sage: len(m)
72
sage: m == loads(dumps(m))
True
```

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList_gamma_h**(*group*, *weight*)

Bases: *sage.modular.modsym.manin_symbol_list.ManinSymbolList_group*

Class for Manin symbols for $\Gamma_H(N)$.

INPUT:

- `group` - (integer): the congruence subgroup.
- `weight` - (integer): the weight.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma_h
sage: G = GammaH(117, [4])
sage: m = ManinSymbolList_gamma_h(G,2); m
Manin Symbol List of weight 2 for Congruence Subgroup Gamma_H(117) with H generated␣
↪by [4]
sage: m.manin_symbol_list()[100:110]
[(1,88),
(1,89),
(1,90),
(1,91),
(1,92),
(1,93),
(1,94),
(1,95),
(1,96),
(1,97)]
sage: len(m.manin_symbol_list())
2016
sage: m == loads(dumps(m))
True
```

group()

Return the group associated to self.

EXAMPLES:

```
sage: ModularSymbols(GammaH(12, [5]), 2).manin_symbols().group()
Congruence Subgroup Gamma_H(12) with H generated by [5]
```

class sage.modular.modsym.manin_symbol_list.**ManinSymbolList_group**(*level*, *weight*, *syms*)

Bases: *sage.modular.modsym.manin_symbol_list.ManinSymbolList*

Base class for Manin symbol lists for a given group.

INPUT:

- `level` – integer level
- `weight` – integer weight
- **syms – something with normalize and list methods,** e.g. *P1List*.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_group
sage: ManinSymbolList_group(11, 2, P1List(11))
<sage.modular.modsym.manin_symbol_list.ManinSymbolList_group_with_category object␣
↪at ...>
```

apply(*j*, *m*)

Apply the matrix $m = [a, b; c, d]$ to the j-th Manin symbol.

INPUT:

- `j` - (int) a symbol index
- `m = [a, b, c, d]` a list of 4 integers, which defines a 2x2 matrix

OUTPUT:

a list of pairs (j_i, α_i), where each α_i is a nonzero integer, j_i is an integer (index of the j_i-th Manin symbol), and $\sum_i \alpha_i x_{j_i}$ is the image of the j-th Manin symbol under the right action of the matrix [a,b;c,d]. Here the right action of $g = [a, b; c, d]$ on a Manin symbol $[P(X, Y), (u, v)]$ is $[P(aX + bY, cX + dY), (u, v)g]$.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: m.apply(40, [2,3,1,1])
[(0, 729), (6, 2916), (12, 4860), (18, 4320),
 (24, 2160), (30, 576), (36, 64)]
```

apply_I(*j*)

Apply the matrix $I = [-1, 0, 0, 1]$ to the j-th Manin symbol.

INPUT:

- j - (int) a symbol index

OUTPUT:

(k, s) where k is the index of the symbol obtained by acting on the j'th symbol with I, and s is the parity of the j'th symbol (a Python int, either 1 or -1)

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: m.apply_I(4)
(3, 1)
sage: [m.apply_I(i) for i in range(10)]
[(0, 1),
(1, 1),
(5, 1),
(4, 1),
(3, 1),
(2, 1),
(6, -1),
(7, -1),
(11, -1),
(10, -1)]
```

apply_S(*j*)

Apply the matrix $S = [0, -1; 1, 0]$ to the j-th Manin symbol.

INPUT:

- j – (int) a symbol index

OUTPUT:

(k, s) where k is the index of the symbol obtained by acting on the j'th symbol with S, and s is the parity of the j'th symbol (a Python int, either 1 or -1).

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: m.apply_S(4)
```

(continues on next page)

(continued from previous page)

```
(40, 1)
sage: [m.apply_S(i) for i in range(len(m))]
[(37, 1),
(36, 1),
(41, 1),
(39, 1),
(40, 1),
(38, 1),
(31, -1),
(30, -1),
(35, -1),
(33, -1),
(34, -1),
(32, -1),
...
(4, 1),
(2, 1)]
```

apply_T(j)

Apply the matrix $T = [0, 1, -1, -1]$ to the j-th Manin symbol.

INPUT:

- j - (int) a symbol index

OUTPUT: see documentation for apply()

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: m.apply_T(4)
[(3, 1), (9, -6), (15, 15), (21, -20), (27, 15), (33, -6), (39, 1)]
sage: [m.apply_T(i) for i in range(10)]
[[(5, 1), (11, -6), (17, 15), (23, -20), (29, 15), (35, -6), (41, 1)],
[(0, 1), (6, -6), (12, 15), (18, -20), (24, 15), (30, -6), (36, 1)],
[(4, 1), (10, -6), (16, 15), (22, -20), (28, 15), (34, -6), (40, 1)],
[(2, 1), (8, -6), (14, 15), (20, -20), (26, 15), (32, -6), (38, 1)],
[(3, 1), (9, -6), (15, 15), (21, -20), (27, 15), (33, -6), (39, 1)],
[(1, 1), (7, -6), (13, 15), (19, -20), (25, 15), (31, -6), (37, 1)],
[(5, 1), (11, -5), (17, 10), (23, -10), (29, 5), (35, -1)],
[(0, 1), (6, -5), (12, 10), (18, -10), (24, 5), (30, -1)],
[(4, 1), (10, -5), (16, 10), (22, -10), (28, 5), (34, -1)],
[(2, 1), (8, -5), (14, 10), (20, -10), (26, 5), (32, -1)]]
```

apply_TT(j)

Apply the matrix $TT = [-1, -1, 0, 1]$ to the j-th Manin symbol.

INPUT:

- j - (int) a symbol index

OUTPUT: see documentation for apply()

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: m.apply_TT(4)
[(38, 1)]
sage: [m.apply_TT(i) for i in range(10)]
[[(37, 1)],
[(41, 1)],
[(39, 1)],
[(40, 1)],
[(38, 1)],
[(36, 1)],
[(31, -1), (37, 1)],
[(35, -1), (41, 1)],
[(33, -1), (39, 1)],
[(34, -1), (40, 1)]]
```

level()

Return the level of this *ManinSymbolList*.

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: ManinSymbolList_gamma0(5,2).level()
5
```

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma1
sage: ManinSymbolList_gamma1(51,2).level()
51
```

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma_h
sage: ManinSymbolList_gamma_h(GammaH(117, [4]),2).level()
117
```

normalize(*x*)

Return the normalization of the Manin symbol `x` with respect to this list.

INPUT:

- `x` – (3-tuple of ints) a tuple defining a ManinSymbol

OUTPUT:

`(i,u,v)` – (3-tuple of ints) another tuple defining the associated normalized ManinSymbol

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: m = ManinSymbolList_gamma0(5,8)
sage: [m.normalize(s.tuple()) for s in m.manin_symbol_list()][:10]
[(0, 0, 1),
 (0, 1, 0),
 (0, 1, 1),
 (0, 1, 2),
 (0, 1, 3),
 (0, 1, 4),
```

(continues on next page)

(continued from previous page)

```
(1, 0, 1),
(1, 1, 0),
(1, 1, 1),
(1, 1, 2)]
```

CHAPTER

NINE

SPACE OF BOUNDARY MODULAR SYMBOLS

Used mainly for computing the cuspidal subspace of modular symbols. The space of boundary symbols of sign 0 is isomorphic as a Hecke module to the dual of the space of Eisenstein series, but this does not give a useful method of computing Eisenstein series, since there is no easy way to extract the constant terms.

We represent boundary modular symbols as a sum of Manin symbols of the form $[P, u/v]$, where u/v is a cusp for our group G. The group of boundary modular symbols naturally embeds into a vector space $B_k(G)$ (see Stein, section 8.4, or Merel, section 1.4, where this space is called $\mathbf{C}[\Gamma \backslash \mathbf{Q}]_k$, for a definition), which is a finite dimensional $\mathbf{Q}$ vector space, of dimension equal to the number of cusps for G (if k is even), or the number of regular cusps (if k is odd). The embedding takes $[P, u/v]$ to $P(u, v) \cdot [(u, v)]$. We represent the basis vectors by pairs $[(u, v)]$ with u, v coprime. On $B_k(G)$, we have the relations

$$[\gamma \cdot (u, v)] = [(u, v)]$$

for all $\gamma \in G$ and

$$[(\lambda u, \lambda v)] = \mathrm{sign}(\lambda)^k [(u, v)]$$

for all $\lambda \in \mathbf{Q}^\times$.

It's possible for these relations to kill a class, i.e., for a pair $[(u, v)]$ to be 0. For example, when $N = 4$ and $k = 3$ then $(-1, -2)$ is equivalent mod $\Gamma_1(4)$ to $(1, 2)$ since $2 = -2 \bmod 4$ and $1 = -1 \bmod 2$. But since k is odd, $[(-1, -2)]$ is also equivalent to $-[(1, 2)]$. Thus this symbol is equivalent to its negative, hence 0 (notice that this wouldn't be the case in characteristic 2). This happens for any irregular cusp when the weight is odd; there are no irregular cusps on $\Gamma_1(N)$ except when $N = 4$, but there can be more on Γ_H groups. See also prop 2.30 of Stein's Ph.D. thesis.

In addition, in the case that our space is of sign $\sigma = 1$ or -1, we also have the relation $[(-u, v)] = \sigma \cdot [(u, v)]$. This relation can also combine with the above to kill a cusp class - for instance, take (u,v) = (1,3) for $\Gamma_1(5)$. Then since the cusp $\frac{1}{3}$ is $\Gamma_1(5)$-equivalent to the cusp $-\frac{1}{3}$, we have that $[(1, 3)] = [(-1, 3)]$. Now, on the minus subspace, we also have that $[(-1, 3)] = -[(1, 3)]$, which means this class must vanish. Notice that this cannot be used to show that $[(1, 0)]$ or $[(0, 1)]$ is 0.

Note: Special care must be taken when working with the images of the cusps 0 and ∞ in $B_k(G)$. For all cusps *except* 0 and ∞, multiplying the cusp by -1 corresponds to taking $[(u, v)]$ to $[(-u, v)]$ in $B_k(G)$. This means that $[(u, v)]$ is equivalent to $[(-u, v)]$ whenever $\frac{u}{v}$ is equivalent to $-\frac{u}{v}$, except in the case of 0 and ∞. We have the following conditions for $[(1, 0)]$ and $[(0, 1)]$:

- $[(0, 1)] = \sigma \cdot [(0, 1)]$, so $[(0, 1)]$ is 0 exactly when $\sigma = -1$.
- $[(1, 0)] = \sigma \cdot [(-1, 0)]$ and $[(1, 0)] = (-1)^k [(-1, 0)]$, so $[(1, 0)] = 0$ whenever $\sigma \neq (-1)^k$.

Note: For all the spaces of boundary symbols below, no work is done to determine the cusps for G at creation time. Instead, cusps are added as they are discovered in the course of computation. As a result, the rank of a space can change as a computation proceeds.

REFERENCES:

- Merel, "Universal Fourier expansions of modular forms." Springer LNM 1585 (1994), pg. 59-95.
- Stein, "Modular Forms, a computational approach." AMS (2007).

class sage.modular.modsym.boundary.**BoundarySpace**(*group=Modular Group SL(2, Z)*, *weight=2*, *sign=0*, *base_ring=Rational Field*, *character=None*)

Bases: sage.modular.hecke.module.HeckeModule_generic

Space of boundary symbols for a congruence subgroup of SL_2(Z).

This class is an abstract base class, so only derived classes should be instantiated.

INPUT:

- weight - int, the weight
- group - arithgroup.congroup_generic.CongruenceSubgroup, a congruence subgroup.
- sign - int, either -1, 0, or 1
- base_ring - rings.Ring (defaults to the rational numbers)

EXAMPLES:

```
sage: B = ModularSymbols(Gamma0(11),2).boundary_space()
sage: isinstance(B, sage.modular.modsym.boundary.BoundarySpace)
True
sage: B == loads(dumps(B))
True
```

character()

Return the Dirichlet character associated to this space of boundary modular symbols.

EXAMPLES:

```
sage: ModularSymbols(DirichletGroup(7).0, 6).boundary_space().character()
Dirichlet character modulo 7 of conductor 7 mapping 3 |--> zeta6
```

free_module()

Return the underlying free module for self.

EXAMPLES:

```
sage: B = ModularSymbols(Gamma1(7), 5, sign=-1).boundary_space()
sage: B.free_module()
Sparse vector space of dimension 0 over Rational Field
sage: x = B(Cusp(0)) ; y = B(Cusp(1/7)) ; B.free_module()
Sparse vector space of dimension 1 over Rational Field
```

gen(*i=0*)

Return the i-th generator of this space.

EXAMPLES:

```
sage: B = ModularSymbols(Gamma0(24), 4).boundary_space()
sage: B.gen(0)
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: only 0 generators known for Space of Boundary Modular Symbols for␣
↪Congruence Subgroup Gamma0(24) of weight 4 over Rational Field
sage: B(Cusp(1/3))
[1/3]
sage: B.gen(0)
[1/3]
```

group()

Return the congruence subgroup associated to this space of boundary modular symbols.

EXAMPLES:

```
sage: ModularSymbols(GammaH(14,[9]), 2).boundary_space().group()
Congruence Subgroup Gamma_H(14) with H generated by [9]
```

is_ambient()

Return `True` if `self` is a space of boundary symbols associated to an ambient space of modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(6), 4)
sage: M.is_ambient()
True
sage: M.boundary_space().is_ambient()
True
```

rank()

The rank of the space generated by boundary symbols that have been found so far in the course of computing the boundary map.

Warning: This number may change as more elements are coerced into this space!! (This is an implementation detail that will likely change.)

EXAMPLES:

```
sage: M = ModularSymbols(Gamma0(72), 2) ; B = M.boundary_space()
sage: B.rank()
0
sage: _ = [ B(x) for x in M.basis() ]
sage: B.rank()
16
```

Test that trac ticket #7837 is fixed:

```
sage: ModularSymbols(Gamma1(4),7).boundary_map().codomain().dimension()
2
sage: ModularSymbols(Gamma1(4),7, sign=1).boundary_map().codomain().dimension()
1
sage: ModularSymbols(Gamma1(4),7, sign=-1).boundary_map().codomain().dimension()
1
```

sign()

Return the sign of the complex conjugation involution on this space of boundary modular symbols.

EXAMPLES:

```
sage: ModularSymbols(13,2,sign=-1).boundary_space().sign()
-1
```

weight()

Return the weight of this space of boundary modular symbols.

EXAMPLES:

```
sage: ModularSymbols(Gamma1(9), 5).boundary_space().weight()
5
```

class sage.modular.modsym.boundary.**BoundarySpaceElement**(*parent*, *x*)

Bases: sage.modular.hecke.element.HeckeModuleElement

Create a boundary symbol.

INPUT:

- parent - BoundarySpace; a space of boundary modular symbols
- x - a dict with integer keys and values in the base field of parent.

EXAMPLES:

```
sage: B = ModularSymbols(Gamma0(32), sign=-1).boundary_space()
sage: B(Cusp(1,8))
[1/8]
sage: B.0
[1/8]
sage: type(B.0)
<class 'sage.modular.modsym.boundary.BoundarySpaceElement'>
```

coordinate_vector()

Return self as a vector on the QQ-vector space with basis self.parent()._known_cusps().

EXAMPLES:

```
sage: B = ModularSymbols(18,4,sign=1).boundary_space()
sage: x = B(Cusp(1/2)) ; x
[1/2]
sage: x.coordinate_vector()
(1)
sage: ((18/5)*x).coordinate_vector()
(18/5)
sage: B(Cusp(0))
[0]
sage: x.coordinate_vector()
(1)
sage: x = B(Cusp(1/2)) ; x
[1/2]
sage: x.coordinate_vector()
(1, 0)
```

class sage.modular.modsym.boundary.**BoundarySpace_wtk_eps**(*eps*, *weight*, *sign=0*)
Bases: *sage.modular.modsym.boundary.BoundarySpace*

Space of boundary modular symbols with given weight, character, and sign.

INPUT:

- eps - dirichlet.DirichletCharacter, the "Nebentypus" character.
- weight - int, the weight = 2
- sign - int, either -1, 0, or 1

EXAMPLES:

```
sage: B = ModularSymbols(DirichletGroup(6).0, 4).boundary_space() ; B
Boundary Modular Symbols space of level 6, weight 4, character [-1] and dimension 0␣
↪over Rational Field
sage: type(B)
<class 'sage.modular.modsym.boundary.BoundarySpace_wtk_eps_with_category'>
sage: B == loads(dumps(B))
True
```

class sage.modular.modsym.boundary.**BoundarySpace_wtk_g0**(*level*, *weight*, *sign*, *F*)
Bases: *sage.modular.modsym.boundary.BoundarySpace*

Initialize a space of boundary symbols of weight k for Gamma_0(N) over base field F.

INPUT:

- level - int, the level
- weight - integer weight = 2.
- sign - int, either -1, 0, or 1
- F - field

EXAMPLES:

```
sage: B = ModularSymbols(Gamma0(2), 5).boundary_space()
sage: type(B)
<class 'sage.modular.modsym.boundary.BoundarySpace_wtk_g0_with_category'>
sage: B == loads(dumps(B))
True
```

class sage.modular.modsym.boundary.**BoundarySpace_wtk_g1**(*level*, *weight*, *sign*, *F*)
Bases: *sage.modular.modsym.boundary.BoundarySpace*

Initialize a space of boundary modular symbols for Gamma1(N).

INPUT:

- level - int, the level
- weight - int, the weight = 2
- sign - int, either -1, 0, or 1
- F - base ring

EXAMPLES:

```
sage: from sage.modular.modsym.boundary import BoundarySpace_wtk_g1
sage: B = BoundarySpace_wtk_g1(17, 2, 0, QQ) ; B
Boundary Modular Symbols space for Gamma_1(17) of weight 2 over Rational Field
sage: B == loads(dumps(B))
True
```

class sage.modular.modsym.boundary.**BoundarySpace_wtk_gamma_h**(*group*, *weight*, *sign*, *F*)

Bases: *sage.modular.modsym.boundary.BoundarySpace*

Initialize a space of boundary modular symbols for GammaH(N).

INPUT:

- `group` - congruence subgroup Gamma_H(N).
- `weight` - int, the weight = 2
- `sign` - int, either -1, 0, or 1
- `F` - base ring

EXAMPLES:

```
sage: from sage.modular.modsym.boundary import BoundarySpace_wtk_gamma_h
sage: B = BoundarySpace_wtk_gamma_h(GammaH(13,[3]), 2, 0, QQ) ; B
Boundary Modular Symbols space for Congruence Subgroup Gamma_H(13) with H generated␣
↪by [3] of weight 2 over Rational Field
sage: B == loads(dumps(B))
True
```

A test case from trac ticket #6072:

```
sage: ModularSymbols(GammaH(8,[5]), 3).boundary_map()
Hecke module morphism boundary map defined by the matrix
[-1  0  0  0]
[ 0 -1  0  0]
[ 0  0 -1  0]
[ 0  0  0 -1]
Domain: Modular Symbols space of dimension 4 for Congruence Subgroup ...
Codomain: Boundary Modular Symbols space for Congruence Subgroup Gamma_H(8) ...
```

CHAPTER

TEN

HEILBRONN MATRIX COMPUTATION

class sage.modular.modsym.heilbronn.**Heilbronn**

Bases: object

apply(*u*, *v*, *N*)

Return a list of pairs ((c,d),m), which is obtained as follows:

1) Compute the images (a,b) of the vector (u,v) (mod N) acted on by each of the HeilbronnCremona matrices in self.

2) Reduce each (a,b) to canonical form (c,d) using p1normalize.

3) Sort.

4) Create the list ((c,d),m), where m is the number of times that (c,d) appears in the list created in steps 1-3 above. Note that the pairs ((c,d),m) are sorted lexicographically by (c,d).

INPUT:

- u, v, N – integers

OUTPUT: list

EXAMPLES:

```
sage: H = sage.modular.modsym.heilbronn.HeilbronnCremona(2); H
The Cremona-Heilbronn matrices of determinant 2
sage: H.apply(1,2,7)
[((1, 1), 1), ((1, 4), 1), ((1, 5), 1), ((1, 6), 1)]
```

to_list()

Return the list of Heilbronn matrices corresponding to self.

Each matrix is given as a list of four ints.

EXAMPLES:

```
sage: H = HeilbronnCremona(2); H
The Cremona-Heilbronn matrices of determinant 2
sage: H.to_list()
[[1, 0, 0, 2], [2, 0, 0, 1], [2, 1, 0, 1], [1, 0, 1, 2]]
```

class sage.modular.modsym.heilbronn.**HeilbronnCremona**

Bases: *sage.modular.modsym.heilbronn.Heilbronn*

Create the list of Heilbronn-Cremona matrices of determinant p.

EXAMPLES:

```
sage: H = HeilbronnCremona(3) ; H
The Cremona-Heilbronn matrices of determinant 3
sage: H.to_list()
[[1, 0, 0, 3],
[3, 1, 0, 1],
[1, 0, 1, 3],
[3, 0, 0, 1],
[3, -1, 0, 1],
[-1, 0, 1, -3]]
```

p

class sage.modular.modsym.heilbronn.**HeilbronnMerel**

Bases: *sage.modular.modsym.heilbronn.Heilbronn*

Initialize the list of Merel-Heilbronn matrices of determinant n.

EXAMPLES:

```
sage: H = HeilbronnMerel(3) ; H
The Merel-Heilbronn matrices of determinant 3
sage: H.to_list()
[[1, 0, 0, 3],
[1, 0, 1, 3],
[1, 0, 2, 3],
[2, 1, 1, 2],
[3, 0, 0, 1],
[3, 1, 0, 1],
[3, 2, 0, 1]]
```

n

sage.modular.modsym.heilbronn.**hecke_images_gamma0_weight2**(*u*, *v*, *N*, *indices*, *R*)

INPUT:

- `u, v, N` - integers so that gcd(u,v,N) = 1
- `indices` - a list of positive integers
- `R` - matrix over QQ that writes each elements of P1 = P1List(N) in terms of a subset of P1.

OUTPUT: a dense matrix whose columns are the images T_n(x) for n in indices and x the Manin symbol (u,v), expressed in terms of the basis.

EXAMPLES:

```
sage: M = ModularSymbols(23,2,1)
sage: A = sage.modular.modsym.heilbronn.hecke_images_gamma0_weight2(1,0,23,[1..6],M.
↪manin_gens_to_basis())
sage: rowsA = A.rows()
sage: z = M((1,0))
sage: all(M.T(n)(z).element() == rowsA[n-1] for n in [1..6])
True
```

sage.modular.modsym.heilbronn.**hecke_images_gamma0_weight_k**(*u*, *v*, *i*, *N*, *k*, *indices*, *R*)

INPUT:

- `u, v, N` - integers so that gcd(u,v,N) = 1

- i - integer with 0 <= i <= k-2
- k - weight
- indices - a list of positive integers
- R - matrix over QQ that writes each elements of P1 = P1List(N) in terms of a subset of P1.

OUTPUT: a dense matrix with rational entries whose columns are the images T_n(x) for n in indices and x the Manin symbol $[X^i * Y^(k - 2 - i), (u, v)]$, expressed in terms of the basis.

EXAMPLES:

```
sage: M = ModularSymbols(15,6,sign=-1)
sage: R = M.manin_gens_to_basis()
sage: a,b,c = sage.modular.modsym.heilbronn.hecke_images_gamma0_weight_k(4,1,3,15,6,
↪[1,11,12], R)
sage: x = M((3,4,1)) ; x.element() == a
True
sage: M.T(11)(x).element() == b
True
sage: M.T(12)(x).element() == c
True
```

sage.modular.modsym.heilbronn.**hecke_images_nonquad_character_weight2**(*u*, *v*, *N*, *indices*, *chi*, *R*)

Return images of the Hecke operators T_n for n in the list indices, where chi must be a quadratic Dirichlet character with values in QQ.

R is assumed to be the relation matrix of a weight modular symbols space over QQ with character chi.

INPUT:

- u, v, N - integers so that gcd(u,v,N) = 1
- indices - a list of positive integers
- chi - a Dirichlet character that takes values in a nontrivial extension of QQ.
- R - matrix over QQ that writes each elements of P1 = P1List(N) in terms of a subset of P1.

OUTPUT: a dense matrix with entries in the field QQ(chi) (the values of chi) whose columns are the images T_n(x) for n in indices and x the Manin symbol (u,v), expressed in terms of the basis.

EXAMPLES:

```
sage: chi = DirichletGroup(13).0^2
sage: M = ModularSymbols(chi)
sage: eps = M.character()
sage: R = M.manin_gens_to_basis()
sage: sage.modular.modsym.heilbronn.hecke_images_nonquad_character_weight2(1,0,13,
↪[1,2,6],eps,R)
[            1             0             0             0]
[   zeta6 + 2             0             0            -1]
[            7 -2*zeta6 + 1   -zeta6 - 1     -2*zeta6]
sage: x = M((1,0)); x.element()
(1, 0, 0, 0)
sage: M.T(2)(x).element()
(zeta6 + 2, 0, 0, -1)
sage: M.T(6)(x).element()
(7, -2*zeta6 + 1, -zeta6 - 1, -2*zeta6)
```

`sage.modular.modsym.heilbronn.`**`hecke_images_quad_character_weight2`**(*u*, *v*, *N*, *indices*, *chi*, *R*)

INPUT:

- `u, v, N` - integers so that gcd(u,v,N) = 1
- `indices` - a list of positive integers
- `chi` - a Dirichlet character that takes values in QQ
- **`R` - matrix over QQ(chi) that writes each elements of P1 =** P1List(N) in terms of a subset of P1.

OUTPUT: a dense matrix with entries in the rational field QQ (the values of chi) whose columns are the images T_n(x) for n in indices and x the Manin symbol (u,v), expressed in terms of the basis.

EXAMPLES:

```
sage: chi = DirichletGroup(29,QQ).0
sage: M = ModularSymbols(chi)
sage: R = M.manin_gens_to_basis()
sage: sage.modular.modsym.heilbronn.hecke_images_quad_character_weight2(2,1,29,[1,3,
↪4],chi,R)
[ 0  0  0  0  0 -1]
[ 0  1  0  1  1  1]
[ 0 -2  0  2 -2 -1]
sage: x = M((2,1)) ; x.element()
(0, 0, 0, 0, 0, -1)
sage: M.T(3)(x).element()
(0, 1, 0, 1, 1, 1)
sage: M.T(4)(x).element()
(0, -2, 0, 2, -2, -1)
```

CHAPTER

ELEVEN

LISTS OF MANIN SYMBOLS (ELEMENTS OF $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$) OVER $\mathbf{Q}$

class sage.modular.modsym.p1list.**P1List**

Bases: object

The class for $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$, the projective line modulo N.

EXAMPLES:

```
sage: P = P1List(12); P
The projective line over the integers modulo 12
sage: list(P)
[(0, 1), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), (1,
↪ 9), (1, 10), (1, 11), (2, 1), (2, 3), (2, 5), (3, 1), (3, 2), (3, 4), (3, 7), (4,
↪ 1), (4, 3), (4, 5), (6, 1)]
```

Saving and loading works.

```
sage: loads(dumps(P)) == P
True
```

N()

Return the level or modulus of this P1List.

EXAMPLES:

```
sage: L = P1List(120)
sage: L.N()
120
```

apply_I(*i*)

Return the index of the result of applying the matrix $I = [-1, 0; 0, 1]$ to the i'th element of this P1List.

INPUT:

- i - integer (the index of the element to act on).

EXAMPLES:

```
sage: L = P1List(120)
sage: L[10]
(1, 9)
sage: L.apply_I(10)
112
sage: L[112]
(1, 111)
```

(continues on next page)

(continued from previous page)

```
sage: L.normalize(-1,9)
(1, 111)
```

This operation is an involution:

```
sage: all(L.apply_I(L.apply_I(i)) == i for i in range(len(L)))
True
```

apply_S(i)
Return the index of the result of applying the matrix $S = [0, -1; 1, 0]$ to the i'th element of this P1List.

INPUT:

- i - integer (the index of the element to act on).

EXAMPLES:

```
sage: L = P1List(120)
sage: L[10]
(1, 9)
sage: L.apply_S(10)
159
sage: L[159]
(3, 13)
sage: L.normalize(-9,1)
(3, 13)
```

This operation is an involution:

```
sage: all(L.apply_S(L.apply_S(i)) == i for i in range(len(L)))
True
```

apply_T(i)
Return the index of the result of applying the matrix $T = [0, 1; -1, -1]$ to the i'th element of this P1List.

INPUT:

- i - integer (the index of the element to act on).

EXAMPLES:

```
sage: L = P1List(120)
sage: L[10]
(1, 9)
sage: L.apply_T(10)
157
sage: L[157]
(3, 10)
sage: L.normalize(9,-10)
(3, 10)
```

This operation has order three:

```
sage: all(L.apply_T(L.apply_T(L.apply_T(i))) == i for i in range(len(L)))
True
```

index(u, v)

Return the index of the class of (u, v) in the fixed list of representatives of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$.

INPUT:

- `u, v` - integers, with $\gcd(u, v, N) = 1$.

OUTPUT:

- `i` - the index of u, v, in the P1list.

EXAMPLES:

```
sage: L = P1List(120)
sage: L[100]
(1, 99)
sage: L.index(1,99)
100
sage: all(L.index(L[i][0],L[i][1])==i for i in range(len(L)))
True
```

index_of_normalized_pair(u, v)

Return the index of the class of (u, v) in the fixed list of representatives of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$.

INPUT:

- `u, v` - integers, with $\gcd(u, v, N) = 1$, normalized so they lie in the list.

OUTPUT:

- `i` - the index of $(u : v)$, in the P1list.

EXAMPLES:

```
sage: L = P1List(120)
sage: L[100]
(1, 99)
sage: L.index_of_normalized_pair(1,99)
100
sage: all(L.index_of_normalized_pair(L[i][0],L[i][1])==i for i in range(len(L)))
True
```

lift_to_sl2z(i)

Lift the i'th element of this P1list to an element of $SL(2, \mathbf{Z})$.

If the i'th element is (c, d), this function computes and returns a list $[a, b, c', d']$ that defines a 2x2 matrix with determinant 1 and integer entries, such that $c = c' \pmod{N}$ and $d = d' \pmod{N}$.

INPUT:

- `i` - integer (the index of the element to lift).

EXAMPLES:

```
sage: p = P1List(11)
sage: p.list()[3]
(1, 2)

sage: p.lift_to_sl2z(3)
[0, -1, 1, 2]
```

AUTHORS:

- Justin Walker

list()

Return the underlying list of this *P1List* object.

EXAMPLES:

```
sage: L = P1List(8)
sage: type(L)
<... 'sage.modular.modsym.p1list.P1List'>
sage: type(L.list())
<... 'list'>
```

normalize(*u*, *v*)

Return a normalised element of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$.

INPUT:

- `u, v` - integers, with $\gcd(u, v, N) = 1$.

OUTPUT:

- a 2-tuple `(uu,vv)` where $(uu : vv)$ is a *normalized* representative of $(u : v)$.

NOTE: See also normalize_with_scalar() which also returns the normalizing scalar.

EXAMPLES:

```
sage: L = P1List(120)
sage: (u,v) = (555555555,7777)
sage: uu,vv = L.normalize(555555555,7777)
sage: (uu,vv)
(15, 13)
sage: (uu*v-vv*u) % L.N() == 0
True
```

normalize_with_scalar(*u*, *v*)

Return a normalised element of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$, together with the normalizing scalar.

INPUT:

- `u, v` - integers, with $\gcd(u, v, N) = 1$.

OUTPUT:

- a 3-tuple `(uu,vv,ss)` where $(uu : vv)$ is a *normalized* representative of $(u : v)$, and ss is a scalar such that $(ss * uu, ss * vv) = (u, v) \pmod N$.

EXAMPLES:

```
sage: L = P1List(120)
sage: (u,v) = (555555555,7777)
sage: uu,vv,ss = L.normalize_with_scalar(555555555,7777)
sage: (uu,vv)
(15, 13)
sage: ((ss*uu-u)%L.N(), (ss*vv-v)%L.N())
(0, 0)
sage: (uu*v-vv*u) % L.N() == 0
True
```

class sage.modular.modsym.p1list.**export**

Bases: object

sage.modular.modsym.p1list.**lift_to_sl2z**(*c*, *d*, *N*)

Return a list of Python ints $[a, b, c', d']$ that are the entries of a 2x2 matrix with determinant 1 and lower two entries congruent to c, d modulo N.

INPUT:

- c,d,N - Python ints or longs such that $\gcd(c, d, N) = 1$.

EXAMPLES:

```
sage: lift_to_sl2z(2,3,6)
[1, 1, 2, 3]
sage: lift_to_sl2z(2,3,6000000)
[1, 1, 2, 3]
```

You will get a ValueError exception if the input is invalid. Note that here gcd(15,6,24)=3:

```
sage: lift_to_sl2z(15,6,24)
Traceback (most recent call last):
...
ValueError: input must have gcd 1
```

This function is not implemented except for N at most 2**31:

```
sage: lift_to_sl2z(1,1,2^32)
Traceback (most recent call last):
...
NotImplementedError: N too large
```

sage.modular.modsym.p1list.**lift_to_sl2z_int**(*c*, *d*, *N*)

Lift a pair (c, d) to an element of $SL(2, \mathbf{Z})$.

(c, d) is assumed to be an element of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$. This function computes and returns a list $[a, b, c', d']$ that defines a 2x2 matrix, with determinant 1 and integer entries, such that $c = c' \pmod{N}$ and $d = d' \pmod{N}$.

INPUT:

- c,d,N - integers such that $\gcd(c, d, N) = 1$.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import lift_to_sl2z_int
sage: lift_to_sl2z_int(2,6,11)
[1, 8, 2, 17]
sage: m = Matrix(Integers(),2,2,lift_to_sl2z_int(2,6,11)); m
[ 1  8]
[ 2 17]
```

AUTHOR:

- Justin Walker

sage.modular.modsym.p1list.**lift_to_sl2z_llong**(*c*, *d*, *N*)

Lift a pair (c, d) (modulo N) to an element of $SL(2, \mathbf{Z})$.

(c, d) is assumed to be an element of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$. This function computes and returns a list $[a, b, c', d']$ that defines a 2x2 matrix, with determinant 1 and integer entries, such that $c = c' \pmod{N}$ and $d = d' \pmod{N}$.

INPUT:

- c,d,N - integers such that $\gcd(c, d, N) = 1$.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import lift_to_sl2z_llong
sage: lift_to_sl2z_llong(2,6,11)
[1, 8, 2, 17]
sage: m = Matrix(Integers(),2,2,lift_to_sl2z_llong(2,6,11)); m
[ 1  8]
[ 2 17]
```

AUTHOR:

- Justin Walker

sage.modular.modsym.p1list.**p1_normalize**(*N*, *u*, *v*)

Computes the canonical representative of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$ equivalent to (u, v) along with a transforming scalar.

INPUT:

- N - an integer
- u - an integer
- v - an integer

OUTPUT: If gcd(u,v,N) = 1, then returns

- uu - an integer
- vv - an integer
- ss - an integer such that $(ss * uu, ss * vv)$ is equivalent to (u, v) mod N;

  if $\gcd(u, v, N) \neq 1$, returns 0, 0, 0.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1_normalize
sage: p1_normalize(90,7,77)
(1, 11, 7)
sage: p1_normalize(90,7,78)
(1, 24, 7)
sage: (7*24-78*1) % 90
0
sage: (7*24) % 90
78
```

```
sage: from sage.modular.modsym.p1list import p1_normalize
sage: p1_normalize(50001,12345,54322)
(3, 4667, 4115)
sage: (12345*4667-54321*3) % 50001
3
sage: 4115*3 % 50001
12345
sage: 4115*4667 % 50001 == 54322 % 50001
True
```

sage.modular.modsym.p1list.**p1_normalize_int**(*N*, *u*, *v*)

Computes the canonical representative of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$ equivalent to (u, v) along with a transforming scalar.

INPUT:

- N - an integer
- u - an integer
- v - an integer

OUTPUT: If gcd(u,v,N) = 1, then returns

- uu - an integer
- vv - an integer
- ss - an integer such that $(ss * uu, ss * vv)$ is congruent to (u, v) (mod N);

  if $\gcd(u, v, N) \neq 1$, returns 0, 0, 0.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1_normalize_int
sage: p1_normalize_int(90,7,77)
(1, 11, 7)
sage: p1_normalize_int(90,7,78)
(1, 24, 7)
sage: (7*24-78*1) % 90
0
sage: (7*24) % 90
78
```

sage.modular.modsym.p1list.**p1_normalize_llong**(*N*, *u*, *v*)

Computes the canonical representative of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$ equivalent to (u, v) along with a transforming scalar.

INPUT:

- N - an integer
- u - an integer
- v - an integer

OUTPUT: If gcd(u,v,N) = 1, then returns

- uu - an integer
- vv - an integer
- ss - an integer such that $(ss * uu, ss * vv)$ is equivalent to (u, v) mod N;

  if $\gcd(u, v, N) \neq 1$, returns 0, 0, 0.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1_normalize_llong
sage: p1_normalize_llong(90000,7,77)
(1, 11, 7)
sage: p1_normalize_llong(90000,7,78)
(1, 77154, 7)
sage: (7*77154-78*1) % 90000
0
```

(continues on next page)

(continued from previous page)

```
sage: (7*77154) % 90000
78
```

sage.modular.modsym.p1list.**p1list**(N)

Return the elements of the projective line modulo N, $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$, as a plain list of 2-tuples.

INPUT:

- N (integer) - a positive integer (less than 2^31).

OUTPUT:

A list of the elements of the projective line $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$, as plain 2-tuples.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1list
sage: list(p1list(7))
[(0, 1), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
sage: N = 23456
sage: len(p1list(N)) == N*prod([1+1/p for p,e in N.factor()])
True
```

sage.modular.modsym.p1list.**p1list_int**(N)

Return a list of the normalized elements of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$.

INPUT:

- `N` - integer (the level or modulus).

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1list_int
sage: p1list_int(6)
[(0, 1),
(1, 0),
(1, 1),
(1, 2),
(1, 3),
(1, 4),
(1, 5),
(2, 1),
(2, 3),
(2, 5),
(3, 1),
(3, 2)]
```

```
sage: p1list_int(120)
[(0, 1),
(1, 0),
(1, 1),
(1, 2),
(1, 3),
...
(30, 7),
(40, 1),
```

(continues on next page)

(continued from previous page)

```
(40, 3),
(40, 11),
(60, 1)]
```

sage.modular.modsym.p1list.**p1list_llong**(N)

Return a list of the normalized elements of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$, as a plain list of 2-tuples.

INPUT:

- N - integer (the level or modulus).

EXAMPLES:

```
sage: from sage.modular.modsym.p1list import p1list_llong
sage: N = 50000
sage: L = p1list_llong(50000)
sage: len(L) == N*prod([1+1/p for p,e in N.factor()])
True
sage: L[0]
(0, 1)
sage: L[len(L)-1]
(25000, 1)
```

CHAPTER

TWELVE

LIST OF COSET REPRESENTATIVES FOR $\Gamma_1(N)$ IN $\mathrm{SL}_2(\mathbf{Z})$

class sage.modular.modsym.g1list.**G1list**(*N*)

Bases: sage.structure.sage_object.SageObject

A class representing a list of coset representatives for $\Gamma_1(N)$ in $\mathrm{SL}_2(\mathbf{Z})$. What we actually calculate is a list of elements of $(\mathbf{Z}/N\mathbf{Z})^2$ of exact order N.

list()

Return a list of vectors representing the cosets. Do not change the returned list!

EXAMPLES:

```
sage: L = sage.modular.modsym.g1list.G1list(4); L.list()
[(0, 1), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1),
↪ (3, 2), (3, 3)]
```

normalize(*u*, *v*)

Given a pair (u, v) of integers, return the unique pair (u', v') such that the pair (u', v') appears in `self.list()` and (u, v) is equivalent to (u', v'). This is rather trivial, but is here for consistency with the `P1List` class which is the equivalent for Γ_0 (where the problem is rather harder).

This will only make sense if $\gcd(u, v, N) = 1$; otherwise the output will not be an element of self.

EXAMPLES:

```
sage: L = sage.modular.modsym.g1list.G1list(4); L.normalize(6, 1)
(2, 1)
sage: L = sage.modular.modsym.g1list.G1list(4); L.normalize(6, 2) # nonsense!
(2, 2)
```

CHAPTER

THIRTEEN

LIST OF COSET REPRESENTATIVES FOR $\Gamma_H(N)$ IN $\mathrm{SL}_2(\mathbf{Z})$

class sage.modular.modsym.ghlist.**GHlist**(*group*)

Bases: sage.structure.sage_object.SageObject

A class representing a list of coset representatives for $\Gamma_H(N)$ in $\mathrm{SL}_2(\mathbf{Z})$.

list()

Return a list of vectors representing the cosets. Do not change the returned list!

EXAMPLES:

```
sage: L = sage.modular.modsym.ghlist.GHlist(GammaH(4,[])); L.list()
[(0, 1), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 1), (2, 3), (3, 0), (3, 1),
↪ (3, 2), (3, 3)]
```

normalize(*u*, *v*)

Given a pair (u, v) of integers, return the unique pair (u', v') such that the pair (u', v') appears in `self.list()` and (u, v) is equivalent to (u', v').

This will only make sense if $\gcd(u, v, N) = 1$; otherwise the output will not be an element of self.

EXAMPLES:

```
sage: sage.modular.modsym.ghlist.GHlist(GammaH(24, [17, 19])).normalize(17, 6)
(1, 6)
sage: sage.modular.modsym.ghlist.GHlist(GammaH(24, [7, 13])).normalize(17, 6)
(5, 6)
sage: sage.modular.modsym.ghlist.GHlist(GammaH(24, [5, 23])).normalize(17, 6)
(7, 18)
```

CHAPTER

FOURTEEN

RELATION MATRICES FOR AMBIENT MODULAR SYMBOLS SPACES

This file contains functions that are used by the various ambient modular symbols classes to compute presentations of spaces in terms of generators and relations, using the standard methods based on Manin symbols.

`sage.modular.modsym.relation_matrix.T_relation_matrix_wtk_g0(syms, mod, field, sparse)`

Compute a matrix whose echelon form gives the quotient by 3-term T relations. Despite the name, this is used for all modular symbols spaces (including those with character and those for Γ_1 and Γ_H groups), not just Γ_0.

INPUT:

- `syms` – *ManinSymbolList*
- `mod` - list that gives quotient modulo some two-term relations, i.e., the S relations, and if sign is nonzero, the I relations.
- `field` - base_ring
- `sparse` - (True or False) whether to use sparse rather than dense linear algebra

OUTPUT: A sparse matrix whose rows correspond to the reduction of the T relations modulo the S and I relations.

EXAMPLES:

```
sage: from sage.modular.modsym.relation_matrix import sparse_2term_quotient, T_
→relation_matrix_wtk_g0, modS_relations
sage: L = sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma_h(GammaH(36,␣
→[17,19]), 2)
sage: modS = sparse_2term_quotient(modS_relations(L), 216, QQ)
sage: T_relation_matrix_wtk_g0(L, modS, QQ, False)
72 x 216 dense matrix over Rational Field (use the '.str()' method to see the␣
→entries)
sage: T_relation_matrix_wtk_g0(L, modS, GF(17), True)
72 x 216 sparse matrix over Finite Field of size 17 (use the '.str()' method to see␣
→the entries)
```

`sage.modular.modsym.relation_matrix.compute_presentation(syms, sign, field, sparse=None)`

Compute the presentation for self, as a quotient of Manin symbols modulo relations.

INPUT:

- `syms` – *ManinSymbolList*
- `sign` - integer (-1, 0, 1)
- `field` - a field

OUTPUT:

- sparse matrix whose rows give each generator in terms of a basis for the quotient

- list of integers that give the basis for the quotient
- mod: list where mod[i]=(j,s) means that x_i = s*x_j modulo the 2-term S (and possibly I) relations.

ALGORITHM:

1. Let $S = [0, -1; 1, 0], T = [0, -1; 1, -1]$, and $I = [-1, 0; 0, 1]$.
2. Let $x_0, \ldots, x_{n-1}$ by a list of all non-equivalent Manin symbols.
3. Form quotient by 2-term S and (possibly) I relations.
4. Create a sparse matrix A with m columns, whose rows encode the relations

$$[x_i] + [x_i T] + [x_i T^2] = 0.$$

   There are about n such rows. The number of nonzero entries per row is at most 3*(k-1). Note that we must include rows for *all* i, since even if $[x_i] = [x_j]$, it need not be the case that $[x_i T] = [x_j T]$, since S and T do not commute. However, in many cases we have an a priori formula for the dimension of the quotient by all these relations, so we can omit many relations and just check that there are enough at the end–if there aren't, we add in more.

5. Compute the reduced row echelon form of A using sparse Gaussian elimination.
6. Use what we've done above to read off a sparse matrix R that uniquely expresses each of the n Manin symbols in terms of a subset of Manin symbols, modulo the relations. This subset of Manin symbols is a basis for the quotient by the relations.

EXAMPLES:

```
sage: L = sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma0(8,2)
sage: sage.modular.modsym.relation_matrix.compute_presentation(L, 1, GF(9,'a'),␣
↪True)
(
[2 0 0]
[1 0 0]
[0 0 0]
[0 2 0]
[0 0 0]
[0 0 2]
[0 0 0]
[0 2 0]
[0 0 0]
[0 1 0]
[0 1 0]
[0 0 1], [1, 9, 11], [(1, 2), (1, 1), (0, 0), (9, 2), (0, 0), (11, 2), (0, 0), (9,␣
↪2), (0, 0), (9, 1), (9, 1), (11, 1)]
)
```

sage.modular.modsym.relation_matrix.**gens_to_basis_matrix**(*syms*, *relation_matrix*, *mod*, *field*, *sparse*)

Compute echelon form of 3-term relation matrix, and read off each generator in terms of basis.

INPUT:

- `syms` – *ManinSymbolList*
- `relation_matrix` - as output by `__compute_T_relation_matrix(self, mod)`
- `mod` - quotient of modular symbols modulo the 2-term S (and possibly I) relations
- `field` - base field

- sparse - (bool): whether or not matrix should be sparse

OUTPUT:

- matrix - a matrix whose ith row expresses the Manin symbol generators in terms of a basis of Manin symbols (modulo the S, (possibly I,) and T rels) Note that the entries of the matrix need not be integers.
- list - integers i, such that the Manin symbols x_i are a basis.

EXAMPLES:

```
sage: from sage.modular.modsym.relation_matrix import sparse_2term_quotient, T_
↪relation_matrix_wtk_g0, gens_to_basis_matrix, modS_relations
sage: L = sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma1(4, 3)
sage: modS = sparse_2term_quotient(modS_relations(L), 24, GF(3))
sage: gens_to_basis_matrix(L, T_relation_matrix_wtk_g0(L, modS, GF(3), 24), modS,␣
↪GF(3), True)
(24 x 2 sparse matrix over Finite Field of size 3, [13, 23])
```

sage.modular.modsym.relation_matrix.**modI_relations**(*syms*, *sign*)

Compute quotient of Manin symbols by the I relations.

INPUT:

- syms – *ManinSymbolList*
- sign - int (either -1, 0, or 1)

OUTPUT:

- rels - set of pairs of pairs (j, s), where if mod[i] = (j,s), then x_i = s*x_j (mod S relations)

EXAMPLES:

```
sage: L = sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma1(4, 3)
sage: sage.modular.modsym.relation_matrix.modI_relations(L, 1)
{((0, 1), (0, -1)),
 ((1, 1), (1, -1)),
 ((2, 1), (8, -1)),
 ((3, 1), (9, -1)),
 ((4, 1), (10, -1)),
 ((5, 1), (11, -1)),
 ((6, 1), (6, -1)),
 ((7, 1), (7, -1)),
 ((8, 1), (2, -1)),
 ((9, 1), (3, -1)),
 ((10, 1), (4, -1)),
 ((11, 1), (5, -1)),
 ((12, 1), (12, 1)),
 ((13, 1), (13, 1)),
 ((14, 1), (20, 1)),
 ((15, 1), (21, 1)),
 ((16, 1), (22, 1)),
 ((17, 1), (23, 1)),
 ((18, 1), (18, 1)),
 ((19, 1), (19, 1)),
 ((20, 1), (14, 1)),
 ((21, 1), (15, 1)),
```

(continues on next page)

(continued from previous page)

```
((22, 1), (16, 1)),
((23, 1), (17, 1))}
```

Warning: We quotient by the involution eta((u,v)) = (-u,v), which has the opposite sign as the involution in Merel's Springer LNM 1585 paper! Thus our +1 eigenspace is his -1 eigenspace, etc. We do this for consistency with MAGMA.

`sage.modular.modsym.relation_matrix.modS_relations(syms)`

Compute quotient of Manin symbols by the S relations.

Here S is the 2x2 matrix [0, -1; 1, 0].

INPUT:

- `syms` – *ManinSymbolList*

OUTPUT:

- `rels` - set of pairs of pairs (j, s), where if mod[i] = (j,s), then x_i = s*x_j (mod S relations)

EXAMPLES:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma0
sage: from sage.modular.modsym.relation_matrix import modS_relations
```

```
sage: syms = ManinSymbolList_gamma0(2, 4); syms
Manin Symbol List of weight 4 for Gamma0(2)
sage: modS_relations(syms)
{((0, 1), (7, 1)),
 ((1, 1), (6, 1)),
 ((2, 1), (8, 1)),
 ((3, -1), (4, 1)),
 ((3, 1), (4, -1)),
 ((5, -1), (5, 1))}
```

```
sage: syms = ManinSymbolList_gamma0(7, 2); syms
Manin Symbol List of weight 2 for Gamma0(7)
sage: modS_relations(syms)
{((0, 1), (1, 1)), ((2, 1), (7, 1)), ((3, 1), (4, 1)), ((5, 1), (6, 1))}
```

Next we do an example with Gamma1:

```
sage: from sage.modular.modsym.manin_symbol_list import ManinSymbolList_gamma1
sage: syms = ManinSymbolList_gamma1(3,2); syms
Manin Symbol List of weight 2 for Gamma1(3)
sage: modS_relations(syms)
{((0, 1), (2, 1)),
 ((0, 1), (5, 1)),
 ((1, 1), (2, 1)),
 ((1, 1), (5, 1)),
 ((3, 1), (4, 1)),
 ((3, 1), (6, 1)),
```

(continues on next page)

(continued from previous page)

```
((4, 1), (7, 1)),
((6, 1), (7, 1))}
```

sage.modular.modsym.relation_matrix.**relation_matrix_wtk_g0**(*syms*, *sign*, *field*, *sparse*)

Compute the matrix of relations. Despite the name, this is used for all spaces (not just for Gamma0). For a description of the algorithm, see the docstring for `compute_presentation`.

INPUT:

- `syms` – *ManinSymbolList*
- `sign`: integer (0, 1 or -1)
- `field`: the base field (non-field base rings not supported at present)
- `sparse`: (True or False) whether to use sparse arithmetic.

Note that ManinSymbolList objects already have a specific weight, so there is no need for an extra `weight` parameter.

OUTPUT: a pair (R, mod) where

- R is a matrix as output by `T_relation_matrix_wtk_g0`
- mod is a set of 2-term relations as output by `sparse_2term_quotient`

EXAMPLES:

```
sage: L = sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma0(8,2)
sage: A = sage.modular.modsym.relation_matrix.relation_matrix_wtk_g0(L, 0, GF(2),␣
↪True); A
(
[0 0 0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 0 1 1 1 0]
[0 0 0 0 0 0 1 0 0 1 1 0]
[0 0 0 0 0 0 1 0 0 0 0 0], [(1, 1), (1, 1), (8, 1), (10, 1), (6, 1), (11, 1), (6,␣
↪1), (9, 1), (8, 1), (9, 1), (10, 1), (11, 1)]
)
sage: A[0].is_sparse()
True
```

sage.modular.modsym.relation_matrix.**sparse_2term_quotient**(*rels*, *n*, *F*)

Perform Sparse Gauss elimination on a matrix all of whose columns have at most 2 nonzero entries. We use an obvious algorithm, which runs fast enough. (Typically making the list of relations takes more time than computing this quotient.) This algorithm is more subtle than just "identify symbols in pairs", since complicated relations can cause generators to surprisingly equal 0.

INPUT:

- `rels` – iterable made of pairs ((i,s), (j,t)). The pair represents the relation s*x_i + t*x_j = 0, where the i, j must be Python int's.
- `n` – int, the x_i are x_0, …, x_n-1.
- `F` – base field

OUTPUT:

- `mod` – list such that mod[i] = (j,s), which means that x_i is equivalent to s*x_j, where the x_j are a basis for the quotient.

EXAMPLES: We quotient out by the relations

$$3 * x0 - x1 = 0, \qquad x1 + x3 = 0, \qquad x2 + x3 = 0, \qquad x4 - x5 = 0$$

to get

```
sage: rels = [((int(0),3), (int(1),-1)), ((int(1),1), (int(3),1)), ((int(2),1),
↪(int(3),1)), ((int(4),1),(int(5),-1))]
sage: n = 6
sage: from sage.modular.modsym.relation_matrix import sparse_2term_quotient
sage: sparse_2term_quotient(rels, n, QQ)
[(3, -1/3), (3, -1), (3, -1), (3, 1), (5, 1), (5, 1)]
```

CHAPTER

FIFTEEN

LISTS OF MANIN SYMBOLS (ELEMENTS OF $\mathbb{P}^1(R/N)$) OVER NUMBER FIELDS

Lists of elements of $\mathbb{P}^1(R/N)$ where R is the ring of integers of a number field K and N is an integral ideal.

AUTHORS:

- Maite Aranes (2009): Initial version

EXAMPLES:

We define a P1NFList:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a^2 - a + 1)
sage: P = P1NFList(N); P
The projective line over the ring of integers modulo the Fractional ideal (5, a^2 - a +
↪1)
```

List operations with the P1NFList:

```
sage: len(P)
26
sage: [p for p in P]
[M-symbol (0: 1) of level Fractional ideal (5, a^2 - a + 1),
...
M-symbol (1: 2*a^2 + 2*a) of level Fractional ideal (5, a^2 - a + 1)]
```

The elements of the P1NFList are M-symbols:

```
sage: type(P[2])
<class 'sage.modular.modsym.p1list_nf.MSymbol'>
```

Definition of MSymbols:

```
sage: alpha = MSymbol(N, 3, a^2); alpha
M-symbol (3: a^2) of level Fractional ideal (5, a^2 - a + 1)
```

Find the index of the class of an M-Symbol $(c : d)$ in the list:

```
sage: i = P.index(alpha)
sage: P[i].c*alpha.d - P[i].d*alpha.c in N
True
```

Lift an MSymbol to a matrix in $SL(2, R)$:

```
sage: alpha = MSymbol(N, a + 2, 3*a^2)
sage: alpha.lift_to_sl2_Ok()
[-3*a^2 + a + 12, 25*a^2 - 50*a + 100, a + 2, a^2 - 3*a + 3]
sage: Ok = k.ring_of_integers()
sage: M = Matrix(Ok, 2, alpha.lift_to_sl2_Ok())
sage: det(M)
1
sage: M[1][1] - alpha.d in N
True
```

Lift an MSymbol from P1NFList to a matrix in $SL(2, R)$

```
sage: P[3]
M-symbol (1: -2*a) of level Fractional ideal (5, a^2 - a + 1)
sage: P.lift_to_sl2_Ok(3)
[0, -1, 1, -2*a]
```

class sage.modular.modsym.p1list_nf.**MSymbol**(*N*, *c*, *d=None*, *check=True*)

Bases: sage.structure.sage_object.SageObject

The constructor for an M-symbol over a number field.

INPUT:

- N – integral ideal (the modulus or level).
- c – integral element of the underlying number field or an MSymbol of level N.
- d – (optional) when present, it must be an integral element such that <c> + <d> + N = R, where R is the corresponding ring of integers.
- check – bool (default True). If check=False the constructor does not check the condition <c> + <d> + N = R.

OUTPUT:

An M-symbol modulo the given ideal N, i.e. an element of the projective line $\mathbb{P}^1(R/N)$, where R is the ring of integers of the underlying number field.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(a + 1, 2)
sage: MSymbol(N, 3, a^2 + 1)
M-symbol (3: a^2 + 1) of level Fractional ideal (2, a + 1)
```

We can give a tuple as input:

```
sage: MSymbol(N, (1, 0))
M-symbol (1: 0) of level Fractional ideal (2, a + 1)
```

We get an error if <c>, <d> and N are not coprime:

```
sage: MSymbol(N, 2*a, a - 1)
Traceback (most recent call last):
...
ValueError: (2*a, a - 1) is not an element of P1(R/N).
sage: MSymbol(N, (0, 0))
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: (0, 0) is not an element of P1(R/N).
```

Saving and loading works:

```
sage: alpha = MSymbol(N, 3, a^2 + 1)
sage: loads(dumps(alpha))==alpha
True
```

N()

Return the level or modulus of this MSymbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
sage: alpha = MSymbol(N, 3, a)
sage: alpha.N()
Fractional ideal (3, 1/2*a - 1/2)
```

c

Return the first coefficient of the M-symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(a + 1, 2)
sage: alpha = MSymbol(N, 3, a^2 + 1)
sage: alpha.c # indirect doctest
3
```

d

Return the second coefficient of the M-symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(a + 1, 2)
sage: alpha = MSymbol(N, 3, a^2 + 1)
sage: alpha.d # indirect doctest
a^2 + 1
```

lift_to_sl2_Ok()

Lift the MSymbol to an element of $SL(2, Ok)$, where Ok is the ring of integers of the corresponding number field.

OUTPUT:

A list of integral elements $[a, b, c', d']$ that are the entries of a 2x2 matrix with determinant 1. The lower two entries are congruent (modulo the level) to the coefficients c, d of the MSymbol self.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
```

(continues on next page)

(continued from previous page)

```
sage: alpha = MSymbol(N, 3*a + 1, a)
sage: alpha.lift_to_sl2_Ok()
[0, -1, 1, a]
```

normalize(*with_scalar=False*)

Return a normalized MSymbol (a canonical representative of an element of $\mathbb{P}^1(R/N)$) equivalent to `self`.

INPUT:

- `with_scalar` – bool (default False)

OUTPUT:

- (only if `with_scalar=True`) a transforming scalar u, such that $(u * c', u * d')$ is congruent to $(c : d)$ (mod N), where $(c : d)$ are the coefficients of `self` and N is the level.
- a normalized MSymbol (c': d') equivalent to `self`.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
sage: alpha1 = MSymbol(N, 3, a); alpha1
M-symbol (3: a) of level Fractional ideal (3, 1/2*a - 1/2)
sage: alpha1.normalize()
M-symbol (0: 1) of level Fractional ideal (3, 1/2*a - 1/2)
sage: alpha2 = MSymbol(N, 4, a + 1)
sage: alpha2.normalize()
M-symbol (1: -a) of level Fractional ideal (3, 1/2*a - 1/2)
```

We get the scaling factor by setting `with_scalar=True`:

```
sage: alpha1.normalize(with_scalar=True)
(a, M-symbol (0: 1) of level Fractional ideal (3, 1/2*a - 1/2))
sage: r, beta1 = alpha1.normalize(with_scalar=True)
sage: r*beta1.c - alpha1.c in N
True
sage: r*beta1.d - alpha1.d in N
True
sage: r, beta2 = alpha2.normalize(with_scalar=True)
sage: r*beta2.c - alpha2.c in N
True
sage: r*beta2.d - alpha2.d in N
True
```

tuple()

Return the MSymbol as a list (c, d).

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
sage: alpha = MSymbol(N, 3, a); alpha
M-symbol (3: a) of level Fractional ideal (3, 1/2*a - 1/2)
sage: alpha.tuple()
(3, a)
```

class sage.modular.modsym.p1list_nf.**P1NFList**(*N*)

Bases: sage.structure.sage_object.SageObject

The class for $\mathbb{P}^1(R/N)$, the projective line modulo N, where R is the ring of integers of a number field K and N is an integral ideal.

INPUT:

- N - integral ideal (the modulus or level).

OUTPUT:

A P1NFList object representing $\mathbb{P}^1(R/N)$.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: P = P1NFList(N); P
The projective line over the ring of integers modulo the Fractional ideal (5, a + 1)
```

Saving and loading works.

```
sage: loads(dumps(P)) == P
True
```

N()

Return the level or modulus of this P1NFList.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 31)
sage: N = k.ideal(5, a + 3)
sage: P = P1NFList(N)
sage: P.N()
Fractional ideal (5, 1/2*a + 3/2)
```

apply_J_epsilon(*i*, *e1*, *e2=1*)

Apply the matrix J_ϵ = [e1, 0, 0, e2] to the i-th M-Symbol of the list.

e1, e2 are units of the underlying number field.

INPUT:

- i – integer
- e1 – unit
- e2 – unit (default 1)

OUTPUT:

integer – the index of the M-Symbol obtained by the right action of the matrix J_ϵ = [e1, 0, 0, e2] on the i-th M-Symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: P = P1NFList(N)
sage: u = k.unit_group().gens_values(); u
```

(continues on next page)

(continued from previous page)

```
[-1, 2*a^2 + 4*a - 1]
sage: P.apply_J_epsilon(4, -1)
2
sage: P.apply_J_epsilon(4, u[0], u[1])
1
```

```
sage: k.<a> = NumberField(x^4 + 13*x - 7)
sage: N = k.ideal(a + 1)
sage: P = P1NFList(N)
sage: u = k.unit_group().gens_values(); u
[-1, -a^3 - a^2 - a - 12, -a^3 - 3*a^2 + 1]
sage: P.apply_J_epsilon(3, u[2]^2)==P.apply_J_epsilon(P.apply_J_epsilon(3,␣
↪u[2]),u[2])
True
```

apply_S(*i*)

Applies the matrix S = [0, -1, 1, 0] to the i-th M-Symbol of the list.

INPUT:

- `i` – integer

OUTPUT:

integer – the index of the M-Symbol obtained by the right action of the matrix S = [0, -1, 1, 0] on the i-th M-Symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: P = P1NFList(N)
sage: j = P.apply_S(P.index_of_normalized_pair(1, 0))
sage: P[j]
M-symbol (0: 1) of level Fractional ideal (5, a + 1)
```

We test that S has order 2:

```
sage: j = randint(0,len(P)-1)
sage: P.apply_S(P.apply_S(j))==j
True
```

apply_TS(*i*)

Applies the matrix TS = [1, -1, 0, 1] to the i-th M-Symbol of the list.

INPUT:

- `i` – integer

OUTPUT:

integer – the index of the M-Symbol obtained by the right action of the matrix TS = [1, -1, 0, 1] on the i-th M-Symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: P = P1NFList(N)
sage: P.apply_TS(3)
2
```

We test that TS has order 3:

```
sage: j = randint(0,len(P)-1)
sage: P.apply_TS(P.apply_TS(P.apply_TS(j)))==j
True
```

apply_T_alpha(*i*, *alpha=1*)

Applies the matrix T_alpha = [1, alpha, 0, 1] to the i-th M-Symbol of the list.

INPUT:

- `i` – integer
- `alpha` – element of the corresponding ring of integers(default 1)

OUTPUT:

integer – the index of the M-Symbol obtained by the right action of the matrix T_alpha = [1, alpha, 0, 1] on the i-th M-Symbol.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: P = P1NFList(N)
sage: P.apply_T_alpha(4, a^ 2 - 2)
3
```

We test that T_a*T_b = T_(a+b):

```
sage: P.apply_T_alpha(3, a^2 - 2)==P.apply_T_alpha(P.apply_T_alpha(3,a^2),-2)
True
```

index(*c*, *d=None*, *with_scalar=False*)

Return the index of the class of the pair (c, d) in the fixed list of representatives of $\mathbb{P}^1(R/N)$.

INPUT:

- `c` – integral element of the corresponding number field, or an MSymbol.
- `d` – (optional) when present, it must be an integral element of the number field such that (c, d) defines an M-symbol of level N.
- `with_scalar` – bool (default False)

OUTPUT:

- `u` - the normalizing scalar (only if `with_scalar=True`)
- `i` - the index of (c, d) in the list.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 31)
sage: N = k.ideal(5, a + 3)
sage: P = P1NFList(N)
sage: P.index(3,a)
5
sage: P[5]==MSymbol(N, 3, a).normalize()
True
```

We can give an MSymbol as input:

```
sage: alpha = MSymbol(N, 3, a)
sage: P.index(alpha)
5
```

We cannot look for the class of an MSymbol of a different level:

```
sage: M = k.ideal(a + 1)
sage: beta = MSymbol(M, 0, 1)
sage: P.index(beta)
Traceback (most recent call last):
...
ValueError: The MSymbol is of a different level
```

If we are interested in the transforming scalar:

```
sage: alpha = MSymbol(N, 3, a)
sage: P.index(alpha, with_scalar=True)
(-a, 5)
sage: u, i = P.index(alpha, with_scalar=True)
sage: (u*P[i].c - alpha.c in N) and (u*P[i].d - alpha.d in N)
True
```

index_of_normalized_pair(*c*, *d=None*)

Return the index of the class (c, d) in the fixed list of representatives of $(P)^1(R/N)$.

INPUT:

- c – integral element of the corresponding number field, or a normalized MSymbol.
- d – (optional) when present, it must be an integral element of the number field such that (c, d) defines a normalized M-symbol of level N.

OUTPUT:

- i - the index of (c, d) in the list.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 31)
sage: N = k.ideal(5, a + 3)
sage: P = P1NFList(N)
sage: P.index_of_normalized_pair(1, 0)
3
sage: j = randint(0,len(P)-1)
sage: P.index_of_normalized_pair(P[j])==j
True
```

lift_to_sl2_Ok(*i*)

Lift the i-th element of this P1NFList to an element of $SL(2, R)$, where R is the ring of integers of the corresponding number field.

INPUT:

- i - integer (index of the element to lift)

OUTPUT:

If the i-th element is $(c : d)$, the function returns a list of integral elements $[a, b, c', d']$ that defines a 2x2 matrix with determinant 1 and such that $c = c' \pmod N$ and $d = d' \pmod N$.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: P = P1NFList(N)
sage: len(P)
16
sage: P[5]
M-symbol (1/2*a + 1/2: -a) of level Fractional ideal (3)
sage: P.lift_to_sl2_Ok(5)
[-a, 2*a - 2, 1/2*a + 1/2, -a]
```

```
sage: Ok = k.ring_of_integers()
sage: L = [Matrix(Ok, 2, P.lift_to_sl2_Ok(i)) for i in range(len(P))]
sage: all(det(L[i]) == 1 for i in range(len(L)))
True
```

list()

Return the underlying list of this P1NFList object.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a+1)
sage: P = P1NFList(N)
sage: type(P)
<class 'sage.modular.modsym.p1list_nf.P1NFList'>
sage: type(P.list())
<... 'list'>
```

normalize(*c*, *d=None*, *with_scalar=False*)

Return a normalised element of $\mathbb{P}^1(R/N)$.

INPUT:

- c – integral element of the underlying number field, or an MSymbol.
- d – (optional) when present, it must be an integral element of the number field such that (c, d) defines an M-symbol of level N.
- with_scalar – bool (default False)

OUTPUT:

- (only if with_scalar=True) a transforming scalar u, such that $(u * c', u * d')$ is congruent to $(c : d)$ (mod N).
- a normalized MSymbol (c’: d’) equivalent to $(c : d)$.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 31)
sage: N = k.ideal(5, a + 3)
sage: P = P1NFList(N)
sage: P.normalize(3, a)
M-symbol (1: 2*a) of level Fractional ideal (5, 1/2*a + 3/2)
```

We can use an MSymbol as input:

```
sage: alpha = MSymbol(N, 3, a)
sage: P.normalize(alpha)
M-symbol (1: 2*a) of level Fractional ideal (5, 1/2*a + 3/2)
```

If we are interested in the normalizing scalar:

```
sage: P.normalize(alpha, with_scalar=True)
(-a, M-symbol (1: 2*a) of level Fractional ideal (5, 1/2*a + 3/2))
sage: r, beta = P.normalize(alpha, with_scalar=True)
sage: (r*beta.c - alpha.c in N) and (r*beta.d - alpha.d in N)
True
```

`sage.modular.modsym.p1list_nf.`**`P1NFList_clear_level_cache`**`()`

Clear the global cache of data for the level ideals.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(a+1)
sage: alpha = MSymbol(N, 2*a^2, 5)
sage: alpha.normalize()
M-symbol (-4*a^2: 5*a^2) of level Fractional ideal (a + 1)
sage: sage.modular.modsym.p1list_nf._level_cache
{Fractional ideal (a + 1): (...)}
sage: sage.modular.modsym.p1list_nf.P1NFList_clear_level_cache()
sage: sage.modular.modsym.p1list_nf._level_cache
{}
```

`sage.modular.modsym.p1list_nf.`**`lift_to_sl2_Ok`**`(N, c, d)`

Lift a pair (c, d) to an element of $SL(2, O_k)$, where O_k is the ring of integers of the corresponding number field.

INPUT:

- `N` – number field ideal
- `c` – integral element of the number field
- `d` – integral element of the number field

OUTPUT:

A list [a, b, c', d'] of integral elements that are the entries of a 2x2 matrix with determinant 1. The lower two entries are congruent to c, d modulo the ideal N.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list_nf import lift_to_sl2_Ok
sage: k.<a> = NumberField(x^2 + 23)
```

(continues on next page)

(continued from previous page)

```
sage: Ok = k.ring_of_integers()
sage: N = k.ideal(3)
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 1, a))
sage: det(M)
1
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 0, a))
sage: det(M)
1
sage: (M[1][0] in N) and (M[1][1] - a in N)
True
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 0, 0))
Traceback (most recent call last):
...
ValueError: Cannot lift (0, 0) to an element of Sl2(Ok).
```

```
sage: k.<a> = NumberField(x^3 + 11)
sage: Ok = k.ring_of_integers()
sage: N = k.ideal(3, a - 1)
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 2*a, 0))
sage: det(M)
1
sage: (M[1][0] - 2*a in N) and (M[1][1] in N)
True
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 4*a^2, a + 1))
sage: det(M)
1
sage: (M[1][0] - 4*a^2 in N) and (M[1][1] - (a+1) in N)
True
```

```
sage: k.<a> = NumberField(x^4 - x^3 -21*x^2 + 17*x + 133)
sage: Ok = k.ring_of_integers()
sage: N = k.ideal(7, a)
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 0, a^2 - 1))
sage: det(M)
1
sage: (M[1][0] in N) and (M[1][1] - (a^2-1) in N)
True
sage: M = Matrix(Ok, 2, lift_to_sl2_Ok(N, 0, 7))
Traceback (most recent call last):
...
ValueError: <0> + <7> and the Fractional ideal (7, a) are not coprime.
```

`sage.modular.modsym.p1list_nf.`**`make_coprime`**`(`*N*, *c*, *d*`)`

Return (c, d’) so d’ is congruent to d modulo N, and such that c and d’ are coprime (<c> + <d’> = R).

INPUT:

- N – number field ideal
- c – integral element of the number field
- d – integral element of the number field

OUTPUT:

A pair (c, d’) where c, d’ are integral elements of the corresponding number field, with d’ congruent to d mod N, and such that <c> + <d’> = R (R being the corresponding ring of integers).

EXAMPLES:

```
sage: from sage.modular.modsym.p1list_nf import make_coprime
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
sage: c = 2*a; d = a + 1
sage: N.is_coprime(k.ideal(c, d))
True
sage: k.ideal(c).is_coprime(d)
False
sage: c, dp = make_coprime(N, c, d)
sage: k.ideal(c).is_coprime(dp)
True
```

sage.modular.modsym.p1list_nf.**p1NFlist**(*N*)

Return a list of the normalized elements of $\mathbb{P}^1(R/N)$, where N is an integral ideal.

INPUT:

- `N` - integral ideal (the level or modulus).

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: from sage.modular.modsym.p1list_nf import p1NFlist, psi
sage: len(p1NFlist(N))==psi(N)
True
```

sage.modular.modsym.p1list_nf.**psi**(*N*)

The index $[\Gamma : \Gamma_0(N)]$, where $\Gamma = GL(2, R)$ for R the corresponding ring of integers, and $\Gamma_0(N)$ standard congruence subgroup.

EXAMPLES:

```
sage: from sage.modular.modsym.p1list_nf import psi
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3, a - 1)
sage: psi(N)
4
```

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(5)
sage: psi(N)
26
```

CHAPTER

SIXTEEN

MONOMIAL EXPANSION OF $(AX + BY)^I (CX + DY)^{J-I}$

class sage.modular.modsym.apply.**Apply**

Bases: `object`

sage.modular.modsym.apply.**apply_to_monomial**(*i*, *j*, *a*, *b*, *c*, *d*)

Return a list of the coefficients of

$$(aX + bY)^i (cX + dY)^{j-i},$$

where $0 \leq i \leq j$, and a, b, c, d are integers.

One should think of j as being $k - 2$ for the application to modular symbols.

INPUT:

- i, j, a, b, c, d – all ints

OUTPUT:

list of ints, which are the coefficients of $Y^j, Y^{j-1}X, \ldots, X^j$, respectively.

EXAMPLES:

We compute that $(X + Y)^2(X - Y) = X^3 + X^2Y - XY^2 - Y^3$:

```
sage: from sage.modular.modsym.apply import apply_to_monomial
sage: apply_to_monomial(2, 3, 1,1,1,-1)
[-1, -1, 1, 1]
sage: apply_to_monomial(5, 8, 1,2,3,4)
[2048, 9728, 20096, 23584, 17200, 7984, 2304, 378, 27]
sage: apply_to_monomial(6,12, 1,1,1,-1)
[1, 0, -6, 0, 15, 0, -20, 0, 15, 0, -6, 0, 1]
```

CHAPTER

SEVENTEEN

SPARSE ACTION OF HECKE OPERATORS

class sage.modular.modsym.hecke_operator.**HeckeOperator**(*parent*, *n*)

Bases: sage.modular.hecke.hecke_operator.HeckeOperator

apply_sparse(*x*)

Return the image of x under self.

If x is not in self.domain(), raise a TypeError.

EXAMPLES:

```
sage: M = ModularSymbols(17,4,-1)
sage: T = M.hecke_operator(4)
sage: T.apply_sparse(M.0)
-27*[X^2,(1,7)] - 167/2*[X^2,(1,9)] - 21/2*[X^2,(1,13)] + 53/2*[X^2,(1,15)]
sage: [T.apply_sparse(x) == T.hecke_module_morphism()(x) for x in M.basis()]
[True, True, True, True]
sage: N = ModularSymbols(17,4,1)
sage: T.apply_sparse(N.0)
Traceback (most recent call last):
...
TypeError: x (=[X^2,(0,1)]) must be in Modular Symbols space
of dimension 4 for Gamma_0(17) of weight 4 with sign -1
over Rational Field
```

CHAPTER

EIGHTEEN

OPTIMIZED CYTHON CODE FOR COMPUTING RELATION MATRICES IN CERTAIN CASES

sage.modular.modsym.relation_matrix_pyx.**sparse_2term_quotient_only_pm1**(*rels*, *n*)

Perform Sparse Gauss elimination on a matrix all of whose columns have at most 2 nonzero entries with relations all 1 or -1.

This algorithm is more subtle than just "identify symbols in pairs", since complicated relations can cause generators to equal 0.

Note: Note the condition on the s,t coefficients in the relations being 1 or -1 for this optimized function. There is a more general function in relation_matrix.py, which is much, much slower.

INPUT:

- `rels` – iterable made of pairs ((i,s), (j,t)). The pair represents the relation s*x_i + t*x_j = 0, where the i, j must be Python int's, and the s,t must all be 1 or -1.
- `n` – int, the x_i are x_0, …, x_n-1.

OUTPUT:

- `mod` – list such that mod[i] = (j,s), which means that x_i is equivalent to s*x_j, where the x_j are a basis for the quotient.

The output depends on the order of the input.

EXAMPLES:

```
sage: from sage.modular.modsym.relation_matrix_pyx import sparse_2term_quotient_
↪only_pm1
sage: rels = [((0,1), (1,-1)), ((1,1), (3,1)), ((2,1),(3,1)), ((4,1),(5,-1))]
sage: n = 6
sage: sparse_2term_quotient_only_pm1(rels, n)
[(3, -1), (3, -1), (3, -1), (3, 1), (5, 1), (5, 1)]
```

CHAPTER

NINETEEN

OVERCONVERGENT MODULAR SYMBOLS

19.1 Pollack-Stevens' Modular Symbols Spaces

This module contains a class for spaces of modular symbols that use Glenn Stevens' conventions, as explained in [PS2011].

There are two main differences between the modular symbols in this directory and the ones in `sage.modular.modsym`:

- There is a shift in the weight: weight $k = 0$ here corresponds to weight $k = 2$ there.
- There is a duality: these modular symbols are functions from $\text{Div}^0(P^1(\mathbf{Q}))$ (cohomological objects), the others are formal linear combinations of $\text{Div}^0(P^1(\mathbf{Q}))$ (homological objects).

EXAMPLES:

First we create the space of modular symbols of weight 0 ($k = 2$) and level 11:

```
sage: M = PollackStevensModularSymbols(Gamma0(11), 0); M
Space of modular symbols for Congruence Subgroup Gamma0(11) with sign 0 and values in␣
↪Sym^0 Q^2
```

One can also create a space of overconvergent modular symbols, by specifying a prime and a precision:

```
sage: M = PollackStevensModularSymbols(Gamma0(11), p = 5, prec_cap = 10, weight = 0); M
Space of overconvergent modular symbols for Congruence Subgroup Gamma0(11) with sign 0␣
↪and values in Space of 5-adic distributions with k=0 action and precision cap 10
```

Currently not much functionality is available on the whole space, and these spaces are mainly used as parents for the modular symbols. These can be constructed from the corresponding classical modular symbols (or even elliptic curves) as follows:

```
sage: A = ModularSymbols(13, sign=1, weight=4).decomposition()[0]
sage: A.is_cuspidal()
True
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_space
sage: f = ps_modsym_from_simple_modsym_space(A); f
Modular symbol of level 13 with values in Sym^2 Q^2
sage: f.values()
[(-13, 0, -1),
 (247/2, 13/2, -6),
 (39/2, 117/2, 42),
 (-39/2, 39, 111/2),
 (-247/2, -117, -209/2)]
```

(continues on next page)

(continued from previous page)

```
sage: f.parent()
Space of modular symbols for Congruence Subgroup Gamma0(13) with sign 1 and values in␣
↪Sym^2 Q^2
```

```
sage: E = EllipticCurve('37a1')
sage: phi = E.pollack_stevens_modular_symbol(); phi
Modular symbol of level 37 with values in Sym^0 Q^2
sage: phi.values()
[0, 1, 0, 0, 0, -1, 1, 0, 0]
sage: phi.parent()
Space of modular symbols for Congruence Subgroup Gamma0(37) with sign 0 and values in␣
↪Sym^0 Q^2
```

class sage.modular.pollack_stevens.space.**PollackStevensModularSymbols_factory**

Bases: sage.structure.factory.UniqueFactory

Create a space of Pollack-Stevens modular symbols.

INPUT:

- group – integer or congruence subgroup
- weight – integer ≥ 0, or None
- sign – integer; -1, 0, 1
- base_ring – ring or None
- p – prime or None
- prec_cap – positive integer or None
- coefficients – the coefficient module (a special type of module, typically distributions), or None

If an explicit coefficient module is given, then the arguments weight, base_ring, prec_cap, and p are redundant and must be None. They are only relevant if coefficients is None, in which case the coefficient module is inferred from the other data.

Note: We emphasize that in the Pollack-Stevens notation, the weight is the usual weight minus 2, so a classical weight 2 modular form corresponds to a modular symbol of "weight 0".

EXAMPLES:

```
sage: M = PollackStevensModularSymbols(Gamma0(7), weight=0, prec_cap = None); M
Space of modular symbols for Congruence Subgroup Gamma0(7) with sign 0 and values␣
↪in Sym^0 Q^2
```

An example with an explicit coefficient module:

```
sage: D = OverconvergentDistributions(3, 7, prec_cap=10)
sage: M = PollackStevensModularSymbols(Gamma0(7), coefficients=D); M
Space of overconvergent modular symbols for Congruence Subgroup Gamma0(7) with sign␣
↪0 and values in Space of 7-adic distributions with k=3 action and precision cap 10
```

create_key(*group*, *weight=None*, *sign=0*, *base_ring=None*, *p=None*, *prec_cap=None*, *coefficients=None*)

Sanitize input.

EXAMPLES:

```
sage: D = OverconvergentDistributions(3, 7, prec_cap=10)
sage: M = PollackStevensModularSymbols(Gamma0(7), coefficients=D) # indirect␣
↪doctest
```

create_object(*version*, *key*)

Create a space of modular symbols from key.

INPUT:

- version – the version of the object to create
- key – a tuple of parameters, as created by *create_key()*

EXAMPLES:

```
sage: D = OverconvergentDistributions(5, 7, 15)
sage: M = PollackStevensModularSymbols(Gamma0(7), coefficients=D) # indirect␣
↪doctest
sage: M2 = PollackStevensModularSymbols(Gamma0(7), coefficients=D) # indirect␣
↪doctest
sage: M is M2
True
```

class sage.modular.pollack_stevens.space.**PollackStevensModularSymbolspace**(*group*, *coefficients*, *sign=0*)

Bases: sage.modules.module.Module

A class for spaces of modular symbols that use Glenn Stevens' conventions. This class should not be instantiated directly by the user: this is handled by the factory object *PollackStevensModularSymbols_factory*.

INPUT:

- group – congruence subgroup
- coefficients – a coefficient module
- sign – (default: 0); 0, -1, or 1

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 11)
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D); M.sign()
0
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D, sign=-1); M.sign()
-1
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D, sign=1); M.sign()
1
```

change_ring(*new_base_ring*)

Change the base ring of this space to new_base_ring.

INPUT:

- new_base_ring – a ring

OUTPUT:

A space of modular symbols over the specified base.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import Symk
sage: D = Symk(4)
sage: M = PollackStevensModularSymbols(Gamma(6), coefficients=D); M
Space of modular symbols for Congruence Subgroup Gamma(6) with sign 0 and␣
↪values in Sym^4 Q^2
sage: M.change_ring(Qp(5,8))
Space of modular symbols for Congruence Subgroup Gamma(6) with sign 0 and␣
↪values in Sym^4 Q_5^2
```

coefficient_module()

Return the coefficient module of this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 11)
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D)
sage: M.coefficient_module()
Space of 11-adic distributions with k=2 action and precision cap 20
sage: M.coefficient_module() is D
True
```

group()

Return the congruence subgroup of this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 5)
sage: G = Gamma0(23)
sage: M = PollackStevensModularSymbols(G, coefficients=D)
sage: M.group()
Congruence Subgroup Gamma0(23)
sage: D = Symk(4)
sage: G = Gamma1(11)
sage: M = PollackStevensModularSymbols(G, coefficients=D)
sage: M.group()
Congruence Subgroup Gamma1(11)
```

level()

Return the level N, where this space is of level $\Gamma_0(N)$.

EXAMPLES:

```
sage: D = OverconvergentDistributions(7, 11)
sage: M = PollackStevensModularSymbols(Gamma1(14), coefficients=D)
sage: M.level()
14
```

ncoset_reps()

Return the number of coset representatives defining the domain of the modular symbols in this space.

OUTPUT:

The number of coset representatives stored in the manin relations. (Just the size of $P^1(\mathbf{Z}/N\mathbf{Z})$)

EXAMPLES:

```
sage: D = Symk(2)
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D)
sage: M.ncoset_reps()
3
```

ngens()

Returns the number of generators defining this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(4, 29)
sage: M = PollackStevensModularSymbols(Gamma1(12), coefficients=D)
sage: M.ngens()
5
sage: D = Symk(2)
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D)
sage: M.ngens()
2
```

precision_cap()

Return the number of moments of each element of this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 5)
sage: M = PollackStevensModularSymbols(Gamma1(13), coefficients=D)
sage: M.precision_cap()
20
sage: D = OverconvergentDistributions(3, 7, prec_cap=10)
sage: M = PollackStevensModularSymbols(Gamma0(7), coefficients=D)
sage: M.precision_cap()
10
```

prime()

Return the prime of this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 11)
sage: M = PollackStevensModularSymbols(Gamma(2), coefficients=D)
sage: M.prime()
11
```

random_element(*M=None*)

Return a random overconvergent modular symbol in this space with M moments

INPUT:

- M – positive integer

OUTPUT:

An element of the modular symbol space with M moments

Returns a random element in this space by randomly choosing values of distributions on all but one divisor, and solves the difference equation to determine the value on the last divisor.

```
sage: D = OverconvergentDistributions(2, 11)
sage: M = PollackStevensModularSymbols(Gamma0(11), coefficients=D)
sage: M.random_element(10)
Traceback (most recent call last):
...
NotImplementedError
```

sign()

Return the sign of this space.

EXAMPLES:

```
sage: D = OverconvergentDistributions(3, 17)
sage: M = PollackStevensModularSymbols(Gamma(5), coefficients=D)
sage: M.sign()
0
sage: D = Symk(4)
sage: M = PollackStevensModularSymbols(Gamma1(8), coefficients=D, sign=-1)
sage: M.sign()
-1
```

source()

Return the domain of the modular symbols in this space.

OUTPUT:

A *sage.modular.pollack_stevens.fund_domain.PollackStevensModularDomain*

EXAMPLES:

```
sage: D = OverconvergentDistributions(2, 11)
sage: M = PollackStevensModularSymbols(Gamma0(2), coefficients=D)
sage: M.source()
Manin Relations of level 2
```

weight()

Return the weight of this space.

> **Warning:** We emphasize that in the Pollack-Stevens notation, this is the usual weight minus 2, so a classical weight 2 modular form corresponds to a modular symbol of "weight 0".

EXAMPLES:

```
sage: D = Symk(5)
sage: M = PollackStevensModularSymbols(Gamma1(7), coefficients=D)
sage: M.weight()
5
```

sage.modular.pollack_stevens.space.**cusps_from_mat**(*g*)

Return the cusps associated to an element of a congruence subgroup.

INPUT:

- g – an element of a congruence subgroup or a matrix

OUTPUT:

A tuple of cusps associated to g.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.space import cusps_from_mat
sage: g = SL2Z.one()
sage: cusps_from_mat(g)
(+Infinity, 0)
```

You can also just give the matrix of g:

```
sage: type(g)
<class 'sage.modular.arithgroup.arithgroup_element.ArithmeticSubgroupElement'>
sage: cusps_from_mat(g.matrix())
(+Infinity, 0)
```

Another example:

```
sage: from sage.modular.pollack_stevens.space import cusps_from_mat
sage: g = GammaH(3, [2]).generators()[1].matrix(); g
[-1  1]
[-3  2]
sage: cusps_from_mat(g)
(1/3, 1/2)
```

sage.modular.pollack_stevens.space.**ps_modsym_from_elliptic_curve**(*E*, *sign=0*, *implementation='eclib'*)

Return the overconvergent modular symbol associated to an elliptic curve defined over the rationals.

INPUT:

- E – an elliptic curve defined over the rationals
- sign – the sign (default: 0). If nonzero, returns either the plus (if sign == 1) or the minus (if sign == -1) modular symbol. The default of 0 returns the sum of the plus and minus symbols.
- implementation – either 'eclib' (default) or 'sage'. This determines which implementation of the underlying classical modular symbols is used.

OUTPUT:

The overconvergent modular symbol associated to E

EXAMPLES:

```
sage: E = EllipticCurve('113a1')
sage: symb = E.pollack_stevens_modular_symbol() # indirect doctest
sage: symb
Modular symbol of level 113 with values in Sym^0 Q^2
sage: symb.values()
[-1/2, 1, -1, 0, 0, 1, 1, -1, 0, -1, 0, 0, 0, 1, -1, 0, 0, 0, 1, 0, 0]

sage: E = EllipticCurve([0,1])
sage: symb = E.pollack_stevens_modular_symbol()
sage: symb.values()
[-1/6, 1/3, 1/2, 1/6, -1/6, 1/3, -1/3, -1/2, -1/6, 1/6, 0, -1/6, -1/6]
```

sage.modular.pollack_stevens.space.**ps_modsym_from_simple_modsym_space**(*A*, *name='alpha'*)

Returns some choice – only well defined up a nonzero scalar (!) – of an overconvergent modular symbol that corresponds to A.

INPUT:

- A – nonzero simple Hecke equivariant new space of modular symbols, which need not be cuspidal.

OUTPUT:

A choice of corresponding overconvergent modular symbols; when dim(A)>1, we make an arbitrary choice of defining polynomial for the codomain field.

EXAMPLES:

The level 11 example:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_
↪space
sage: A = ModularSymbols(11, sign=1, weight=2).decomposition()[0]
sage: A.is_cuspidal()
True
sage: f = ps_modsym_from_simple_modsym_space(A); f
Modular symbol of level 11 with values in Sym^0 Q^2
sage: f.values()
[1, -5/2, -5/2]
sage: f.weight()         # this is A.weight()-2  !!!!!!
0
```

And the -1 sign for the level 11 example:

```
sage: A = ModularSymbols(11, sign=-1, weight=2).decomposition()[0]
sage: f = ps_modsym_from_simple_modsym_space(A); f.values()
[0, 1, -1]
```

A does not have to be cuspidal; it can be Eisenstein:

```
sage: A = ModularSymbols(11, sign=1, weight=2).decomposition()[1]
sage: A.is_cuspidal()
False
sage: f = ps_modsym_from_simple_modsym_space(A); f
Modular symbol of level 11 with values in Sym^0 Q^2
sage: f.values()
[1, 0, 0]
```

We create the simplest weight 2 example in which A has dimension bigger than 1:

```
sage: A = ModularSymbols(23, sign=1, weight=2).decomposition()[0]
sage: f = ps_modsym_from_simple_modsym_space(A); f.values()
[1, 0, 0, 0, 0]
sage: A = ModularSymbols(23, sign=-1, weight=2).decomposition()[0]
sage: f = ps_modsym_from_simple_modsym_space(A); f.values()
[0, 1, -alpha, alpha, -1]
sage: f.base_ring()
Number Field in alpha with defining polynomial x^2 + x - 1
```

We create the +1 modular symbol attached to the weight 12 modular form Delta:

```
sage: A = ModularSymbols(1, sign=+1, weight=12).decomposition()[0]
sage: f = ps_modsym_from_simple_modsym_space(A); f
Modular symbol of level 1 with values in Sym^10 Q^2
sage: f.values()
[(-1620/691, 0, 1, 0, -9/14, 0, 9/14, 0, -1, 0, 1620/691), (1620/691, 1620/691, 929/
↪691, -453/691, -29145/9674, -42965/9674, -2526/691, -453/691, 1620/691, 1620/691,␣
↪0), (0, -1620/691, -1620/691, 453/691, 2526/691, 42965/9674, 29145/9674, 453/691,␣
↪-929/691, -1620/691, -1620/691)]
```

And, the -1 modular symbol attached to `Delta`:

```
sage: A = ModularSymbols(1, sign=-1, weight=12).decomposition()[0]
sage: f = ps_modsym_from_simple_modsym_space(A); f
Modular symbol of level 1 with values in Sym^10 Q^2
sage: f.values()
[(0, 1, 0, -25/48, 0, 5/12, 0, -25/48, 0, 1, 0), (0, -1, -2, -119/48, -23/12, -5/24,
↪ 23/12, 3, 2, 0, 0), (0, 0, 2, 3, 23/12, -5/24, -23/12, -119/48, -2, -1, 0)]
```

A consistency check with *sage.modular.pollack_stevens.space.ps_modsym_from_simple_modsym_space()*:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_
↪space
sage: E = EllipticCurve('11a')
sage: f_E = E.pollack_stevens_modular_symbol(); f_E.values()
[-1/5, 1, 0]
sage: A = ModularSymbols(11, sign=1, weight=2).decomposition()[0]
sage: f_plus = ps_modsym_from_simple_modsym_space(A); f_plus.values()
[1, -5/2, -5/2]
sage: A = ModularSymbols(11, sign=-1, weight=2).decomposition()[0]
sage: f_minus = ps_modsym_from_simple_modsym_space(A); f_minus.values()
[0, 1, -1]
```

We find that a linear combination of the plus and minus parts equals the Pollack-Stevens symbol attached to `E`. This illustrates how `ps_modsym_from_simple_modsym_space` is only well-defined up to a nonzero scalar:

```
sage: (-1/5)*vector(QQ, f_plus.values()) + (1/2)*vector(QQ, f_minus.values())
(-1/5, 1, 0)
sage: vector(QQ, f_E.values())
(-1/5, 1, 0)
```

The next few examples all illustrate the ways in which exceptions are raised if A does not satisfy various constraints.

First, `A` must be new:

```
sage: A = ModularSymbols(33,sign=1).cuspidal_subspace().old_subspace()
sage: ps_modsym_from_simple_modsym_space(A)
Traceback (most recent call last):
...
ValueError: A must be new
```

`A` must be simple:

```
sage: A = ModularSymbols(43,sign=1).cuspidal_subspace()
sage: ps_modsym_from_simple_modsym_space(A)
Traceback (most recent call last):
...
ValueError: A must be simple
```

A must have sign -1 or +1 in order to be simple:

```
sage: A = ModularSymbols(11).cuspidal_subspace()
sage: ps_modsym_from_simple_modsym_space(A)
Traceback (most recent call last):
...
ValueError: A must have sign +1 or -1 (otherwise it is not simple)
```

The dimension must be positive:

```
sage: A = ModularSymbols(10).cuspidal_subspace(); A
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 3 for␣
↪Gamma_0(10) of weight 2 with sign 0 over Rational Field
sage: ps_modsym_from_simple_modsym_space(A)
Traceback (most recent call last):
...
ValueError: A must have positive dimension
```

We check that forms of nontrivial character are getting handled correctly:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_
↪space
sage: f = Newforms(Gamma1(13), names='a')[0]
sage: phi = ps_modsym_from_simple_modsym_space(f.modular_symbols(1))
sage: phi.hecke(7)
Modular symbol of level 13 with values in Sym^0 (Number Field in alpha with␣
↪defining polynomial x^2 + 3*x + 3)^2 twisted by Dirichlet character modulo 13 of␣
↪conductor 13 mapping 2 |--> -alpha - 1
sage: phi.hecke(7).values()
[0, 0, 0, 0, 0]
```

19.2 Spaces of Distributions for Pollack-Stevens modular symbols

The Pollack-Stevens version of modular symbols take values on a $\Sigma_0(N)$-module which can be either a symmetric power of the standard representation of GL2, or a finite approximation module to the module of overconvergent distributions.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import Symk
sage: S = Symk(6); S
Sym^6 Q^2
sage: v = S(list(range(7))); v
(0, 1, 2, 3, 4, 5, 6)
sage: v.act_right([1,2,3,4])
(18432, 27136, 39936, 58752, 86400, 127008, 186624)
```

(continues on next page)

(continued from previous page)

```
sage: S = Symk(4,Zp(5)); S
Sym^4 Z_5^2
sage: S([1,2,3,4,5])
(1 + O(5^20), 2 + O(5^20), 3 + O(5^20), 4 + O(5^20), 5 + O(5^21))
```

```
sage: from sage.modular.pollack_stevens.distributions import OverconvergentDistributions
sage: D = OverconvergentDistributions(3, 11, 5); D
Space of 11-adic distributions with k=3 action and precision cap 5
sage: D([1,2,3,4,5])
(1 + O(11^5), 2 + O(11^4), 3 + O(11^3), 4 + O(11^2), 5 + O(11))
```

class sage.modular.pollack_stevens.distributions.**OverconvergentDistributions_abstract**(*k*, *p=None*, *prec_cap=None*, *base=None*, *character=None*, *adjuster=None*, *act_on_left=False*, *dettwist=None*, *act_padic=False*, *implementation=None*)

Bases: `sage.modules.module.Module`

Parent object for distributions. Not to be used directly, see derived classes *Symk_class* and *OverconvergentDistributions_class*.

INPUT:

- `k` – integer; k is the usual modular forms weight minus 2
- `p` – None or prime
- `prec_cap` – None or positive integer
- `base` – None or the base ring over which to construct the distributions
- `character` – None or Dirichlet character
- `adjuster` – None or a way to specify the action among different conventions
- `act_on_left` – bool (default: False)
- `dettwist` – None or integer (twist by determinant). Ignored for Symk spaces
- `act_padic` – bool (default: False) If true, will allow action by p-adic matrices.
- `implementation` – string (default: None) Either automatic (if None), 'vector' or 'long'.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions
sage: OverconvergentDistributions(2, 17, 100)
Space of 17-adic distributions with k=2 action and precision cap 100

sage: D = OverconvergentDistributions(2, 3, 5); D
Space of 3-adic distributions with k=2 action and precision cap 5
sage: type(D)
<class 'sage.modular.pollack_stevens.distributions.OverconvergentDistributions_
↪class_with_category'>
```

acting_matrix(*g*, *M*)

Return the matrix for the action of g on `self`, truncated to the first M moments.

EXAMPLES:

```
sage: V = Symk(3)
sage: from sage.modular.pollack_stevens.sigma0 import Sigma0
sage: V.acting_matrix(Sigma0(1)([3,4,0,1]), 4)
[27 36 48 64]
[ 0  9 24 48]
[ 0  0  3 12]
[ 0  0  0  1]

sage: from sage.modular.btquotients.pautomorphicform import _btquot_adjuster
sage: V = Symk(3, adjuster = _btquot_adjuster())
sage: from sage.modular.pollack_stevens.sigma0 import Sigma0
sage: V.acting_matrix(Sigma0(1)([3,4,0,1]), 4)
[  1   4  16  64]
[  0   3  24 144]
[  0   0   9 108]
[  0   0   0  27]
```

approx_module(*M=None*)

Return the M-th approximation module, or if M is not specified, return the largest approximation module.

INPUT:

- `M` – None or nonnegative integer that is at most the precision cap

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions
sage: D = OverconvergentDistributions(0, 5, 10)
sage: D.approx_module()
Ambient free module of rank 10 over the principal ideal domain 5-adic Ring with␣
↪capped absolute precision 10
sage: D.approx_module(1)
Ambient free module of rank 1 over the principal ideal domain 5-adic Ring with␣
↪capped absolute precision 10
sage: D.approx_module(0)
Ambient free module of rank 0 over the principal ideal domain 5-adic Ring with␣
↪capped absolute precision 10
```

Note that `M` must be at most the precision cap, and must be nonnegative:

```
sage: D.approx_module(11)
Traceback (most recent call last):
...
ValueError: M (=11) must be less than or equal to the precision cap (=10)
sage: D.approx_module(-1)
Traceback (most recent call last):
...
ValueError: rank (=-1) must be nonnegative
```

basis(*M=None*)

Return a basis for this space of distributions.

INPUT:

- M – (Default: None) If not None, specifies the M-th approximation module, in case that this makes sense.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 4); D
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D.basis()
[(1 + O(7^4), O(7^3), O(7^2), O(7)),
 (O(7^4), 1 + O(7^3), O(7^2), O(7)),
 (O(7^4), O(7^3), 1 + O(7^2), O(7)),
 (O(7^4), O(7^3), O(7^2), 1 + O(7))]
sage: D.basis(2)
[(1 + O(7^2), O(7)), (O(7^2), 1 + O(7))]
sage: D = Symk(3, base=QQ); D
Sym^3 Q^2
sage: D.basis()
[(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)]
sage: D.basis(2)
Traceback (most recent call last):
...
ValueError: Sym^k objects do not support approximation modules
```

clear_cache()

Clear some caches that are created only for speed purposes.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 10)
sage: D.clear_cache()
```

lift(*p=None*, *M=None*, *new_base_ring=None*)

Return distribution space that contains lifts with given p, precision cap M, and base ring new_base_ring.

INPUT:

- p – prime or None
- M – nonnegative integer or None

- `new_base_ring` – ring or None

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = Symk(0, Qp(7)); D
Sym^0 Q_7^2
sage: D.lift(M=20)
Space of 7-adic distributions with k=0 action and precision cap 20
sage: D.lift(p=7, M=10)
Space of 7-adic distributions with k=0 action and precision cap 10
sage: D.lift(p=7, M=10, new_base_ring=QpCR(7,15)).base_ring()
7-adic Field with capped relative precision 15
```

precision_cap()

Return the precision cap on distributions.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 10); D
Space of 7-adic distributions with k=0 action and precision cap 10
sage: D.precision_cap()
10
sage: D = Symk(389, base=QQ); D
Sym^389 Q^2
sage: D.precision_cap()
390
```

prime()

Return prime p such that this is a space of p-adic distributions.

In case this space is Symk of a non-padic field, we return 0.

OUTPUT:

- a prime or 0

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7); D
Space of 7-adic distributions with k=0 action and precision cap 20
sage: D.prime()
7
sage: D = Symk(4, base=GF(7)); D
Sym^4 (Finite Field of size 7)^2
sage: D.prime()
0
```

But Symk of a p-adic field does work:

```
sage: D = Symk(4, base=Qp(7)); D
Sym^4 Q_7^2
```

(continues on next page)

(continued from previous page)

```
sage: D.prime()
7
sage: D.is_symk()
True
```

random_element(*M=None*, ***args*)

Return a random element of the M-th approximation module with non-negative valuation.

INPUT:

- `M` – None or a nonnegative integer

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions
sage: D = OverconvergentDistributions(0, 5, 10)
sage: D.random_element()
(..., ..., ..., ..., ..., ..., ..., ..., ..., ...)
sage: D.random_element(0)
()
sage: D.random_element(5)
(..., ..., ..., ..., ...)
sage: D.random_element(-1)
Traceback (most recent call last):
...
ValueError: rank (=-1) must be nonnegative
sage: D.random_element(11)
Traceback (most recent call last):
...
ValueError: M (=11) must be less than or equal to the precision cap (=10)
```

weight()

Return the weight of this distribution space. The standard caveat applies, namely that the weight of Sym^k is defined to be k, not $k + 2$.

OUTPUT:

- nonnegative integer

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7); D
Space of 7-adic distributions with k=0 action and precision cap 20
sage: D.weight()
0
sage: OverconvergentDistributions(389, 7).weight()
389
```

class sage.modular.pollack_stevens.distributions.**OverconvergentDistributions_class**(*k*, *p=None*, *prec_cap=None*, *base=None*, *character=None*, *adjuster=None*, *act_on_left=False*, *dettwist=None*, *act_padic=False*, *implementation=None*)

Bases: *sage.modular.pollack_stevens.distributions.OverconvergentDistributions_abstract*

The class of overconvergent distributions

This class represents the module of finite approximation modules, which are finite-dimensional spaces with a $\Sigma_0(N)$ action which approximate the module of overconvergent distributions. There is a specialization map to the finite-dimensional Symk module as well.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions
sage: D = OverconvergentDistributions(0, 5, 10)
sage: TestSuite(D).run()
```

change_ring(*new_base_ring*)

Return space of distributions like this one, but with the base ring changed.

INPUT: a ring over which the distribution can be coerced.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 4); D
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D.base_ring()
7-adic Ring with capped absolute precision 4
sage: D2 = D.change_ring(QpCR(7)); D2
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D2.base_ring()
7-adic Field with capped relative precision 20
```

is_symk()

Whether or not this distributions space is $Sym^k(R)$ for some ring R.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(4, 17, 10); D
Space of 17-adic distributions with k=4 action and precision cap 10
```

(continues on next page)

(continued from previous page)

```
sage: D.is_symk()
False
sage: D = Symk(4); D
Sym^4 Q^2
sage: D.is_symk()
True
sage: D = Symk(4, base=GF(7)); D
Sym^4 (Finite Field of size 7)^2
sage: D.is_symk()
True
```

specialize(*new_base_ring=None*)

Return distribution space got by specializing to Sym^k, over the `new_base_ring`. If `new_base_ring` is not given, use current `base_ring`.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 4); D
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D.is_symk()
False
sage: D2 = D.specialize(); D2
Sym^0 Z_7^2
sage: D2.is_symk()
True
sage: D2 = D.specialize(QQ); D2
Sym^0 Q^2
```

class sage.modular.pollack_stevens.distributions.**OverconvergentDistributions_factory**

Bases: `sage.structure.factory.UniqueFactory`

Create a space of distributions.

INPUT:

- `k` – nonnegative integer
- `p` – prime number or None
- `prec_cap` – positive integer or None
- `base` – ring or None
- `character` – a Dirichlet character or None
- `adjuster` – None or callable that turns 2 x 2 matrices into a 4-tuple
- `act_on_left` – bool (default: False)
- `dettwist` – integer or None (interpreted as 0)
- `act_padic` – whether monoid should allow p-adic coefficients
- `implementation` – string (default: None). Either None (for automatic), 'long', or 'vector'

EXAMPLES:

```
sage: D = OverconvergentDistributions(3, 11, 20)
sage: D
Space of 11-adic distributions with k=3 action and precision cap 20
sage: v = D([1,0,0,0,0])
sage: v.act_right([2,1,0,1])
(8 + O(11^5), 4 + O(11^4), 2 + O(11^3), 1 + O(11^2), 6 + O(11))
```

```
sage: D = OverconvergentDistributions(3, 11, 20, dettwist=1)
sage: v = D([1,0,0,0,0])
sage: v.act_right([2,1,0,1])
(5 + 11 + O(11^5), 8 + O(11^4), 4 + O(11^3), 2 + O(11^2), 1 + O(11))
```

create_key(*k, p=None, prec_cap=None, base=None, character=None, adjuster=None, act_on_left=False, dettwist=None, act_padic=False, implementation=None*)

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions
sage: OverconvergentDistributions(20, 3, 10)              # indirect doctest
Space of 3-adic distributions with k=20 action and precision cap 10
sage: TestSuite(OverconvergentDistributions).run()
```

create_object(*version, key*)

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: OverconvergentDistributions(0, 7, 5)              # indirect doctest
Space of 7-adic distributions with k=0 action and precision cap 5
```

class sage.modular.pollack_stevens.distributions.**Symk_class**(*k, base, character, adjuster, act_on_left, dettwist, act_padic, implementation*)

Bases: *sage.modular.pollack_stevens.distributions.OverconvergentDistributions_abstract*

EXAMPLES:

```
sage: D = sage.modular.pollack_stevens.distributions.Symk(4); D
Sym^4 Q^2
sage: TestSuite(D).run() # indirect doctest
```

base_extend(*new_base_ring*)

Extend scalars to a new base ring.

EXAMPLES:

```
sage: Symk(3).base_extend(Qp(3))
Sym^3 Q_3^2
```

change_ring(*new_base_ring*)

Return a Symk with the same k but a different base ring.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(0, 7, 4); D
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D.base_ring()
7-adic Ring with capped absolute precision 4
sage: D2 = D.change_ring(QpCR(7)); D2
Space of 7-adic distributions with k=0 action and precision cap 4
sage: D2.base_ring()
7-adic Field with capped relative precision 20
```

is_symk()

Whether or not this distributions space is $Sym^k(R)$ for some ring R.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import␣
↪OverconvergentDistributions, Symk
sage: D = OverconvergentDistributions(4, 17, 10); D
Space of 17-adic distributions with k=4 action and precision cap 10
sage: D.is_symk()
False
sage: D = Symk(4); D
Sym^4 Q^2
sage: D.is_symk()
True
sage: D = Symk(4, base=GF(7)); D
Sym^4 (Finite Field of size 7)^2
sage: D.is_symk()
True
```

class sage.modular.pollack_stevens.distributions.**Symk_factory**

Bases: sage.structure.factory.UniqueFactory

Create the space of polynomial distributions of degree k (stored as a sequence of $k + 1$ moments).

INPUT:

- k - (integer): the degree (degree k corresponds to weight $k + 2$ modular forms)
- base - (ring, default None): the base ring (None is interpreted as $\mathbf{Q}$)
- character - (Dirichlet character or None, default None) the character
- adjuster - (None or a callable that turns 2×2 matrices into a 4-tuple, default None)
- act_on_left - (boolean, default False) whether to have the group acting on the left rather than the right.
- dettwist (integer or None) – power of determinant to twist by

EXAMPLES:

```
sage: D = Symk(4)
sage: loads(dumps(D)) is D
True
sage: loads(dumps(D)) == D
True
sage: from sage.modular.pollack_stevens.distributions import Symk
```

(continues on next page)

(continued from previous page)

```
sage: Symk(5)
Sym^5 Q^2
sage: Symk(5, RR)
Sym^5 (Real Field with 53 bits of precision)^2
sage: Symk(5, oo.parent()) # don't do this
Sym^5 (The Infinity Ring)^2
sage: Symk(5, act_on_left = True)
Sym^5 Q^2
```

The `dettwist` attribute:

```
sage: V = Symk(6)
sage: v = V([1,0,0,0,0,0,0])
sage: v.act_right([2,1,0,1])
(64, 32, 16, 8, 4, 2, 1)
sage: V = Symk(6, dettwist=-1)
sage: v = V([1,0,0,0,0,0,0])
sage: v.act_right([2,1,0,1])
(32, 16, 8, 4, 2, 1, 1/2)
```

create_key(*k*, *base=None*, *character=None*, *adjuster=None*, *act_on_left=False*, *dettwist=None*, *act_padic=False*, *implementation=None*)

Sanitize input.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import Symk
sage: Symk(6) # indirect doctest
Sym^6 Q^2

sage: V = Symk(6, Qp(7))
sage: TestSuite(V).run()
```

create_object(*version*, *key*)

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.distributions import Symk
sage: Symk(6) # indirect doctest
Sym^6 Q^2
```

19.3 Manin Relations for overconvergent modular symbols

Code to create the Manin Relations class, which solves the "Manin relations". That is, a description of $Div^0(P^1(\mathbf{Q}))$ as a $\mathbf{Z}[\Gamma_0(N)]$-module in terms of generators and relations is found. The method used is geometric, constructing a nice fundamental domain for $\Gamma_0(N)$ and reading the relevant Manin relations off of that picture. The algorithm follows [PS2011].

AUTHORS:

- Robert Pollack, Jonathan Hanke (2012): initial version

`sage.modular.pollack_stevens.fund_domain.`**M2Z**(*x*)

Create an immutable 2×2 integer matrix from `x`.

INPUT: anything that can be converted into a 2×2 matrix.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import M2Z
sage: M2Z([1,2,3,4])
[1 2]
[3 4]
sage: M2Z(1)
[1 0]
[0 1]
```

class sage.modular.pollack_stevens.fund_domain.**ManinRelations**(*N*)

Bases: *sage.modular.pollack_stevens.fund_domain.PollackStevensModularDomain*

This class gives a description of $Div^0(P^1(\mathbf{Q}))$ as a $\mathbf{Z}[\Gamma_0(N)]$-module.

INPUT:

- N – a positive integer, the level of $\Gamma_0(N)$ to work with

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: ManinRelations(1)
Manin Relations of level 1
sage: ManinRelations(11)
Manin Relations of level 11
```

Large values of N are not supported:

```
sage: ManinRelations(2^20)
Traceback (most recent call last):
...
OverflowError: Modulus is too large (must be <= 46340)
```

fd_boundary(*C*)

Find matrices whose associated unimodular paths give the boundary of a fundamental domain.

Here the fundamental domain is for $\Gamma_0(N)$. (In the case when $\Gamma_0(N)$ has elements of order three the shape cut out by these unimodular matrices is a little smaller than a fundamental domain. See Section 2.5 of [PS2011].)

INPUT:

- C – a list of rational numbers coming from self.form_list_of_cusps()

OUTPUT:

A list of 2×2 integer matrices of determinant 1 whose associated unimodular paths give the boundary of a fundamental domain for $\Gamma_0(N)$ (or nearly so in the case of 3-torsion).

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: C = A.form_list_of_cusps(); C
[-1, -2/3, -1/2, -1/3, 0]
sage: A.fd_boundary(C)
```

(continues on next page)

(continued from previous page)

```
[
[1 0]  [ 1  1]  [ 0 -1]  [-1 -1]  [-1 -2]  [-2 -1]
[0 1], [-1  0], [ 1  3], [ 3  2], [ 2  3], [ 3  1]
]
sage: A = ManinRelations(13)
sage: C = A.form_list_of_cusps(); C
[-1, -2/3, -1/2, -1/3, 0]
sage: A.fd_boundary(C)
[
[1 0]  [ 1  1]  [ 0 -1]  [-1 -1]  [-1 -2]  [-2 -1]
[0 1], [-1  0], [ 1  3], [ 3  2], [ 2  3], [ 3  1]
]
sage: A = ManinRelations(101)
sage: C = A.form_list_of_cusps(); C
[-1, -6/7, -5/6, -4/5, -7/9, -3/4, -11/15, -8/11, -5/7, -7/10,
-9/13, -2/3, -5/8, -13/21, -8/13, -3/5, -7/12, -11/19, -4/7, -1/2,
-4/9, -3/7, -5/12, -7/17, -2/5, -3/8, -4/11, -1/3, -2/7, -3/11,
-1/4, -2/9, -1/5, -1/6, 0]
sage: A.fd_boundary(C)
[
[1 0]  [ 1  1]  [ 0 -1]  [-1 -1]  [-1 -2]  [-2 -1]  [-1 -3]  [-3 -2]
[0 1], [-1  0], [ 1  6], [ 6  5], [ 5  9], [ 9  4], [ 4 11], [11  7],

[-2 -1]  [-1 -4]  [-4 -3]  [-3 -2]  [-2 -7]  [-7 -5]  [-5 -3]  [-3 -4]
[ 7  3], [ 3 11], [11  8], [ 8  5], [ 5 17], [17 12], [12  7], [ 7  9],

[-4 -1]  [-1 -4]  [ -4 -11]  [-11  -7]  [-7 -3]  [-3 -8]  [ -8 -13]
[ 9  2], [ 2  7], [  7  19], [ 19  12], [12  5], [ 5 13], [ 13  21],

[-13  -5]  [-5 -2]  [-2 -9]  [-9 -7]  [-7 -5]  [-5 -8]  [ -8 -11]
[ 21   8], [ 8  3], [ 3 13], [13 10], [10  7], [ 7 11], [ 11  15],

[-11  -3]  [-3 -7]  [-7 -4]  [-4 -5]  [-5 -6]  [-6 -1]
[ 15   4], [ 4  9], [ 9  5], [ 5  6], [ 6  7], [ 7  1]
]
```

form_list_of_cusps()

Return the intersection of a fundamental domain for $\Gamma_0(N)$ with the real axis.

The construction of this fundamental domain follows the arguments of [PS2011] Section 2. The boundary of this fundamental domain consists entirely of unimodular paths when $\Gamma_0(N)$ has no elements of order 3. (See [PS2011] Section 2.5 for the case when there are elements of order 3.)

OUTPUT:

A sorted list of rational numbers marking the intersection of a fundamental domain for $\Gamma_0(N)$ with the real axis.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.form_list_of_cusps()
[-1, -2/3, -1/2, -1/3, 0]
```

(continues on next page)

(continued from previous page)

```
sage: A = ManinRelations(13)
sage: A.form_list_of_cusps()
[-1, -2/3, -1/2, -1/3, 0]
sage: A = ManinRelations(101)
sage: A.form_list_of_cusps()
[-1, -6/7, -5/6, -4/5, -7/9, -3/4, -11/15, -8/11, -5/7, -7/10,
-9/13, -2/3, -5/8, -13/21, -8/13, -3/5, -7/12, -11/19, -4/7, -1/2,
-4/9, -3/7, -5/12, -7/17, -2/5, -3/8, -4/11, -1/3, -2/7, -3/11,
-1/4, -2/9, -1/5, -1/6, 0]
```

indices_with_three_torsion()

A list of indices of coset representatives whose associated unimodular path contains a point fixed by a $\Gamma_0(N)$ element of order 3 in the ideal triangle directly below that path (the order is computed in $PSL_2(\mathbf{Z})$).

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(11)
sage: MR.indices_with_three_torsion()
[]
sage: MR = ManinRelations(13)
sage: MR.indices_with_three_torsion()
[2, 5]
sage: B = MR.reps(2); B
[ 0 -1]
[ 1  3]
```

The corresponding matrix of order three:

```
sage: A = MR.three_torsion_matrix(B); A
[-4 -1]
[13  3]
sage: A^3
[1 0]
[0 1]
```

The columns of `B` and the columns of `A*B` and `A^2*B` give the same rational cusps:

```
sage: B
[ 0 -1]
[ 1  3]
sage: A*B, A^2*B
(
[-1  1]  [ 1  0]
[ 3 -4], [-4  1]
)
```

indices_with_two_torsion()

Return the indices of coset representatives whose associated unimodular path contains a point fixed by a $\Gamma_0(N)$ element of order 2 (where the order is computed in $PSL_2(\mathbf{Z})$).

OUTPUT:

A list of integers.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(11)
sage: MR.indices_with_two_torsion()
[]
sage: MR = ManinRelations(13)
sage: MR.indices_with_two_torsion()
[3, 4]
sage: MR.reps(3), MR.reps(4)
(
[-1 -1]  [-1 -2]
[ 3  2], [ 2  3]
)
```

The corresponding matrix of order 2:

```
sage: A = MR.two_torsion_matrix(MR.reps(3)); A
[  5   2]
[-13  -5]
sage: A^2
[-1  0]
[ 0 -1]
```

You can see that multiplication by `A` just interchanges the rational cusps determined by the columns of the matrix `MR.reps(3)`:

```
sage: MR.reps(3), A*MR.reps(3)
(
[-1 -1]  [ 1 -1]
[ 3  2], [-2  3]
)
```

is_unimodular_path(*r1*, *r2*)

Determine whether two (non-infinite) cusps are connected by a unimodular path.

INPUT:

- `r1, r2` – rational numbers

OUTPUT:

A boolean expressing whether or not a unimodular path connects `r1` to `r2`.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.is_unimodular_path(0,1/3)
True
sage: A.is_unimodular_path(1/3,0)
True
sage: A.is_unimodular_path(0,2/3)
False
sage: A.is_unimodular_path(2/3,0)
False
```

prep_hecke_on_gen(*l*, *gen*, *modulus=None*)

This function does some precomputations needed to compute T_l.

In particular, if ϕ is a modular symbol and D_m is the divisor associated to the generator `gen`, to compute $(\phi|T_l)(D_m)$ one needs to compute $\phi(\gamma_a D_m)|\gamma_a$ where γ_a runs through the $l+1$ matrices defining T_l. One then takes $\gamma_a D_m$ and writes it as a sum of unimodular divisors. For each such unimodular divisor, say $[M]$ where M is a SL_2 matrix, we then write $M = \gamma h$ where γ is in $\Gamma_0(N)$ and h is one of our chosen coset representatives. Then $\phi([M]) = \phi([h])|\gamma^{-1}$. Thus, one has

$$(\phi|\gamma_a)(D_m) = \sum_h \sum_j \phi([h])|\gamma_{hj}^{-1} \cdot \gamma_a$$

as h runs over all coset representatives and j simply runs over however many times M_h appears in the above computation.

Finally, the output of this function is a dictionary `D` whose keys are the coset representatives in `self.reps()` where each value is a list of matrices, and the entries of `D` satisfy:

$$D[h][j] = \gamma_{hj} * \gamma_a$$

INPUT:

- `l` – a prime
- `gen` – a generator

OUTPUT:

A list of lists (see above).

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: M = phi.parent().source()
sage: w = M.prep_hecke_on_gen(2, M.gens()[0])
sage: one = Matrix(ZZ,2,2,1)
sage: one.set_immutable()
sage: w[one]
[[1 0]
[0 2], [1 1]
[0 2], [2 0]
[0 1]]
```

prep_hecke_on_gen_list(*l*, *gen*, *modulus=None*)

Return the precomputation to compute T_l in a way that speeds up the Hecke calculation.

Namely, returns a list of the form [h,A].

INPUT:

- `l` – a prime
- `gen` – a generator

OUTPUT:

A list of lists (see above).

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: M = phi.parent().source()
sage: len(M.prep_hecke_on_gen_list(2, M.gens()[0]))
4
```

reps_with_three_torsion()

A list of coset representatives whose associated unimodular path contains a point fixed by a $\Gamma_0(N)$ element of order 3 in the ideal triangle directly below that path (the order is computed in $PSL_2(\mathbf{Z})$).

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(13)
sage: B = MR.reps_with_three_torsion()[0]; B
[ 0 -1]
[ 1  3]
```

The corresponding matrix of order three:

```
sage: A = MR.three_torsion_matrix(B); A
[-4 -1]
[13  3]
sage: A^3
[1 0]
[0 1]
```

The columns of B and the columns of A*B and A^2*B give the same rational cusps:

```
sage: B
[ 0 -1]
[ 1  3]
sage: A*B, A^2*B
(
[-1  1]  [ 1  0]
[ 3 -4], [-4  1]
)
```

reps_with_two_torsion()

The coset representatives whose associated unimodular path contains a point fixed by a $\Gamma_0(N)$ element of order 2 (where the order is computed in $PSL_2(\mathbf{Z})$).

OUTPUT:

A list of matrices.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(11)
sage: MR.reps_with_two_torsion()
[]
sage: MR = ManinRelations(13)
```

(continues on next page)

(continued from previous page)

```
sage: MR.reps_with_two_torsion()
[
[-1 -1]  [-1 -2]
[ 3  2], [ 2  3]
]
sage: B = MR.reps_with_two_torsion()[0]
```

The corresponding matrix of order 2:

```
sage: A = MR.two_torsion_matrix(B); A
[  5   2]
[-13  -5]
sage: A^2
[-1  0]
[ 0 -1]
```

You can see that multiplication by `A` just interchanges the rational cusps determined by the columns of the matrix `MR.reps(3)`:

```
sage: B, A*B
(
[-1 -1]  [ 1 -1]
[ 3  2], [-2  3]
)
```

three_torsion_matrix(*A*)

Return the matrix of order two in $\Gamma_0(N)$ which corresponds to an `A` in `self.reps_with_two_torsion()`.

INPUT:

- `A` – a matrix in `self.reps_with_two_torsion()`

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(37)
sage: B = MR.reps_with_three_torsion()[0]
```

The corresponding matrix of order 3:

```
sage: A = MR.three_torsion_matrix(B); A
[-11  -3]
[ 37  10]
sage: A^3
[1 0]
[0 1]
```

two_torsion_matrix(*A*)

Return the matrix of order two in $\Gamma_0(N)$ which corresponds to an `A` in `self.reps_with_two_torsion()`.

INPUT:

- `A` – a matrix in `self.reps_with_two_torsion()`

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(25)
sage: B = MR.reps_with_two_torsion()[0]
```

The corresponding matrix of order 2:

```
sage: A = MR.two_torsion_matrix(B); A
[  7   2]
[-25  -7]
sage: A^2
[-1  0]
[ 0 -1]
```

unimod_to_matrices(*r1*, *r2*)

Return the two matrices whose associated unimodular paths connect `r1` and `r2` and `r2` and `r1`, respectively.

INPUT:

- `r1, r2` – rational numbers (that are assumed to be connected by a unimodular path)

OUTPUT:

A pair of 2×2 matrices of determinant 1

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.unimod_to_matrices(0,1/3)
(
[ 0  1]  [1 0]
[-1  3], [3 1]
)
```

class sage.modular.pollack_stevens.fund_domain.**PollackStevensModularDomain**(*N*, *reps*, *indices*, *rels*, *equiv_ind*)

Bases: `sage.structure.sage_object.SageObject`

The domain of a modular symbol.

INPUT:

- `N` – a positive integer, the level of the congruence subgroup $\Gamma_0(N)$
- `reps` – a list of 2×2 matrices, the coset representatives of $Div^0(P^1(\mathbf{Q}))$
- `indices` – a list of integers; indices of elements in `reps` which are generators
- `rels` – a list of list of triples `(d, A, i)`, one for each coset representative of `reps` which describes how to express the elements of `reps` in terms of generators specified by `indices`. See *relations()* for a detailed explanations of these triples.
- `equiv_ind` – a dictionary which maps normalized coordinates on $P^1(\mathbf{Z}/N\mathbf{Z})$ to an integer such that a matrix whose bottom row is equivalent to $[a : b]$ in $P^1(\mathbf{Z}/N\mathbf{Z})$ is in the coset of `reps[equiv_ind[(a, b)]]`

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import␣
↪PollackStevensModularDomain, M2Z
sage: PollackStevensModularDomain(2 , [M2Z([1,0,0,1]), M2Z([1,1,-1,0]), M2Z([0,-1,1,
↪1])], [0,2], [[(1, M2Z([1,0,0,1]), 0)], [(-1,M2Z([-1,-1,0,-1]),0)], [(1, M2Z([1,0,
↪0,1]), 2)]], {(0,1): 0, (1,0): 1, (1,1): 2})
Modular Symbol domain of level 2
```

P1()

Return the Sage representation of $P^1(\mathbf{Z}/N\mathbf{Z})$.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.P1()
The projective line over the integers modulo 11
```

equivalent_index(A)

Return the index of the coset representative equivalent to `A`.

Here by equivalent we mean the unique coset representative whose bottom row is equivalent to the bottom row of `A` in $P^1(\mathbf{Z}/N\mathbf{Z})$.

INPUT:

- `A` – an element of $SL_2(\mathbf{Z})$

OUTPUT:

The unique integer `j` satisfying that the bottom row of `self.reps(j)` is equivalent to the bottom row of `A`.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(11)
sage: A = matrix(ZZ,2,2,[1,5,3,16])
sage: j = MR.equivalent_index(A); j
11
sage: MR.reps(11)
[ 1 -1]
[-1  2]
sage: MR.equivalent_rep(A)
[ 1 -1]
[-1  2]
sage: MR.P1().normalize(3,16)
(1, 9)
```

equivalent_rep(A)

Return a coset representative that is equivalent to `A` modulo $\Gamma_0(N)$.

INPUT:

- `A` – a matrix in $SL_2(\mathbf{Z})$

OUTPUT:

The unique generator congruent to `A` modulo $\Gamma_0(N)$.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = matrix([[5,3],[38,23]])
sage: ManinRelations(60).equivalent_rep(A)
[-7 -3]
[26 11]
```

gen(*n=0*)

Return the n-th generator.

INPUT:

- n – integer (default: 0), which generator is desired

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(137)
sage: A.gen(17)
[-4 -1]
[ 9  2]
```

gens()

Return the list of coset representatives chosen as generators.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.gens()
[
[1 0]  [ 0 -1]  [-1 -1]
[0 1], [ 1  3], [ 3  2]
]
```

indices(*n=None*)

Return the n-th index of the coset representatives which were chosen as our generators.

In particular, the divisors associated to these coset representatives generate all divisors over $\mathbf{Z}[\Gamma_0(N)]$, and thus a modular symbol is uniquely determined by its values on these divisors.

INPUT:

- n – integer (default: None)

OUTPUT:

The n-th index of the generating set in `self.reps()` or all indices if n is None.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.indices()
[0, 2, 3]

sage: A.indices(2)
3

```

(continues on next page)

(continued from previous page)

```
sage: A = ManinRelations(13)
sage: A.indices()
[0, 2, 3, 4, 5]

sage: A = ManinRelations(101)
sage: A.indices()
[0, 2, 3, 4, 5, 6, 8, 9, 11, 13, 14, 16, 17, 19, 20, 23, 24, 26, 28]
```

level()

Return the level N of $\Gamma_0(N)$ that we work with.

OUTPUT:

The integer N of the group $\Gamma_0(N)$ for which the Manin Relations are being computed.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
sage: A.level()
11
```

ngens()

Return the number of generators.

OUTPUT:

The number of coset representatives from which a modular symbol's value on any coset can be derived.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(1137)
sage: A.ngens()
255
```

relations(*A=None*)

Express the divisor attached to the coset representative of `A` in terms of our chosen generators.

INPUT:

- `A` – `None`, an integer, or a coset representative (default: `None`)

OUTPUT:

A $\mathbf{Z}[\Gamma_0(N)]$-relation expressing the divisor attached to `A` in terms of the generating set. The relation is given as a list of triples `(d, B, i)` such that the divisor attached to A is the sum of `d` times the divisor attached to `B^{-1} * self.reps(i)`.

If `A` is an integer, then return this data for the `A`-th coset representative.

If `A` is `None`, then return this data in a list for all coset representatives.

Note: These relations allow us to recover the value of a modular symbol on any coset representative in terms of its values on our generating set.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: MR = ManinRelations(11)
sage: MR.indices()
[0, 2, 3]
sage: MR.relations(0)
[(1, [1 0]
[0 1], 0)]
sage: MR.relations(2)
[(1, [1 0]
[0 1], 2)]
sage: MR.relations(3)
[(1, [1 0]
[0 1], 3)]
```

The fourth coset representative can be expressed through the second coset representative:

```
sage: MR.reps(4)
[-1 -2]
[ 2  3]
sage: d, B, i = MR.relations(4)[0]
sage: P = B.inverse()*MR.reps(i); P
[ 2 -1]
[-3  2]
sage: d # the above corresponds to minus the divisor of A.reps(4) since d is -1
-1
```

The sixth coset representative can be expressed as the sum of the second and the third:

```
sage: MR.reps(6)
[ 0 -1]
[ 1  2]
sage: MR.relations(6)
[(1, [1 0]
[0 1], 2), (1, [1 0]
[0 1], 3)]
sage: MR.reps(2), MR.reps(3) # MR.reps(6) is the sum of these divisors
(
[ 0 -1]  [-1 -1]
[ 1  3], [ 3  2]
)
```

reps(*n=None*)

Return the n-th coset representative associated with our fundamental domain.

INPUT:

- n – integer (default: None)

OUTPUT:

The n-th coset representative or all coset representatives if n is None.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: A = ManinRelations(11)
```

(continues on next page)

(continued from previous page)

```
sage: A.reps(0)
[1 0]
[0 1]
sage: A.reps(1)
[ 1  1]
[-1  0]
sage: A.reps(2)
[ 0 -1]
[ 1  3]
sage: A.reps()
[
[1 0]  [ 1  1]  [ 0 -1]  [-1 -1]  [-1 -2]  [-2 -1]  [ 0 -1]  [ 1  0]
[0 1], [-1  0], [ 1  3], [ 3  2], [ 2  3], [ 3  1], [ 1  2], [-2  1],

[ 0 -1]  [ 1  0]  [-1 -1]  [ 1 -1]
[ 1  1], [-1  1], [ 2  1], [-1  2]
]
```

sage.modular.pollack_stevens.fund_domain.**basic_hecke_matrix**(*a*, *l*)

Return the 2×2 matrix with entries `[1, a, 0, l]` if a<l and `[l, 0, 0, 1]` if a>=l.

INPUT:

- a – an integer or Infinity
- `l` – a prime

OUTPUT:

A 2×2 matrix of determinant l

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.fund_domain import basic_hecke_matrix
sage: basic_hecke_matrix(0, 7)
[1 0]
[0 7]
sage: basic_hecke_matrix(5, 7)
[1 5]
[0 7]
sage: basic_hecke_matrix(7, 7)
[7 0]
[0 1]
sage: basic_hecke_matrix(19, 7)
[7 0]
[0 1]
sage: basic_hecke_matrix(infinity, 7)
[7 0]
[0 1]
```

19.4 p-adic L-series attached to overconvergent eigensymbols

An overconvergent eigensymbol gives rise to a p-adic L-series, which is essentially defined as the evaluation of the eigensymbol at the path $0 \to \infty$. The resulting distribution on $\mathbf{Z}_p$ can be restricted to $\mathbf{Z}_p^\times$, thus giving the measure attached to the sought p-adic L-series.

All this is carefully explained in [PS2011].

`sage.modular.pollack_stevens.padic_lseries.`**`log_gamma_binomial`**(*p*, *gamma*, *n*, *M*)

Return the list of coefficients in the power series expansion (up to precision M) of $\binom{\log_p(z)/\log_p(\gamma)}{n}$

INPUT:

- `p` – prime
- `gamma` – topological generator, e.g. $1 + p$
- `n` – nonnegative integer
- `M` – precision

OUTPUT:

The list of coefficients in the power series expansion of $\binom{\log_p(z)/\log_p(\gamma)}{n}$

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.padic_lseries import log_gamma_binomial
sage: log_gamma_binomial(5,1+5,2,4)
[0, -3/205, 651/84050, -223/42025]
sage: log_gamma_binomial(5,1+5,3,4)
[0, 2/205, -223/42025, 95228/25845375]
```

class `sage.modular.pollack_stevens.padic_lseries.`**`pAdicLseries`**(*symb*, *gamma=None*, *quadratic_twist=1*, *precision=None*)

Bases: `sage.structure.sage_object.SageObject`

The p-adic L-series associated to an overconvergent eigensymbol.

INPUT:

- `symb` – an overconvergent eigensymbol
- `gamma` – topological generator of $1 + p\mathbf{Z}_p$ (default: $1 + p$ or 5 if $p = 2$)
- `quadratic_twist` – conductor of quadratic twist χ (default: 1)
- `precision` – if `None` (default) is specified, the correct precision bound is computed and the answer is returned modulo that accuracy

EXAMPLES:

```
sage: E = EllipticCurve('37a')
sage: p = 5
sage: prec = 4
sage: L = E.padic_lseries(p, implementation="pollackstevens", precision=prec) #␣
↪long time
sage: L[1]               # long time
1 + 4*5 + 2*5^2 + O(5^3)
sage: L.series(3)     # long time
O(5^4) + (1 + 4*5 + 2*5^2 + O(5^3))*T + (3 + O(5^2))*T^2 + O(T^3)
```

```
sage: from sage.modular.pollack_stevens.padic_lseries import pAdicLseries
sage: E = EllipticCurve('20a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.p_stabilize_and_lift(3, 4) # long time
sage: L = pAdicLseries(Phi)               # long time
sage: L.series(4)                         # long time
2*3 + O(3^4) + (3 + O(3^2))*T + (2 + O(3))*T^2 + O(3^0)*T^3 + O(T^4)
```

An example of a p-adic L-series associated to a modular abelian surface. This is not tested as it takes too long.:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_
↪space
sage: from sage.modular.pollack_stevens.padic_lseries import pAdicLseries
sage: A = ModularSymbols(103,2,1).cuspidal_submodule().new_subspace().
↪decomposition()[0]
sage: p = 19
sage: prec = 4
sage: phi = ps_modsym_from_simple_modsym_space(A)
sage: ap = phi.Tq_eigenvalue(p,prec)
sage: c1,c2 = phi.completions(p,prec)
sage: phi1,psi1 = c1
sage: phi2,psi2 = c2
sage: phi1p = phi1.p_stabilize_and_lift(p,ap = psi1(ap), M = prec) # not tested -␣
↪too long
sage: L1 = pAdicLseries(phi1p)                                      # not tested -␣
↪too long
sage: phi2p = phi2.p_stabilize_and_lift(p,ap = psi2(ap), M = prec) # not tested -␣
↪too long
sage: L2  = pAdicLseries(phi2p)                                     # not tested -␣
↪too long
sage: L1[1]*L2[1]                                                   # not tested -␣
↪too long
13 + 9*19 + 18*19^2 + O(19^3)
```

interpolation_factor(*ap*, *chip=1*, *psi=None*)

Return the interpolation factor associated to self. This is the p-adic multiplier that which appears in the interpolation formula of the p-adic L-function. It has the form $(1-\alpha_p^{-1})^2$, where α_p is the unit root of $X^2 - \psi(a_p)\chi(p)X + p$.

INPUT:

- ap – the eigenvalue of the Up operator
- chip – the value of the nebentype at p (default: 1)
- psi – a twisting character (default: None)

OUTPUT: a p-adic number

EXAMPLES:

```
sage: E = EllipticCurve('19a2')
sage: L = E.padic_lseries(3,implementation="pollackstevens",precision=6)   #␣
↪long time
sage: ap = E.ap(3)              # long time
```

(continues on next page)

(continued from previous page)

```
sage: L.interpolation_factor(ap) # long time
3^2 + 3^3 + 2*3^5 + 2*3^6 + O(3^7)
```

Comparing against a different implementation:

```
sage: L = E.padic_lseries(3)
sage: (1-1/L.alpha(prec=4))^2
3^2 + 3^3 + O(3^5)
```

prime()

Return the prime p as in p-adic L-series.

EXAMPLES:

```
sage: E = EllipticCurve('19a')
sage: L = E.padic_lseries(19, implementation="pollackstevens",precision=6) #␣
↪long time
sage: L.prime()                    # long time
19
```

quadratic_twist()

Return the discriminant of the quadratic twist.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.padic_lseries import pAdicLseries
sage: E = EllipticCurve('37a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.lift(37,4)
sage: L = pAdicLseries(Phi, quadratic_twist=-3)
sage: L.quadratic_twist()
-3
```

series(*prec=5*)

Return the prec-th approximation to the p-adic L-series associated to self, as a power series in T (corresponding to $\gamma - 1$ with γ the chosen generator of $1 + p\mathbf{Z}_p$).

INPUT:

- prec – (default 5) is the precision of the power series

EXAMPLES:

```
sage: E = EllipticCurve('14a2')
sage: p = 3
sage: prec = 6
sage: L = E.padic_lseries(p,implementation="pollackstevens",precision=prec) #␣
↪long time
sage: L.series(4)          # long time
2*3 + 3^4 + 3^5 + O(3^6) + (2*3 + 3^2 + O(3^4))*T + (2*3 + O(3^2))*T^2 + (3 +␣
↪O(3^2))*T^3 + O(T^4)

sage: E = EllipticCurve("15a3")
sage: L = E.padic_lseries(5,implementation="pollackstevens",precision=15)  #␣
↪long time
```

(continues on next page)

(continued from previous page)

```
sage: L.series(3)            # long time
O(5^15) + (2 + 4*5^2 + 3*5^3 + 5^5 + 2*5^6 + 3*5^7 + 3*5^8 + 2*5^9 + 2*5^10 +␣
↪3*5^11 + 5^12 + O(5^13))*T + (4*5 + 4*5^3 + 3*5^4 + 4*5^5 + 3*5^6 + 2*5^7 + 5^
↪8 + 4*5^9 + 3*5^10 + O(5^11))*T^2 + O(T^3)

sage: E = EllipticCurve("79a1")
sage: L = E.padic_lseries(2,implementation="pollackstevens",precision=10) # not␣
↪tested
sage: L.series(4)  # not tested
O(2^9) + (2^3 + O(2^4))*T + O(2^0)*T^2 + (O(2^-3))*T^3 + O(T^4)
```

symbol()

Return the overconvergent modular symbol.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.padic_lseries import pAdicLseries
sage: E = EllipticCurve('21a4')
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.p_stabilize_and_lift(2,5)   # long time
sage: L = pAdicLseries(Phi)                 # long time
sage: L.symbol()                              # long time
Modular symbol of level 42 with values in Space of 2-adic
distributions with k=0 action and precision cap 15
sage: L.symbol() is Phi                       # long time
True
```

19.5 Manin map

Represents maps from a set of right coset representatives to a coefficient module.

This is a basic building block for implementing modular symbols, and provides basic arithmetic and right action of matrices.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi
Modular symbol of level 11 with values in Sym^0 Q^2
sage: phi.values()
[-1/5, 1, 0]

sage: from sage.modular.pollack_stevens.manin_map import ManinMap, M2Z
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: D = OverconvergentDistributions(0, 11, 10)
sage: MR = ManinRelations(11)
sage: data  = {M2Z([1,0,0,1]):D([1,2]), M2Z([0,-1,1,3]):D([3,5]), M2Z([-1,-1,3,2]):D([1,
↪1])}
sage: f = ManinMap(D, MR, data)
sage: f(M2Z([1,0,0,1]))
(1 + O(11^2), 2 + O(11))
```

(continues on next page)

(continued from previous page)

```
sage: S = Symk(0,QQ)
sage: MR = ManinRelations(37)
sage: data  = {M2Z([-2,-3,5,7]): S(0), M2Z([1,0,0,1]): S(0), M2Z([-1,-2,3,5]): S(0),␣
↪M2Z([-1,-4,2,7]): S(1), M2Z([0,-1,1,4]): S(1), M2Z([-3,-1,7,2]): S(-1), M2Z([-2,-3,3,
↪4]): S(0), M2Z([-4,-3,7,5]): S(0), M2Z([-1,-1,4,3]): S(0)}
sage: f = ManinMap(S,MR,data)
sage: f(M2Z([2,3,4,5]))
1
```

class sage.modular.pollack_stevens.manin_map.**ManinMap**(*codomain*, *manin_relations*, *defining_data*, *check=True*)

Bases: `object`

Map from a set of right coset representatives of $\Gamma_0(N)$ in $SL_2(\mathbf{Z})$ to a coefficient module that satisfies the Manin relations.

INPUT:

- `codomain` – coefficient module
- `manin_relations` – a *sage.modular.pollack_stevens.fund_domain.ManinRelations* object
- `defining_data` – a dictionary whose keys are a superset of `manin_relations.gens()` and a subset of `manin_relations.reps()`, and whose values are in the codomain.
- `check` – do numerous (slow) checks and transformations to ensure that the input data is perfect.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: D = OverconvergentDistributions(0, 11, 10)
sage: manin = sage.modular.pollack_stevens.fund_domain.ManinRelations(11)
sage: data  = {M2Z([1,0,0,1]):D([1,2]), M2Z([0,-1,1,3]):D([3,5]), M2Z([-1,-1,3,
↪2]):D([1,1])}
sage: f = ManinMap(D, manin, data); f # indirect doctest
Map from the set of right cosets of Gamma0(11) in SL_2(Z) to Space of 11-adic␣
↪distributions with k=0 action and precision cap 10
sage: f(M2Z([1,0,0,1]))
(1 + O(11^2), 2 + O(11))
```

apply(*f*, *codomain=None*, *to_moments=False*)

Return Manin map given by $x \mapsto f(self(x))$, where f is anything that can be called with elements of the coefficient module.

This might be used to normalize, reduce modulo a prime, change base ring, etc.

INPUT:

- `f` – anything that can be called with elements of the coefficient module
- `codomain` – (default: None) the codomain of the return map
- `to_moments` – (default: False) if True, will apply `f` to each of the moments instead

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
```

(continues on next page)

(continued from previous page)

```
sage: S = Symk(0,QQ)
sage: MR = ManinRelations(37)
sage: data  = {M2Z([-2,-3,5,7]): S(0), M2Z([1,0,0,1]): S(0), M2Z([-1,-2,3,5]):␣
↪S(0), M2Z([-1,-4,2,7]): S(1), M2Z([0,-1,1,4]): S(1), M2Z([-3,-1,7,2]): S(-1),␣
↪M2Z([-2,-3,3,4]): S(0), M2Z([-4,-3,7,5]): S(0), M2Z([-1,-1,4,3]): S(0)}
sage: f = ManinMap(S,MR,data)
sage: list(f.apply(lambda t:2*t))
[0, 2, 0, 0, 0, -2, 2, 0, 0]
```

compute_full_data()

Compute the values of self on all coset reps from its values on our generating set.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: S = Symk(0,QQ)
sage: MR = ManinRelations(37); MR.gens()
[
[1 0]  [ 0 -1]  [-1 -1]  [-1 -2]  [-2 -3]  [-3 -1]  [-1 -4]  [-4 -3]
[0 1], [ 1  4], [ 4  3], [ 3  5], [ 5  7], [ 7  2], [ 2  7], [ 7  5],

[-2 -3]
[ 3  4]
]

sage: data  = {M2Z([-2,-3,5,7]): S(0), M2Z([1,0,0,1]): S(0), M2Z([-1,-2,3,5]):␣
↪S(0), M2Z([-1,-4,2,7]): S(1), M2Z([0,-1,1,4]): S(1), M2Z([-3,-1,7,2]): S(-1),␣
↪M2Z([-2,-3,3,4]): S(0), M2Z([-4,-3,7,5]): S(0), M2Z([-1,-1,4,3]): S(0)}
sage: f = ManinMap(S,MR,data)
sage: len(f._dict)
9
sage: f.compute_full_data()
sage: len(f._dict)
38
```

extend_codomain(*new_codomain*, *check=True*)

Extend the codomain of self to new_codomain. There must be a valid conversion operation from the old to the new codomain. This is most often used for extension of scalars from $\mathbf{Q}$ to $\mathbf{Q}_p$.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import ManinMap, M2Z
sage: from sage.modular.pollack_stevens.fund_domain import ManinRelations
sage: S = Symk(0,QQ)
sage: MR = ManinRelations(37)
sage: data  = {M2Z([-2,-3,5,7]): S(0), M2Z([1,0,0,1]): S(0), M2Z([-1,-2,3,5]):␣
↪S(0), M2Z([-1,-4,2,7]): S(1), M2Z([0,-1,1,4]): S(1), M2Z([-3,-1,7,2]): S(-1),␣
↪M2Z([-2,-3,3,4]): S(0), M2Z([-4,-3,7,5]): S(0), M2Z([-1,-1,4,3]): S(0)}
sage: m = ManinMap(S, MR, data); m
Map from the set of right cosets of Gamma0(37) in SL_2(Z) to Sym^0 Q^2
sage: m.extend_codomain(Symk(0, Qp(11)))
Map from the set of right cosets of Gamma0(37) in SL_2(Z) to Sym^0 Q_11^2
```

hecke(*ell*, *algorithm='prep'*)

Return the image of this Manin map under the Hecke operator T_ℓ.

INPUT:

- `ell` – a prime
- `algorithm` – a string, either 'prep' (default) or 'naive'

OUTPUT:

- The image of this ManinMap under the Hecke operator T_ℓ

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi.is_Tq_eigensymbol(7,7,10)
True
sage: phi.hecke(7).values()
[2/5, -2, 0]
sage: phi.Tq_eigenvalue(7,7,10)
-2
```

normalize()

Normalize every value of self – e.g., reduces each value's j-th moment modulo p^{N-j}

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: D = OverconvergentDistributions(0, 11, 10)
sage: manin = sage.modular.pollack_stevens.fund_domain.ManinRelations(11)
sage: data  = {M2Z([1,0,0,1]):D([1,2]), M2Z([0,-1,1,3]):D([3,5]), M2Z([-1,-1,3,
↪2]):D([1,1])}
sage: f = ManinMap(D, manin, data)
sage: f._dict[M2Z([1,0,0,1])]
(1 + O(11^2), 2 + O(11))
sage: g = f.normalize()
sage: g._dict[M2Z([1,0,0,1])]
(1 + O(11^2), 2 + O(11))
```

p_stabilize(*p*, *alpha*, *V*)

Return the p-stabilization of self to level $N * p$ on which U_p acts by α.

INPUT:

- `p` – a prime.
- `alpha` – a U_p-eigenvalue.
- `V` – a space of modular symbols.

OUTPUT:

- The image of this ManinMap under the Hecke operator T_ℓ

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: f = phi._map
sage: V = phi.parent()
sage: f.p_stabilize(5,1,V)
Map from the set of right cosets of Gamma0(11) in SL_2(Z) to Sym^0 Q^2
```

reduce_precision(*M*)

Reduce the precision of all the values of the Manin map.

INPUT:

- M – an integer, the new precision.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: D = OverconvergentDistributions(0, 11, 10)
sage: manin = sage.modular.pollack_stevens.fund_domain.ManinRelations(11)
sage: data  = {M2Z([1,0,0,1]):D([1,2]), M2Z([0,-1,1,3]):D([3,5]), M2Z([-1,-1,3,
↪2]):D([1,1])}
sage: f = ManinMap(D, manin, data)
sage: f._dict[M2Z([1,0,0,1])]
(1 + O(11^2), 2 + O(11))
sage: g = f.reduce_precision(1)
sage: g._dict[M2Z([1,0,0,1])]
1 + O(11^2)
```

specialize(**args*)

Specialize all the values of the Manin map to a new coefficient module. Assumes that the codomain has a specialize method, and passes all its arguments to that method.

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.manin_map import M2Z, ManinMap
sage: D = OverconvergentDistributions(0, 11, 10)
sage: manin = sage.modular.pollack_stevens.fund_domain.ManinRelations(11)
sage: data  = {M2Z([1,0,0,1]):D([1,2]), M2Z([0,-1,1,3]):D([3,5]), M2Z([-1,-1,3,
↪2]):D([1,1])}
sage: f = ManinMap(D, manin, data)
sage: g = f.specialize()
sage: g._codomain
Sym^0 Z_11^2
```

sage.modular.pollack_stevens.manin_map.**unimod_matrices_from_infty**(*r*, *s*)

Return a list of matrices whose associated unimodular paths connect ∞ to r/s.

INPUT:

- r, s – rational numbers

OUTPUT:

- a list of $SL_2(\mathbf{Z})$ matrices

EXAMPLES:

```
sage: v = sage.modular.pollack_stevens.manin_map.unimod_matrices_from_infty(19,23);
↪v
[
[ 0  1]  [-1  0]  [-4  1]  [-5 -4]  [-19   5]
[-1  0], [-1 -1], [-5  1], [-6 -5], [-23   6]
]
sage: [a.det() for a in v]
[1, 1, 1, 1, 1]

sage: sage.modular.pollack_stevens.manin_map.unimod_matrices_from_infty(11,25)
[
[ 0  1]  [-1  0]  [-3  1]  [-4 -3]  [-11   4]
[-1  0], [-2 -1], [-7  2], [-9 -7], [-25   9]
]
```

ALGORITHM:

This is Manin's continued fraction trick, which gives an expression $\{\infty, r/s\} = \{\infty, 0\} + ... + \{a, b\} + ... + \{*, r/s\}$, where each $\{a, b\}$ is the image of $\{0, \infty\}$ under a matrix in $SL_2(\mathbf{Z})$.

sage.modular.pollack_stevens.manin_map.**unimod_matrices_to_infty**(*r*, *s*)

Return a list of matrices whose associated unimodular paths connect 0 to `r/s`.

INPUT:

- `r`, `s` – rational numbers

OUTPUT:

- a list of matrices in $SL_2(\mathbf{Z})$

EXAMPLES:

```
sage: v = sage.modular.pollack_stevens.manin_map.unimod_matrices_to_infty(19,23); v
[
[1 0]  [ 0  1]  [1 4]  [-4  5]  [ 5 19]
[0 1], [-1  1], [1 5], [-5  6], [ 6 23]
]
sage: [a.det() for a in v]
[1, 1, 1, 1, 1]

sage: sage.modular.pollack_stevens.manin_map.unimod_matrices_to_infty(11,25)
[
[1 0]  [ 0  1]  [1 3]  [-3  4]  [ 4 11]
[0 1], [-1  2], [2 7], [-7  9], [ 9 25]
]
```

ALGORITHM:

This is Manin's continued fraction trick, which gives an expression $\{0, r/s\} = \{0, \infty\} + ... + \{a, b\} + ... + \{*, r/s\}$, where each $\{a, b\}$ is the image of $\{0, \infty\}$ under a matrix in $SL_2(\mathbf{Z})$.

19.6 Element class for Pollack-Stevens' Modular Symbols

This is the class of elements in the spaces of Pollack-Steven's modular symbols as described in [PS2011].

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol(); phi
Modular symbol of level 11 with values in Sym^0 Q^2
sage: phi.weight() # Note that weight k=2 of a modular form corresponds here to weight 0
0
sage: phi.values()
[-1/5, 1, 0]
sage: phi.is_ordinary(11)
True
sage: phi_lift = phi.lift(11, 5, eigensymbol = True) # long time
sage: phi_lift.padic_lseries().series(5) # long time
O(11^5) + (10 + 3*11 + 6*11^2 + 9*11^3 + O(11^4))*T + (6 + 3*11 + 2*11^2 + O(11^3))*T^2␣
↪+ (2 + 2*11 + O(11^2))*T^3 + (5 + O(11))*T^4 + O(T^5)
```

```
sage: A = ModularSymbols(Gamma1(8),4).decomposition()[0].plus_submodule().new_subspace()
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_modsym_space
sage: phi = ps_modsym_from_simple_modsym_space(A)
sage: phi.values()
[(-1, 0, 0), (1, 0, 0), (-9, -6, -4)]
```

class sage.modular.pollack_stevens.modsym.**PSModSymAction**(*actor*, *MSspace*)

Bases: sage.categories.action.Action

Create the action

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: g = phi._map._codomain._act._Sigma0(matrix(ZZ,2,2,[1,2,3,4]))
sage: phi * g # indirect doctest
Modular symbol of level 11 with values in Sym^0 Q^2
```

class sage.modular.pollack_stevens.modsym.**PSModularSymbolElement**(*map_data*, *parent*, *construct=False*)

Bases: sage.structure.element.ModuleElement

Initialize a modular symbol

EXAMPLES:

```
sage: E = EllipticCurve('37a')
sage: phi = E.pollack_stevens_modular_symbol()
```

Tq_eigenvalue(*q*, *p=None*, *M=None*, *check=True*)

Eigenvalue of T_q modulo p^M

INPUT:

- q – prime of the Hecke operator
- p – prime we are working modulo (default: None)

- M – degree of accuracy of approximation (default: None)
- check – check that self is an eigensymbol

OUTPUT:

- Constant c such that $self|T_q - c * self$ has valuation greater than or equal to M (if it exists), otherwise raises ValueError

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi_ord = phi.p_stabilize(p = 3, ap = E.ap(3), M = 10, ordinary = True)
sage: phi_ord.Tq_eigenvalue(2,3,10) + 2
O(3^10)

sage: phi_ord.Tq_eigenvalue(3,3,10)
2 + 3^2 + 2*3^3 + 2*3^4 + 2*3^6 + 3^8 + 2*3^9 + O(3^10)
sage: phi_ord.Tq_eigenvalue(3,3,100)
Traceback (most recent call last):
...
ValueError: result not determined to high enough precision
```

diagonal_valuation(p)

Return the minimum of the diagonal valuation on the values of self

INPUT:

- p – a positive integral prime

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi.diagonal_valuation(2)
0
sage: phi.diagonal_valuation(3)
0
sage: phi.diagonal_valuation(5)
-1
sage: phi.diagonal_valuation(7)
0
```

dict()

Return dictionary on the modular symbol self, where keys are generators and values are the corresponding values of self on generators

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: Set([x.moment(0) for x in phi.dict().values()]) == Set([-1/5, 1, 0])
True
```

evaluate_twisted(*a*, *chi*)

Return $\Phi_\chi(\{a/p\} - \{\infty\})$ where Φ is `self` and χ is a quadratic character

INPUT:

- a – integer in the range range(p)
- `chi` – the modulus of a quadratic character.

OUTPUT:

The distribution $\Phi_\chi(\{a/p\} - \{\infty\})$.

EXAMPLES:

```
sage: E = EllipticCurve('17a1')
sage: L = E.padic_lseries(5, implementation="pollackstevens", precision=4)
↪#long time
sage: D = L.quadratic_twist()           # long time
sage: L.symbol().evaluate_twisted(1,D) # long time
(1 + 5 + 3*5^2 + 5^3 + O(5^4), 5^2 + O(5^3), 1 + O(5^2), 2 + O(5))

sage: E = EllipticCurve('40a4')
sage: L = E.padic_lseries(7, implementation="pollackstevens", precision=4)
↪#long time
sage: D = L.quadratic_twist()           # long time
sage: L.symbol().evaluate_twisted(1,D) # long time
(4 + 6*7 + 3*7^2 + O(7^4), 6*7 + 6*7^2 + O(7^3), 6 + O(7^2), 1 + O(7))
```

hecke(*ell*, *algorithm='prep'*)

Return self | T_ℓ by making use of the precomputations in self.prep_hecke()

INPUT:

- `ell` – a prime
- `algorithm` – a string, either 'prep' (default) or 'naive'

OUTPUT:

- The image of this element under the Hecke operator T_ℓ

ALGORITHMS:

- If `algorithm == 'prep'`, precomputes a list of matrices that only depend on the level, then uses them to speed up the action.
- If `algorithm == 'naive'`, just acts by the matrices defining the Hecke operator. That is, it computes sum_a self | [1,a,0,ell] + self | [ell,0,0,1], the last term occurring only if the level is prime to ell.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi.hecke(2) == phi * E.ap(2)
True
sage: phi.hecke(3) == phi * E.ap(3)
True
sage: phi.hecke(5) == phi * E.ap(5)
```

(continues on next page)

(continued from previous page)

```
True
sage: phi.hecke(101) == phi * E.ap(101)
True

sage: all(phi.hecke(p, algorithm='naive') == phi * E.ap(p) for p in [2,3,5,
↪101]) # long time
True
```

is_Tq_eigensymbol(*q*, *p=None*, *M=None*)

Determine if self is an eigenvector for T_q modulo p^M

INPUT:

- q – prime of the Hecke operator
- p – prime we are working modulo
- M – degree of accuracy of approximation

OUTPUT:

- True/False

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi_ord = phi.p_stabilize(p = 3, ap = E.ap(3), M = 10, ordinary = True)
sage: phi_ord.is_Tq_eigensymbol(2,3,10)
True
sage: phi_ord.is_Tq_eigensymbol(2,3,100)
False
sage: phi_ord.is_Tq_eigensymbol(2,3,1000)
False
sage: phi_ord.is_Tq_eigensymbol(3,3,10)
True
sage: phi_ord.is_Tq_eigensymbol(3,3,100)
False
```

is_ordinary(*p=None*, *P=None*)

Return true if the p-th eigenvalue is a p-adic unit.

INPUT:

- p - a positive integral prime, or None (default None)
- P - a prime of the base ring above p, or None. This is ignored unless the base ring is a number field.

OUTPUT:

- True/False

EXAMPLES:

```
sage: E = EllipticCurve('11a1')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.is_ordinary(2)
```

(continues on next page)

(continued from previous page)

```
False
sage: E.ap(2)
-2
sage: phi.is_ordinary(3)
True
sage: E.ap(3)
-1
sage: phip = phi.p_stabilize(3,20)
sage: phip.is_ordinary()
True
```

A number field example. Here there are multiple primes above p, and ϕ is ordinary at one but not the other.:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_
↪modsym_space
sage: f = Newforms(32, 8, names='a')[1]
sage: phi = ps_modsym_from_simple_modsym_space(f.modular_symbols(1))
sage: (p1, _), (p2, _) = phi.base_ring().ideal(3).factor()
sage: phi.is_ordinary(p1) != phi.is_ordinary(p2)
True
sage: phi.is_ordinary(3)
Traceback (most recent call last):
...
TypeError: P must be an ideal
```

minus_part()

Return the minus part of self – i.e. self - self | [1,0,0,-1]

Note that we haven't divided by 2. Is this a problem?

OUTPUT:

- self – self | [1,0,0,-1]

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: (phi.plus_part()+phi.minus_part()) == phi * 2
True
```

plus_part()

Return the plus part of self – i.e. `self + self | [1,0,0,-1]`.

Note that we haven't divided by 2. Is this a problem?

OUTPUT:

- self + self | [1,0,0,-1]

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
```

(continues on next page)

(continued from previous page)

```
sage: phi.values()
[-1/5, 1, 0]
sage: (phi.plus_part()+phi.minus_part()) == 2 * phi
True
```

valuation(*p=None*)

Return the valuation of `self` at p.

Here the valuation is the minimum of the valuations of the values of `self`.

INPUT:

- `p` - prime

OUTPUT:

- The valuation of `self` at p

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.values()
[-1/5, 1, 0]
sage: phi.valuation(2)
0
sage: phi.valuation(3)
0
sage: phi.valuation(5)
-1
sage: phi.valuation(7)
0
sage: phi.valuation()
Traceback (most recent call last):
...
ValueError: you must specify a prime

sage: phi2 = phi.lift(11, M=2)
sage: phi2.valuation()
0
sage: phi2.valuation(3)
Traceback (most recent call last):
...
ValueError: inconsistent prime
sage: phi2.valuation(11)
0
```

values()

Return the values of the symbol `self` on our chosen generators.

The generators are listed in `self.dict()`.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
```

(continues on next page)

(continued from previous page)

```
sage: phi.values()
[-1/5, 1, 0]
sage: sorted(phi.dict())
[
[-1 -1]  [ 0 -1]  [1 0]
[ 3  2], [ 1  3], [0 1]
]
sage: sorted(phi.values()) == sorted(phi.dict().values())
True
```

weight()

Return the weight of this Pollack-Stevens modular symbol.

This is $k - 2$, where k is the usual notion of weight for modular forms!

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: phi.weight()
0
```

class sage.modular.pollack_stevens.modsym.**PSModularSymbolElement_dist**(*map_data*, *parent*, *construct=False*)

Bases: *sage.modular.pollack_stevens.modsym.PSModularSymbolElement*

padic_lseries(**args*, ***kwds*)

Return the p-adic L-series of this modular symbol.

EXAMPLES:

```
sage: E = EllipticCurve('37a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: L = phi.lift(37, M=6, eigensymbol=True).padic_lseries(); L  # long time
37-adic L-series of Modular symbol of level 37 with values in Space of 37-adic␣
↪distributions with k=0 action and precision cap 7
sage: L.series(2) # long time
O(37^6) + (4 + 37 + 36*37^2 + 19*37^3 + 21*37^4 + O(37^5))*T + O(T^2)
```

precision_relative()

Return the number of moments of each value of self

EXAMPLES:

```
sage: D = OverconvergentDistributions(0, 5, 10)
sage: M = PollackStevensModularSymbols(Gamma0(5), coefficients=D)
sage: f = M(1)
sage: f.precision_relative()
1
```

reduce_precision(M)

Only hold on to M moments of each value of self

EXAMPLES:

```
sage: D = OverconvergentDistributions(0, 5, 10)
sage: M = PollackStevensModularSymbols(Gamma0(5), coefficients=D)
sage: f = M(1)
sage: f.reduce_precision(1)
Modular symbol of level 5 with values in Space of 5-adic distributions with k=0␣
↪action and precision cap 10
```

specialize(*new_base_ring=None*)

Return the underlying classical symbol of weight k - i.e., applies the canonical map $D_k \to Sym^k$ to all values of self.

EXAMPLES:

```
sage: D = OverconvergentDistributions(0, 5, 10);  M =␣
↪PollackStevensModularSymbols(Gamma0(5), coefficients=D); M
Space of overconvergent modular symbols for Congruence Subgroup Gamma0(5) with␣
↪sign 0
and values in Space of 5-adic distributions with k=0 action and precision cap 10
sage: f = M(1)
sage: f.specialize()
Modular symbol of level 5 with values in Sym^0 Z_5^2
sage: f.specialize().values()
[1 + O(5), 1 + O(5), 1 + O(5)]
sage: f.values()
[1 + O(5), 1 + O(5), 1 + O(5)]
sage: f.specialize().parent()
Space of modular symbols for Congruence Subgroup Gamma0(5) with sign 0 and␣
↪values in Sym^0 Z_5^2
sage: f.specialize().parent().coefficient_module()
Sym^0 Z_5^2
sage: f.specialize().parent().coefficient_module().is_symk()
True
sage: f.specialize(Qp(5,20))
Modular symbol of level 5 with values in Sym^0 Q_5^2
```

class sage.modular.pollack_stevens.modsym.**PSModularSymbolElement_symk**(*map_data*, *parent*, *construct=False*)

Bases: *sage.modular.pollack_stevens.modsym.PSModularSymbolElement*

completions(*p*, *M*)

If K is the base_ring of self, this function takes all maps $K \to \mathbf{Q}_p$ and applies them to self return a list of (modular symbol,map: $K \to \mathbf{Q}_p$) as map varies over all such maps.

Note: This only returns all completions when p splits completely in K

INPUT:

- p – prime
- M – precision

OUTPUT:

- A list of tuples (modular symbol,map: $K \to \mathbf{Q}_p$) as map varies over all such maps

EXAMPLES:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_
↪modsym_space
sage: D = ModularSymbols(67,2,1).cuspidal_submodule().new_subspace().
↪decomposition()[1]
sage: f = ps_modsym_from_simple_modsym_space(D)
sage: S = f.completions(41,10); S
[(Modular symbol of level 67 with values in Sym^0 Q_41^2, Ring morphism:
  From: Number Field in alpha with defining polynomial x^2 + 3*x + 1
  To:   41-adic Field with capped relative precision 10
  Defn: alpha |--> 5 + 22*41 + 19*41^2 + 10*41^3 + 28*41^4 + 22*41^5 + 9*41^6 +␣
↪25*41^7 + 40*41^8 + 8*41^9 + O(41^10)), (Modular symbol of level 67 with␣
↪values in Sym^0 Q_41^2, Ring morphism:
  From: Number Field in alpha with defining polynomial x^2 + 3*x + 1
  To:   41-adic Field with capped relative precision 10
  Defn: alpha |--> 33 + 18*41 + 21*41^2 + 30*41^3 + 12*41^4 + 18*41^5 + 31*41^6␣
↪+ 15*41^7 + 32*41^9 + O(41^10))]
sage: TestSuite(S[0][0]).run(skip=['_test_category'])
```

lift(*p=None*, *M=None*, *alpha=None*, *new_base_ring=None*, *algorithm=None*, *eigensymbol=False*, *check=True*)

Return a (p-adic) overconvergent modular symbol with M moments which lifts self up to an Eisenstein error

Here the Eisenstein error is a symbol whose system of Hecke eigenvalues equals $\ell + 1$ for T_ℓ when ℓ does not divide Np and 1 for U_q when q divides Np.

INPUT:

- `p` – prime
- `M` – integer equal to the number of moments
- `alpha` – U_p eigenvalue
- `new_base_ring` – change of base ring
- `algorithm` – 'stevens' or 'greenberg' (default 'stevens')
- `eigensymbol` – if True, lifts to Hecke eigensymbol (self must be a p-ordinary eigensymbol)

(Note: `eigensymbol = True` does *not* just indicate to the code that self is an eigensymbol; it solves a wholly different problem, lifting an eigensymbol to an eigensymbol.)

OUTPUT:

An overconvergent modular symbol whose specialization equals self, up to some Eisenstein error if `eigensymbol` is False. If `eigensymbol = True` then the output will be an overconvergent Hecke eigensymbol (and it will lift the input exactly, the Eisenstein error disappears).

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: f = E.pollack_stevens_modular_symbol()
sage: g = f.lift(11,4,algorithm='stevens',eigensymbol=True)
sage: g.is_Tq_eigensymbol(2)
True
sage: g.Tq_eigenvalue(3)
10 + 10*11 + 10*11^2 + 10*11^3 + O(11^4)
```

(continues on next page)

(continued from previous page)

```
sage: g.Tq_eigenvalue(11)
1 + O(11^4)
```

We check that lifting and then specializing gives back the original symbol:

```
sage: g.specialize() == f
True
```

Another example, which showed precision loss in an earlier version of the code:

```
sage: E = EllipticCurve('37a')
sage: p = 5
sage: prec = 4
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.p_stabilize_and_lift(p,prec, algorithm='stevens',␣
↪eigensymbol=True) # long time
sage: Phi.Tq_eigenvalue(5,M = 4) # long time
3 + 2*5 + 4*5^2 + 2*5^3 + O(5^4)
```

Another example:

```
sage: from sage.modular.pollack_stevens.padic_lseries import pAdicLseries
sage: E = EllipticCurve('37a')
sage: p = 5
sage: prec = 6
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.p_stabilize_and_lift(p=p,M=prec,alpha=None,algorithm='stevens',
↪eigensymbol=True) #long time
sage: L = pAdicLseries(Phi)          # long time
sage: L.symbol() is Phi              # long time
True
```

Examples using Greenberg's algorithm:

```
sage: E = EllipticCurve('11a')
sage: phi = E.pollack_stevens_modular_symbol()
sage: Phi = phi.lift(11,8,algorithm='greenberg',eigensymbol=True)
sage: Phi2 = phi.lift(11,8,algorithm='stevens',eigensymbol=True)
sage: Phi == Phi2
True
```

An example in higher weight:

```
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_
↪modsym_space
sage: f = ps_modsym_from_simple_modsym_space(Newforms(7, 4)[0].modular_
↪symbols(1))
sage: fs = f.p_stabilize(5)
sage: FsG = fs.lift(M=6, eigensymbol=True,algorithm='greenberg') # long time
sage: FsG.values()[0]                                            # long time
5^-1 * (2*5 + 5^2 + 3*5^3 + 4*5^4 + O(5^7), O(5^6), 2*5^2 + 3*5^3 + O(5^5), O(5^
↪4), 5^2 + O(5^3), O(5^2))
sage: FsS = fs.lift(M=6, eigensymbol=True,algorithm='stevens')   # long time
```

(continues on next page)

(continued from previous page)

```
sage: FsS == FsG                                                # long time
True
```

p_stabilize(*p=None*, *M=20*, *alpha=None*, *ap=None*, *new_base_ring=None*, *ordinary=True*, *check=True*)

Return the p-stabilization of self to level Np on which U_p acts by α.

Note that since α is p-adic, the resulting symbol is just an approximation to the true p-stabilization (depending on how well α is approximated).

INPUT:

- `p` – prime not dividing the level of self
- `M` – (default: 20) precision of $\mathbf{Q}_p$
- `alpha` – U_p eigenvalue
- `ap` – Hecke eigenvalue
- `new_base_ring` – change of base ring
- **`ordinary` – (default: True) whether to return the ordinary** (at p) eigensymbol.
- `check` – (default: True) whether to perform extra sanity checks

OUTPUT:

A modular symbol with the same Hecke eigenvalues as self away from p and eigenvalue α at p. The eigenvalue α depends on the parameter `ordinary`.

If `ordinary` == True: the unique modular symbol of level Np with the same Hecke eigenvalues as self away from p and unit eigenvalue at p; else the unique modular symbol of level Np with the same Hecke eigenvalues as self away from p and non-unit eigenvalue at p.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: p = 5
sage: prec = 4
sage: phi = E.pollack_stevens_modular_symbol()
sage: phis = phi.p_stabilize(p,M = prec)
sage: phis
Modular symbol of level 55 with values in Sym^0 Q_5^2
sage: phis.hecke(7) == phis*E.ap(7)
True
sage: phis.hecke(5) == phis*E.ap(5)
False
sage: phis.hecke(3) == phis*E.ap(3)
True
sage: phis.Tq_eigenvalue(5)
1 + 4*5 + 3*5^2 + 2*5^3 + O(5^4)
sage: phis.Tq_eigenvalue(5,M = 3)
1 + 4*5 + 3*5^2 + O(5^3)

sage: phis = phi.p_stabilize(p,M = prec,ordinary=False)
sage: phis.Tq_eigenvalue(5)
5 + 5^2 + 2*5^3 + O(5^5)
```

A complicated example (with nontrivial character):

```
sage: chi = DirichletGroup(24)([-1, -1, -1])
sage: f = Newforms(chi,names='a')[0]
sage: from sage.modular.pollack_stevens.space import ps_modsym_from_simple_
↪modsym_space
sage: phi = ps_modsym_from_simple_modsym_space(f.modular_symbols(1))
sage: phi11, h11 = phi.completions(11,20)[0]
sage: phi11s = phi11.p_stabilize()
sage: phi11s.is_Tq_eigensymbol(11) # long time
True
```

p_stabilize_and_lift(*p*, *M*, *alpha=None*, *ap=None*, *new_base_ring=None*, *ordinary=True*, *algorithm='greenberg'*, *eigensymbol=False*, *check=True*)

p-stabilize and lift self

INPUT:

- p – prime, not dividing the level of self
- M – precision
- alpha – (default: None) the U_p eigenvalue, if known
- ap – (default: None) the Hecke eigenvalue at p (before stabilizing), if known
- new_base_ring – (default: None) if specified, force the resulting eigensymbol to take values in the given ring
- **ordinary – (default: True) whether to return the ordinary** (at p) eigensymbol.
- algorithm – (default: 'greenberg') a string, either 'greenberg' or 'stevens', specifying whether to use the lifting algorithm of M.Greenberg or that of Pollack–Stevens. The latter one solves the difference equation, which is not needed. The option to use Pollack–Stevens' algorithm here is just for historical reasons.
- eigensymbol – (default: False) if True, return an overconvergent eigensymbol. Otherwise just perform a naive lift
- check – (default: True) whether to perform extra sanity checks

OUTPUT:

p-stabilized and lifted version of self.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: f = E.pollack_stevens_modular_symbol()
sage: g = f.p_stabilize_and_lift(3,10)  # long time
sage: g.Tq_eigenvalue(5)                 # long time
1 + O(3^10)
sage: g.Tq_eigenvalue(7)                 # long time
1 + 2*3 + 2*3^2 + 2*3^3 + 2*3^4 + 2*3^5 + 2*3^6 + 2*3^7 + 2*3^8 + 2*3^9 + O(3^
↪10)
sage: g.Tq_eigenvalue(3)                 # long time
2 + 3^2 + 2*3^3 + 2*3^4 + 2*3^6 + 3^8 + 2*3^9 + O(3^10)
```

19.7 Quotients of the Bruhat-Tits tree

This package contains all the functionality described and developed in [FM2014]. It allows for computations with fundamental domains of the Bruhat-Tits tree, under the action of arithmetic groups arising from units in definite quaternion algebras.

EXAMPLES:

Create the quotient attached to a maximal order of the quaternion algebra of discriminant 13, at the prime $p = 5$:

```
sage: Y = BruhatTitsQuotient(5, 13)
```

We can query for its genus, as well as get it back as a graph:

```
sage: Y.genus()
5
sage: Y.get_graph()
Multi-graph on 2 vertices
```

The rest of functionality can be found in the docstrings below.

AUTHORS:

- Cameron Franc and Marc Masdeu (2011): initial version

class sage.modular.btquotients.btquotient.**BruhatTitsQuotient**(*p*, *Nminus*, *Nplus=1*, *character=None*, *use_magma=False*, *seed=None*, *magma_session=None*)

Bases: `sage.structure.sage_object.SageObject`, `sage.structure.unique_representation.UniqueRepresentation`

This function computes the quotient of the Bruhat-Tits tree by an arithmetic quaternionic group. The group in question is the group of norm 1 elements in an Eichler $\mathbf{Z}[1/p]$-order of some (tame) level inside of a definite quaternion algebra that is unramified at the prime p. Note that this routine relies in Magma in the case $p = 2$ or when $N^+ > 1$.

INPUT:

- `p` - a prime number
- `Nminus` - squarefree integer divisible by an odd number of distinct primes and relatively prime to p. This is the discriminant of the definite quaternion algebra that one is quotienting by.
- `Nplus` - an integer coprime to pNminus (Default: 1). This is the tame level. It need not be squarefree! If Nplus is not 1 then the user currently needs magma installed due to sage's inability to compute well with nonmaximal Eichler orders in rational (definite) quaternion algebras.
- `character` - a Dirichlet character (Default: None) of modulus pN^-N^+.
- `use_magma` - boolean (default: False). If True, uses Magma for quaternion arithmetic.
- `magma_session` – (default: None). If specified, the Magma session to use.

EXAMPLES:

Here is an example without a Dirichlet character:

```
sage: X = BruhatTitsQuotient(13, 19)
sage: X.genus()
19
```

(continues on next page)

(continued from previous page)

```
sage: G = X.get_graph(); G
Multi-graph on 4 vertices
```

And an example with a Dirichlet character:

```
sage: f = DirichletGroup(6)[1]
sage: X = BruhatTitsQuotient(3,2*5*7,character = f)
sage: X.genus()
5
```

Note: A sage implementation of Eichler orders in rational quaternions algebras would remove the dependency on magma.

AUTHORS:

- Marc Masdeu (2012-02-20)

B_one()

Return the coordinates of 1 in the basis for the quaternion order.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: v,pow = X.B_one()
sage: X._conv(v) == 1
True
```

Nminus()

Return the discriminant of the relevant definite quaternion algebra.

OUTPUT:

An integer equal to N^-.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.Nminus()
7
```

Nplus()

Return the tame level N^+.

OUTPUT:

An integer equal to N^+.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7,1)
sage: X.Nplus()
1
```

dimension_harmonic_cocycles(*k*, *lev=None*, *Nplus=None*, *character=None*)

Compute the dimension of the space of harmonic cocycles of weight k on `self`.

OUTPUT:

An integer equal to the dimension

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,7)
sage: [X.dimension_harmonic_cocycles(k) for k in range(2,20,2)]
[1, 4, 4, 8, 8, 12, 12, 16, 16]

sage: X = BruhatTitsQuotient(2,5) # optional - magma
sage: [X.dimension_harmonic_cocycles(k) for k in range(2,40,2)] # optional -␣
↪magma
[0, 1, 3, 1, 3, 5, 3, 5, 7, 5, 7, 9, 7, 9, 11, 9, 11, 13, 11]

sage: X = BruhatTitsQuotient(7, 2 * 3 * 5)
sage: X.dimension_harmonic_cocycles(4)
12
sage: X = BruhatTitsQuotient(7, 2 * 3 * 5 * 11 * 13)
sage: X.dimension_harmonic_cocycles(2)
481
sage: X.dimension_harmonic_cocycles(4)
1440
```

e3()

Compute the e_3 invariant defined by the formula

$$e_k = \prod_{\ell | pN^-} \left(1 - \left(\frac{-3}{\ell}\right)\right) \prod_{\ell || N^+} \left(1 + \left(\frac{-3}{\ell}\right)\right) \prod_{\ell^2 | N^+} \nu_\ell(3)$$

OUTPUT:

an integer

EXAMPLES:

```
sage: X = BruhatTitsQuotient(31,3)
sage: X.e3
1
```

e4()

Compute the e_4 invariant defined by the formula

$$e_k = \prod_{\ell | pN^-} \left(1 - \left(\frac{-k}{\ell}\right)\right) \prod_{\ell || N^+} \left(1 + \left(\frac{-k}{\ell}\right)\right) \prod_{\ell^2 | N^+} \nu_\ell(k)$$

OUTPUT:

an integer

EXAMPLES:

```
sage: X = BruhatTitsQuotient(31,3)
sage: X.e4
2
```

embed(*g*, *exact=False*, *prec=None*)

Embed the quaternion element g into a matrix algebra.

INPUT:

- g a row vector of size 4 whose entries represent a quaternion in our basis.
- exact boolean (default: False) - If True, tries to embed g into a matrix algebra over a number field. If False, the target is the matrix algebra over $\mathbf{Q}_p$.

OUTPUT:

A 2x2 matrix with coefficients in $\mathbf{Q}_p$ if exact is False, or a number field if exact is True.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,2)
sage: l = X.get_units_of_order()
sage: len(l)
12
sage: l[3] # random
[-1]
[ 0]
[ 1]
[ 1]
sage: u = X.embed_quaternion(l[3]); u # random
[    O(7) 3 + O(7)]
[2 + O(7) 6 + O(7)]
sage: X._increase_precision(5)
sage: v = X.embed_quaternion(l[3]); v # random
[                7 + 3*7^2 + 7^3 + 4*7^4 + O(7^6)             3 + 7 + 3*7^2 + 7^
↪3 + 4*7^4 + O(7^6)]
[            2 + 7 + 3*7^2 + 7^3 + 4*7^4 + O(7^6) 6 + 5*7 + 3*7^2 + 5*7^3 + 2*7^
↪4 + 6*7^5 + O(7^6)]
sage: u == v
True
```

embed_quaternion(*g*, *exact=False*, *prec=None*)

Embed the quaternion element g into a matrix algebra.

INPUT:

- g a row vector of size 4 whose entries represent a quaternion in our basis.
- exact boolean (default: False) - If True, tries to embed g into a matrix algebra over a number field. If False, the target is the matrix algebra over $\mathbf{Q}_p$.

OUTPUT:

A 2x2 matrix with coefficients in $\mathbf{Q}_p$ if exact is False, or a number field if exact is True.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,2)
sage: l = X.get_units_of_order()
sage: len(l)
12
sage: l[3] # random
[-1]
[ 0]
[ 1]
[ 1]
sage: u = X.embed_quaternion(l[3]); u # random
[    O(7) 3 + O(7)]
```

(continues on next page)

(continued from previous page)

```
[2 + O(7) 6 + O(7)]
sage: X._increase_precision(5)
sage: v = X.embed_quaternion(l[3]); v # random
[                7 + 3*7^2 + 7^3 + 4*7^4 + O(7^6)             3 + 7 + 3*7^2 + 7^
↪3 + 4*7^4 + O(7^6)]
[            2 + 7 + 3*7^2 + 7^3 + 4*7^4 + O(7^6) 6 + 5*7 + 3*7^2 + 5*7^3 + 2*7^
↪4 + 6*7^5 + O(7^6)]
sage: u == v
True
```

fundom_rep(*v1*)

Find an equivalent vertex in the fundamental domain.

INPUT:

- v1 - a 2x2 matrix representing a normalized vertex.

OUTPUT:

A Vertex equivalent to v1, in the fundamental domain.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,7)
sage: M = Matrix(ZZ,2,2,[1,3,2,7])
sage: M.set_immutable()
sage: X.fundom_rep(M)
Vertex of Bruhat-Tits tree for p = 3
```

genus()

Compute the genus of the quotient graph using a formula This should agree with self.genus_no_formula().

Compute the genus of the Shimura curve corresponding to this quotient via Cerednik-Drinfeld. It is computed via a formula and not in terms of the quotient graph.

INPUT:

- level: Integer (default: None) a level. By default, use that of self.
- Nplus: Integer (default: None) a conductor. By default, use that of self.

OUTPUT:

An integer equal to the genus

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,2*5*31)
sage: X.genus()
21
sage: X.genus() == X.genus_no_formula()
True
```

genus_no_formula()

Compute the genus of the quotient from the data of the quotient graph. This should agree with self.genus().

OUTPUT:

An integer

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,2*3*29)
sage: X.genus_no_formula()
17
sage: X.genus_no_formula() == X.genus()
True
```

get_edge_list()

Return a list of Edge which represent a fundamental domain inside the Bruhat-Tits tree for the quotient.

OUTPUT:

A list of Edge.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(37,3)
sage: len(X.get_edge_list())
8
```

get_edge_stabilizers()

Compute the stabilizers in the arithmetic group of all edges in the Bruhat-Tits tree within a fundamental domain for the quotient graph. The stabilizers of an edge and its opposite are equal, and so we only store half the data.

OUTPUT:

A list of lists encoding edge stabilizers. It contains one entry for each edge. Each entry is a list of data corresponding to the group elements in the stabilizer of the edge. The data consists of: (0) a column matrix representing a quaternion, (1) the power of p that one needs to divide by in order to obtain a quaternion of norm 1, and hence an element of the arithmetic group Γ, (2) a boolean that is only used to compute spaces of modular forms.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,2)
sage: s = X.get_edge_stabilizers()
sage: len(s) == X.get_num_ordered_edges()/2
True
sage: len(s[0])
3
```

get_eichler_order(*magma=False*, *force_computation=False*)

Return the underlying Eichler order of level N^+.

OUTPUT:

An Eichler order.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.get_eichler_order()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis␣
↪(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

get_eichler_order_basis()

Return a basis for the global Eichler order.

OUTPUT:

Basis for the underlying Eichler order of level Nplus.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: X.get_eichler_order_basis()
[1/2 + 1/2*j, 1/2*i + 1/2*k, j, k]
```

get_eichler_order_quadform()

This function return the norm form for the underlying Eichler order of level `Nplus`. Required for finding elements in the arithmetic subgroup Gamma.

OUTPUT:

The norm form of the underlying Eichler order

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: X.get_eichler_order_quadform()
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 3 0 11 0 ]
[ * 3 0 11 ]
[ * * 11 0 ]
[ * * * 11 ]
```

get_eichler_order_quadmatrix()

This function returns the matrix of the quadratic form of the underlying Eichler order in the fixed basis.

OUTPUT:

A 4x4 integral matrix describing the norm form.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: X.get_eichler_order_quadmatrix()
[ 6  0 11  0]
[ 0  6  0 11]
[11  0 22  0]
[ 0 11  0 22]
```

get_embedding(*prec=None*)

Return a function which embeds quaternions into a matrix algebra.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,3)
sage: f = X.get_embedding(prec = 4)
sage: b = Matrix(ZZ,4,1,[1,2,3,4])
sage: f(b)
[2 + 3*5 + 2*5^2 + 4*5^3 + O(5^4)       3 + 2*5^2 + 4*5^3 + O(5^4)]
[        5 + 5^2 + 3*5^3 + O(5^4)           4 + 5 + 2*5^2 + O(5^4)]
```

get_embedding_matrix(*prec=None*, *exact=False*)

Return the matrix of the embedding.

INPUT:

- `exact` boolean (Default: `False`). If `True`, return an embedding into a matrix algebra with coefficients in a number field. Otherwise, embed into matrices over p-adic numbers.
- `prec` Integer (Default: `None`). If specified, return the matrix with precision `prec`. Otherwise, return the cached matrix (with the current working precision).

OUTPUT:

- A 4x4 matrix representing the embedding.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,2*3*5)
sage: X.get_embedding_matrix(4)
[                      1 + O(7^4)         5 + 2*7 + 3*7^3 + O(7^4) 4 + 5*7 +␣
↪6*7^2 + 6*7^3 + O(7^4)       6 + 3*7^2 + 4*7^3 + O(7^4)]
[                          O(7^4)                           O(7^4)             ␣
↪      3 + 7 + O(7^4) 1 + 6*7 + 3*7^2 + 2*7^3 + O(7^4)]
[                          O(7^4)         2 + 5*7 + 6*7^3 + O(7^4) 3 + 5*7 +␣
↪6*7^2 + 6*7^3 + O(7^4)         3 + 3*7 + 3*7^2 + O(7^4)]
[                      1 + O(7^4) 3 + 4*7 + 6*7^2 + 3*7^3 + O(7^4)             ␣
↪      3 + 7 + O(7^4) 1 + 6*7 + 3*7^2 + 2*7^3 + O(7^4)]
sage: X.get_embedding_matrix(3)
[                      1 + O(7^4)         5 + 2*7 + 3*7^3 + O(7^4) 4 + 5*7 +␣
↪6*7^2 + 6*7^3 + O(7^4)       6 + 3*7^2 + 4*7^3 + O(7^4)]
[                          O(7^4)                           O(7^4)             ␣
↪      3 + 7 + O(7^4) 1 + 6*7 + 3*7^2 + 2*7^3 + O(7^4)]
[                          O(7^4)         2 + 5*7 + 6*7^3 + O(7^4) 3 + 5*7 +␣
↪6*7^2 + 6*7^3 + O(7^4)         3 + 3*7 + 3*7^2 + O(7^4)]
[                      1 + O(7^4) 3 + 4*7 + 6*7^2 + 3*7^3 + O(7^4)             ␣
↪      3 + 7 + O(7^4) 1 + 6*7 + 3*7^2 + 2*7^3 + O(7^4)]
sage: X.get_embedding_matrix(5)
[                              1 + O(7^5)         5 + 2*7 + 3*7^3 + 6*7^4 + O(7^
↪5) 4 + 5*7 + 6*7^2 + 6*7^3 + 6*7^4 + O(7^5)       6 + 3*7^2 + 4*7^3 + 5*7^4 +␣
↪O(7^5)]
[                                  O(7^5)                                 O(7^
↪5)                           3 + 7 + O(7^5)   1 + 6*7 + 3*7^2 + 2*7^3 + 7^4 +␣
↪O(7^5)]
[                                  O(7^5)         2 + 5*7 + 6*7^3 + 5*7^4 + O(7^
↪5) 3 + 5*7 + 6*7^2 + 6*7^3 + 6*7^4 + O(7^5)         3 + 3*7 + 3*7^2 + 5*7^4 +␣
↪O(7^5)]
[                              1 + O(7^5)         3 + 4*7 + 6*7^2 + 3*7^3 + O(7^
↪5)                           3 + 7 + O(7^5)   1 + 6*7 + 3*7^2 + 2*7^3 + 7^4 +␣
↪O(7^5)]
```

get_extra_embedding_matrices()

Return a list of matrices representing the different embeddings.

Note: The precision is very low (currently set to 5 digits), since these embeddings are only used to apply a character.

EXAMPLES:

This portion of the code is only relevant when working with a nontrivial Dirichlet character. If there is no such character then the code returns an empty list. Even if the character is not trivial it might return an empty list:

```
sage: f = DirichletGroup(6)[1]
sage: X = BruhatTitsQuotient(3,2*5*7,character = f)
sage: X.get_extra_embedding_matrices()
[]
```

```
sage: f = DirichletGroup(6)[1]
sage: X = BruhatTitsQuotient(5,2,3, character = f, use_magma=True) # optional -␣
↪magma
sage: X.get_extra_embedding_matrices() # optional - magma
[
[1 0 2 0]
[0 0 2 0]
[0 0 0 0]
[1 0 2 2]
]
```

get_fundom_graph()

Return the fundamental domain (and computes it if needed).

OUTPUT:

A fundamental domain for the action of Γ.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(11,5)
sage: X.get_fundom_graph()
Graph on 24 vertices
```

get_generators()

Use a fundamental domain in the Bruhat-Tits tree, and certain gluing data for boundary vertices, in order to compute a collection of generators for the arithmetic quaternionic group that one is quotienting by. This is analogous to using a polygonal rep. of a compact real surface to present its fundamental domain.

OUTPUT:

- A generating list of elements of an arithmetic quaternionic group.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,2)
sage: len(X.get_generators())
2
```

get_graph()

Return the quotient graph (and compute it if needed).

OUTPUT:

A graph representing the quotient of the Bruhat-Tits tree.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(11,5)
sage: X.get_graph()
Multi-graph on 2 vertices
```

get_list()

Return a list of Edge which represent a fundamental domain inside the Bruhat-Tits tree for the quotient, together with a list of the opposite edges. This is used to work with automorphic forms.

OUTPUT:

A list of Edge.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(37,3)
sage: len(X.get_list())
16
```

get_maximal_order(*magma=False*, *force_computation=False*)

Return the underlying maximal order containing the Eichler order.

OUTPUT:

A maximal order.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.get_maximal_order()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis␣
↪(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

get_nontorsion_generators()

Use a fundamental domain in the Bruhat-Tits tree, and certain gluing data for boundary vertices, in order to compute a collection of generators for the nontorsion part of the arithmetic quaternionic group that one is quotienting by. This is analogous to using a polygonal rep. of a compact real surface to present its fundamental domain.

OUTPUT:

- A generating list of elements of an arithmetic quaternionic group.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,13)
sage: len(X.get_nontorsion_generators())
3
```

get_num_ordered_edges()

Return the number of ordered edges E in the quotient using the formula relating the genus g with the number of vertices V and that of unordered edges $E/2$: $E = 2(g + V - 1)$.

OUTPUT:

- An integer

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,2)
sage: X.get_num_ordered_edges()
2
```

get_num_verts()

Return the number of vertices in the quotient using the formula $V = 2(\mu/12 + e_3/3 + e_4/4)$.

OUTPUT:

- An integer (the number of vertices)

EXAMPLES:

```
sage: X = BruhatTitsQuotient(29,11)
sage: X.get_num_verts()
4
```

get_quaternion_algebra()

Return the underlying quaternion algebra.

OUTPUT:

The underlying definite quaternion algebra

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.get_quaternion_algebra()
Quaternion Algebra (-1, -7) with base ring Rational Field
```

get_splitting_field()

Return a quadratic field that splits the quaternion algebra attached to `self`. Currently requires Magma.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,11)
sage: X.get_splitting_field()
Traceback (most recent call last):
...
NotImplementedError: Sage does not know yet how to work with the kind of orders
↪that you are trying to use. Try installing Magma first and set it up so that
↪Sage can use it.
```

If we do have Magma installed, then it works:

```
sage: X = BruhatTitsQuotient(5,11,use_magma=True) # optional - magma
sage: X.get_splitting_field() # optional - magma
Number Field in a with defining polynomial X1^2 + 11
```

get_stabilizers()

Compute the stabilizers in the arithmetic group of all edges in the Bruhat-Tits tree within a fundamental domain for the quotient graph. This is similar to get_edge_stabilizers, except that here we also store the stabilizers of the opposites.

OUTPUT:

A list of lists encoding edge stabilizers. It contains one entry for each edge. Each entry is a list of data corresponding to the group elements in the stabilizer of the edge. The data consists of: (0) a column matrix representing a quaternion, (1) the power of p that one needs to divide by in order to obtain a quaternion of norm 1, and hence an element of the arithmetic group Γ, (2) a boolean that is only used to compute spaces of modular forms.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,5)
sage: s = X.get_stabilizers()
sage: len(s) == X.get_num_ordered_edges()
True
sage: gamma = X.embed_quaternion(s[1][0][0][0],prec = 20)
sage: v = X.get_edge_list()[0].rep
sage: X._BT.edge(gamma*v) == v
True
```

get_units_of_order()

Return the units of the underlying Eichler $\mathbf{Z}$-order. This is a finite group since the order lives in a definite quaternion algebra over $\mathbf{Q}$.

OUTPUT:

A list of elements of the global Eichler $\mathbf{Z}$-order of level N^+.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: X.get_units_of_order()
[
[ 0]  [-2]
[-2]  [ 0]
[ 0]  [ 1]
[ 1], [ 0]
]
```

get_vertex_dict()

This function returns the vertices of the quotient viewed as a dict.

OUTPUT:

A python dict with the vertices of the quotient.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(37,3)
sage: X.get_vertex_dict()
{[1 0]
[0 1]: Vertex of Bruhat-Tits tree for p = 37, [ 1  0]
[ 0 37]: Vertex of Bruhat-Tits tree for p = 37}
```

get_vertex_list()

Return a list of the vertices of the quotient.

OUTPUT:

- A list with the vertices of the quotient.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(37,3)
sage: X.get_vertex_list()
[Vertex of Bruhat-Tits tree for p = 37, Vertex of Bruhat-Tits tree for p = 37]
```

get_vertex_stabs()

This function computes the stabilizers in the arithmetic group of all vertices in the Bruhat-Tits tree within a fundamental domain for the quotient graph.

OUTPUT:

A list of vertex stabilizers. Each vertex stabilizer is a finite cyclic subgroup, so we return generators for these subgroups.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(13,2)
sage: S = X.get_vertex_stabs()
sage: gamma = X.embed_quaternion(S[0][0][0],prec = 20)
sage: v = X.get_vertex_list()[0].rep
sage: X._BT.vertex(gamma*v) == v
True
```

harmonic_cocycle_from_elliptic_curve(*E*, *prec=None*)

Return a harmonic cocycle with the same Hecke eigenvalues as E.

Given an elliptic curve E having a conductor N of the form pN^-N^+, return the harmonic cocycle over `self` which is attached to E via modularity. The result is only well-defined up to scaling.

INPUT:

- E – an elliptic curve over the rational numbers
- `prec` – (default: None) If specified, the harmonic cocycle will take values in $\mathbf{Q}_p$ with precision `prec`. Otherwise it will take values in $\mathbf{Z}$.

OUTPUT:

A harmonic cocycle attached via modularity to the given elliptic curve.

EXAMPLES:

```
sage: E = EllipticCurve('21a1')
sage: X = BruhatTitsQuotient(7,3)
sage: f = X.harmonic_cocycle_from_elliptic_curve(E,10)
sage: T29 = f.parent().hecke_operator(29)
sage: T29(f) == E.ap(29) * f
True
sage: E = EllipticCurve('51a1')
sage: X = BruhatTitsQuotient(3,17)
sage: f = X.harmonic_cocycle_from_elliptic_curve(E,20)
sage: T31 = f.parent().hecke_operator(31)
sage: T31(f) == E.ap(31) * f
True
```

harmonic_cocycles(*k*, *prec=None*, *basis_matrix=None*, *base_field=None*)

Compute the space of harmonic cocycles of a given even weight `k`.

INPUT:

- `k` - integer - The weight. It must be even.
- `prec` - integer (default: None). If specified, the precision for the coefficient module
- `basis_matrix` - a matrix (default: None).
- `base_field` - a ring (default: None)

OUTPUT: A space of harmonic cocycles

EXAMPLES:

```
sage: X = BruhatTitsQuotient(31,7)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: H
Space of harmonic cocycles of weight 2 on Quotient of the Bruhat Tits tree of␣
↪GL_2(QQ_31) with discriminant 7 and level 1
sage: H.basis()[0]
Harmonic cocycle with values in Sym^0 Q_31^2
```

is_admissible(*D*)

Test whether the imaginary quadratic field of discriminant D embeds in the quaternion algebra. It furthermore tests the Heegner hypothesis in this setting (e.g., is p inert in the field, etc).

INPUT:

- D - an integer whose squarefree part will define the quadratic field

OUTPUT:

A boolean describing whether the quadratic field is admissible

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: [X.is_admissible(D) for D in range(-1,-20,-1)]
[False, True, False, False, False, False, False, True, False, False, False,␣
↪False, False, False, False, False, False, True, False]
```

level()

Return pN^-, which is the discriminant of the indefinite quaternion algebra that is uniformed by Cerednik-Drinfeld.

OUTPUT:

An integer equal to pN^-.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.level()
35
```

mu()

Compute the mu invariant of self.

OUTPUT:

An integer.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(29,3)
sage: X.mu
2
```

padic_automorphic_forms(*U*, *prec=None*, *t=None*, *R=None*, *overconvergent=False*)

The module of (quaternionic) p-adic automorphic forms over `self`.

INPUT:

- U – A distributions module or an integer. If U is a distributions module then this creates the relevant space of automorphic forms. If U is an integer then the coefficients are the $(U-2)$nd power of the symmetric representation of $GL_2(\mathbf{Q}_p)$.
- prec – A precision (default : None). If not None should be a positive integer.
- t – (default : None). The number of additional moments to store. If None, determine it automatically from prec, U and the overconvergent flag.
- R – (default : None). If specified, coefficient field of the automorphic forms. If not specified it defaults to the base ring of the distributions U, or to $\mathbf{Q}_p$ with the working precision prec.
- overconvergent – Boolean (default = False). If True, will construct overconvergent p-adic automorphic forms. Otherwise it constructs the finite dimensional space of p-adic automorphic forms which is isomorphic to the space of harmonic cocycles.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(11,5)
sage: X.padic_automorphic_forms(2,prec=10)
Space of automorphic forms on Quotient of the Bruhat Tits tree of GL_2(QQ_11)␣
↪with discriminant 5 and level 1 with values in Sym^0 Q_11^2
```

plot(**args*, ***kwargs*)

Plot the quotient graph.

OUTPUT:

A plot of the quotient graph

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,23)
sage: X.plot()
Graphics object consisting of 17 graphics primitives
```

plot_fundom(**args*, ***kwargs*)

Plot a fundamental domain.

OUTPUT:

A plot of the fundamental domain.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,23)
sage: X.plot_fundom()
Graphics object consisting of 88 graphics primitives
```

prime()

Return the prime one is working with.

OUTPUT:

An integer equal to the fixed prime p

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: X.prime()
5
```

class sage.modular.btquotients.btquotient.**BruhatTitsTree**(*p*)

Bases: sage.structure.sage_object.SageObject, sage.structure.unique_representation.UniqueRepresentation

An implementation of the Bruhat-Tits tree for $GL_2(\mathbf{Q}_p)$.

INPUT:

- p - a prime number. The corresponding tree is then $p+1$ regular

EXAMPLES:

We create the tree for $GL_2(\mathbf{Q}_5)$:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 5
sage: T = BruhatTitsTree(p)
sage: m = Matrix(ZZ,2,2,[p**5,p**2,p**3,1+p+p*3])
sage: e = T.edge(m); e
[  0  25]
[625  21]
sage: v0 = T.origin(e); v0
[ 25   0]
[ 21 125]
sage: v1 = T.target(e); v1
[ 25   0]
[ 21 625]
sage: T.origin(T.opposite(e)) == v1
True
sage: T.target(T.opposite(e)) == v0
True
```

A value error is raised if a prime is not passed:

```
sage: T = BruhatTitsTree(4)
Traceback (most recent call last):
...
ValueError: Input (4) must be prime
```

AUTHORS:

- Marc Masdeu (2012-02-20)

edge(*M*)

Normalize a matrix to the correct normalized edge representative.

INPUT:

- M - a 2x2 integer matrix

OUTPUT:

- newM - a 2x2 integer matrix

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: T = BruhatTitsTree(3)
sage: T.edge( Matrix(ZZ,2,2,[0,-1,3,0]) )
```

(continues on next page)

(continued from previous page)

```
[0 1]
[3 0]
```

edge_between_vertices(*v1*, *v2*, *normalized=False*)

Compute the normalized matrix rep. for the edge passing between two vertices.

INPUT:

- v1 - 2x2 integer matrix
- v2 - 2x2 integer matrix
- normalized - boolean (default: False), whether the vertices are normalized.

OUTPUT:

- 2x2 integer matrix, representing the edge from v1 to v2. If v1 and v2 are not at distance 1, raise a ValueError.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 7
sage: T = BruhatTitsTree(p)
sage: v1 = T.vertex(Matrix(ZZ,2,2,[p,0,0,1])); v1
[7 0]
[0 1]
sage: v2 = T.vertex(Matrix(ZZ,2,2,[p,1,0,1])); v2
[1 0]
[1 7]
sage: T.edge_between_vertices(v1,v2)
Traceback (most recent call last):
...
ValueError: Vertices are not adjacent.

sage: v3 = T.vertex(Matrix(ZZ,2,2,[1,0,0,1])); v3
[1 0]
[0 1]
sage: T.edge_between_vertices(v1,v3)
[0 1]
[1 0]
```

edges_leaving_origin()

Find normalized representatives for the $p+1$ edges leaving the origin vertex corresponding to the homothety class of $\mathbf{Z}_p^2$. These are cached.

OUTPUT:

- A list of size $p + 1$ of 2x2 integer matrices

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: T = BruhatTitsTree(3)
sage: T.edges_leaving_origin()
[
[0 1]  [3 0]  [0 1]  [0 1]
```

(continues on next page)

(continued from previous page)

```
[3 0], [0 1], [3 1], [3 2]
]
```

entering_edges(*v*)

This function returns the edges entering a given vertex.

INPUT:

- v - 2x2 integer matrix

OUTPUT:

A list of size $p + 1$ of 2x2 integer matrices

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 7
sage: T = BruhatTitsTree(p)
sage: T.entering_edges(Matrix(ZZ,2,2,[1,0,0,1]))
[
[1 0]  [0 1]  [1 0]  [1 0]  [1 0]  [1 0]  [1 0]  [1 0]
[0 1], [1 0], [1 1], [4 1], [5 1], [2 1], [3 1], [6 1]
]
```

find_containing_affinoid(*z*)

Return the vertex corresponding to the affinoid in the p-adic upper half plane that a given (unramified!) point reduces to.

INPUT:

- z - an element of an unramified extension of $\mathbf{Q}_p$ that is not contained in $\mathbf{Q}_p$.

OUTPUT:

A 2x2 integer matrix representing a vertex of `self`.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: T = BruhatTitsTree(5)
sage: K.<a> = Qq(5^2,20)
sage: T.find_containing_affinoid(a)
[1 0]
[0 1]
sage: z = 5*a+3
sage: v = T.find_containing_affinoid(z).inverse(); v
[   1    0]
[-2/5  1/5]
```

Note that the translate of z belongs to the standard affinoid. That is, it is a p-adic unit and its reduction modulo p is not in $\mathbf{F}_p$:

```
sage: gz = (v[0,0]*z+v[0,1])/(v[1,0]*z+v[1,1]); gz
(a + 1) + O(5^19)
sage: gz.valuation() == 0
True
```

find_covering(*z1*, *z2*, *level=0*)

Compute a covering of $P^1(\mathbf{Q}_p)$ adapted to a certain geodesic in self.

More precisely, the p-adic upper half plane points z1 and z2 reduce to vertices v_1, v_2. The returned covering consists of all the edges leaving the geodesic from v_1 to v_2.

INPUT:

- z1, z2 - unramified algebraic points of h_p

OUTPUT:

a list of 2x2 integer matrices representing edges of self

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 3
sage: K.<a> = Qq(p^2)
sage: T = BruhatTitsTree(p)
sage: z1 = a + a*p
sage: z2 = 1 + a*p + a*p^2 - p^6
sage: T.find_covering(z1,z2)
[
[0 1]  [3 0]  [0 1]  [0 1]  [0 1]  [0 1]
[3 0], [0 1], [3 2], [9 1], [9 4], [9 7]
]
```

Note: This function is used to compute certain Coleman integrals on P^1. That's why the input consists of two points of the p-adic upper half plane, but decomposes $P^1(\mathbf{Q}_p)$. This decomposition is what allows us to represent the relevant integrand as a locally analytic function. The z1 and z2 appear in the integrand.

find_geodesic(*v1*, *v2*, *normalized=True*)

This function computes the geodesic between two vertices

INPUT:

- v1 - 2x2 integer matrix representing a vertex
- v2 - 2x2 integer matrix representing a vertex
- normalized - boolean (default: True)

OUTPUT:

An ordered list of 2x2 integer matrices representing the vertices of the paths joining v1 and v2.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 3
sage: T = BruhatTitsTree(p)
sage: v1 = T.vertex( Matrix(ZZ,2,2,[p^3, 0, 1, p^1]) ); v1
[27  0]
[ 1  3]
sage: v2 = T.vertex( Matrix(ZZ,2,2,[p,2,0,p]) ); v2
[1 0]
[6 9]
```

(continues on next page)

(continued from previous page)

```
sage: T.find_geodesic(v1,v2)
[
[27  0]  [27  0]  [9 0]  [3 0]  [1 0]  [1 0]  [1 0]
[ 1  3], [ 0  1], [0 1], [0 1], [0 1], [0 3], [6 9]
]
```

find_path(*v*, *boundary=None*)

Compute a path from a vertex to a given set of so-called boundary vertices, whose interior must contain the origin vertex. In the case that the boundary is not specified, it computes the geodesic between the given vertex and the origin. In the case that the boundary contains more than one vertex, it computes the geodesic to some point of the boundary.

INPUT:

- `v` - a 2x2 matrix representing a vertex `boundary`
- a list of matrices (default: None). If omitted, finds the geodesic from `v` to the central vertex.

OUTPUT:

An ordered list of vertices describing the geodesic from `v` to `boundary`, followed by the vertex in the boundary that is closest to `v`.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 3
sage: T = BruhatTitsTree(p)
sage: T.find_path( Matrix(ZZ,2,2,[p^4,0,0,1]) )
(
[[81  0]
[ 0  1], [27  0]
[ 0  1], [9 0]
[0 1], [3 0]      [1 0]
[0 1]]          , [0 1]
)
sage: T.find_path( Matrix(ZZ,2,2,[p^3,0,134,p^2]) )
(
[[27  0]
[ 8  9], [27  0]
[ 2  3], [27  0]
[ 0  1], [9 0]
[0 1], [3 0]      [1 0]
[0 1]]          , [0 1]
)
```

get_balls(*center=1*, *level=1*)

Return a decomposition of $P^1(\mathbf{Q}_p)$ into compact open balls.

Each vertex in the Bruhat-Tits tree gives a decomposition of $P^1(\mathbf{Q}_p)$ into $p+1$ open balls. Each of these balls may be further subdivided, to get a finer decomposition.

This function returns the decomposition of $P^1(\mathbf{Q}_p)$ corresponding to `center` into $(p+1)p^{\text{level}}$ balls.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 2
sage: T = BruhatTitsTree(p)
sage: T.get_balls(Matrix(ZZ,2,2,[p,0,0,1]),1)
[
[0 1]  [0 1]  [8 0]  [0 4]  [0 2]  [0 2]
[2 0], [2 1], [0 1], [2 1], [4 1], [4 3]
]
```

leaving_edges(*M*)

Return edges leaving a vertex

INPUT:

- `M` - 2x2 integer matrix

OUTPUT:

List of size $p + 1$ of 2x2 integer matrices

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 7
sage: T = BruhatTitsTree(p)
sage: T.leaving_edges(Matrix(ZZ,2,2,[1,0,0,1]))
[
[0 1]  [7 0]  [0 1]  [0 1]  [0 1]  [0 1]  [0 1]  [0 1]
[7 0], [0 1], [7 1], [7 4], [7 5], [7 2], [7 3], [7 6]
]
```

opposite(*e*)

This function returns the edge oriented oppositely to a given edge.

INPUT:

- e - 2x2 integer matrix

OUTPUT:

2x2 integer matrix

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 7
sage: T = BruhatTitsTree(p)
sage: e = Matrix(ZZ,2,2,[1,0,0,1])
sage: T.opposite(e)
[0 1]
[7 0]
sage: T.opposite(T.opposite(e)) == e
True
```

origin(*e*, *normalized=False*)

Return the origin vertex of the edge represented by the input matrix e.

INPUT:

- e - a 2x2 matrix with integer entries

- `normalized` - boolean (default: false). If True then the input matrix M is assumed to be normalized

OUTPUT:

- `e` - A 2x2 integer matrix

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: T = BruhatTitsTree(7)
sage: T.origin(Matrix(ZZ,2,2,[1,5,8,9]))
[1 0]
[1 7]
```

subdivide(*edgelist*, *level*)

(Ordered) edges of self may be regarded as open balls in $P^1(\mathbf{Q}_p)$. Given a list of edges, this function return a list of edges corresponding to the level-th subdivision of the corresponding opens. That is, each open ball of the input is broken up into p^{level} subballs of equal radius.

INPUT:

- `edgelist` - a list of edges
- `level` - an integer

OUTPUT:

A list of 2x2 integer matrices

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 3
sage: T = BruhatTitsTree(p)
sage: T.subdivide([Matrix(ZZ,2,2,[p,0,0,1])],2)
[
[27  0]  [0 9]  [0 9]  [0 3]  [0 3]  [0 3]  [0 3]  [0 3]  [0 3]
[ 0  1], [3 1], [3 2], [9 1], [9 4], [9 7], [9 2], [9 5], [9 8]
]
```

target(*e*, *normalized=False*)

Return the target vertex of the edge represented by the input matrix e.

INPUT:

- `e` - a 2x2 matrix with integer entries
- **`normalized` - boolean (default: false). If True** then the input matrix is assumed to be normalized.

OUTPUT:

- `e` - 2x2 integer matrix representing the target of the input edge

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: T = BruhatTitsTree(7)
sage: T.target(Matrix(ZZ,2,2,[1,5,8,9]))
[1 0]
[0 1]
```

vertex(*M*)

Normalize a matrix to the corresponding normalized vertex representative

INPUT:

- `M` - 2x2 integer matrix

OUTPUT:

- a 2x2 integer matrix

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import BruhatTitsTree
sage: p = 5
sage: T = BruhatTitsTree(p)
sage: m = Matrix(ZZ,2,2,[p**5,p**2,p**3,1+p+p*3])
sage: e = T.edge(m)
sage: t = m.inverse()*e
sage: scaling = Qp(p,20)(t.determinant()).sqrt()
sage: t = 1/scaling * t
sage: min([t[ii,jj].valuation(p) for ii in range(2) for jj in range(2)]) >= 0
True
sage: t[1,0].valuation(p) > 0
True
```

class sage.modular.btquotients.btquotient.**DoubleCosetReduction**(*Y*, *x*, *extrapow=0*)

Bases: `sage.structure.sage_object.SageObject`

Edges in the Bruhat-Tits tree are represented by cosets of matrices in GL_2. Given a matrix x in GL_2, this class computes and stores the data corresponding to the double coset representation of x in terms of a fundamental domain of edges for the action of the arithmetic group Γ.

More precisely:

Initialized with an element x of $GL_2(\mathbf{Z})$, finds elements γ in Γ, t and an edge e such that $get = x$. It stores these values as members `gamma`, `label` and functions `self.sign()`, `self.t()` and `self.igamma()`, satisfying:

- if `self.sign() == +1`: `igamma() * edge_list[label].rep * t() == x`
- if `self.sign() == -1`: `igamma() * edge_list[label].opposite.rep * t() == x`

It also stores a member called power so that:

```
p**(2*power) = gamma.reduced_norm()
```

The usual decomposition $get = x$ would be:

- g = gamma / (p ** power)
- e = edge_list[label]
- t' = t * p ** power

Here usual denotes that we have rescaled gamma to have unit determinant, and so that the result is honestly an element of the arithmetic quaternion group under consideration. In practice we store integral multiples and keep track of the powers of p.

INPUT:

- `Y` - BruhatTitsQuotient object in which to work
- **`x` - Something coercible into a matrix in $GL_2(\mathbf{Z})$. In** principle we should allow elements in $GL_2(\mathbf{Q}_p)$, but it is enough to work with integral entries

- **extrapow - gets added to the power attribute, and it is** used for the Hecke action.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import DoubleCosetReduction
sage: Y = BruhatTitsQuotient(5, 13)
sage: x = Matrix(ZZ,2,2,[123,153,1231,1231])
sage: d = DoubleCosetReduction(Y,x)
sage: d.sign()
-1
sage: d.igamma()*Y._edge_list[d.label - len(Y.get_edge_list())].opposite.rep*d.t()␣
↪== x
True
sage: x = Matrix(ZZ,2,2,[1423,113553,11231,12313])
sage: d = DoubleCosetReduction(Y,x)
sage: d.sign()
1
sage: d.igamma()*Y._edge_list[d.label].rep*d.t() == x
True
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu

igamma(*embedding=None*, *scale=1*)

Image under gamma.

Elements of the arithmetic group can be regarded as elements of the global quaternion order, and hence may be represented exactly. This function computes the image of such an element under the local splitting and returns the corresponding p-adic approximation.

INPUT:

- `embedding` - an integer, or a function (default: none). If `embedding` is None, then the image of `self.gamma` under the local splitting associated to `self.Y` is used. If `embedding` is an integer, then the precision of the local splitting of self.Y is raised (if necessary) to be larger than this integer, and this new local splitting is used. If a function is passed, then map `self.gamma` under `embedding`.
- `scale` – (default: 1) scaling factor applied to the output

OUTPUT:

a 2x2 matrix with p-adic entries encoding the image of `self` under the local splitting

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import DoubleCosetReduction
sage: Y = BruhatTitsQuotient(7, 11)
sage: d = DoubleCosetReduction(Y,Matrix(ZZ,2,2,[123,45,88,1]))
sage: d.igamma()
[6 + 6*7 + 6*7^2 + 6*7^3 + 6*7^4 + O(7^5)                                   O(7^
↪5)]
[                                   O(7^5) 6 + 6*7 + 6*7^2 + 6*7^3 + 6*7^4 + O(7^
↪5)]
sage: d.igamma(embedding = 7)
```

(continues on next page)

(continued from previous page)

```
[6 + 6*7 + 6*7^2 + 6*7^3 + 6*7^4 + 6*7^5 + 6*7^6 + O(7^7)                        ␣
↪                            O(7^7)]
[                                                    O(7^7) 6 + 6*7 + 6*7^2 + 6*7^
↪3 + 6*7^4 + 6*7^5 + 6*7^6 + O(7^7)]
```

sign()

Return the direction of the edge.

The Bruhat-Tits quotients are directed graphs but we only store half the edges (we treat them more like unordered graphs). The sign tells whether the matrix self.x is equivalent to the representative in the quotient (sign = +1), or to the opposite of one of the representatives (sign = -1).

OUTPUT:

an int that is +1 or -1 according to the sign of `self`

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import DoubleCosetReduction
sage: Y = BruhatTitsQuotient(3, 11)
sage: x = Matrix(ZZ,2,2,[123,153,1231,1231])
sage: d = DoubleCosetReduction(Y,x)
sage: d.sign()
-1
sage: d.igamma()*Y._edge_list[d.label - len(Y.get_edge_list())].opposite.rep*d.
↪t() == x
True
sage: x = Matrix(ZZ,2,2,[1423,113553,11231,12313])
sage: d = DoubleCosetReduction(Y,x)
sage: d.sign()
1
sage: d.igamma()*Y._edge_list[d.label].rep*d.t() == x
True
```

t(*prec=None*)

Return the 't part' of the decomposition using the rest of the data.

INPUT:

- `prec` - a p-adic precision that t will be computed to. Defaults to the default working precision of self.

OUTPUT:

a 2x2 p-adic matrix with entries of precision `prec` that is the 't-part' of the decomposition of self

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import DoubleCosetReduction
sage: Y = BruhatTitsQuotient(5, 13)
sage: x = Matrix(ZZ,2,2,[123,153,1231,1232])
sage: d = DoubleCosetReduction(Y,x)
sage: t = d.t(20)
sage: t[1,0].valuation() > 0
True
```

class `sage.modular.btquotients.btquotient.`**Edge**(*p*, *label*, *rep*, *origin*, *target*, *links=None*, *opposite=None*, *determinant=None*, *valuation=None*)

Bases: `sage.structure.sage_object.SageObject`

This is a structure to represent edges of quotients of the Bruhat-Tits tree. It is useful to enrich the representation of an edge as a matrix with extra data.

INPUT:

- `p` - a prime integer.
- `label` - An integer which uniquely identifies this edge.
- `rep` - A 2x2 matrix in reduced form representing this edge.
- `origin` - The origin vertex of `self`.
- `target` - The target vertex of `self`.
- `links` - (Default: empty list) A list of elements of Γ which identify different edges in the Bruhat-Tits tree which are equivalent to `self`.
- `opposite` - (Default: None) The edge opposite to `self`
- `determinant` - (Default: None) The determinant of `rep`, if known.
- `valuation` - (Default: None) The valuation of the determinant of `rep`, if known.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import Edge, Vertex
sage: v1 = Vertex(7,0,Matrix(ZZ,2,2,[1,2,3,18]))
sage: v2 = Vertex(7,0,Matrix(ZZ,2,2,[3,2,1,18]))
sage: e1 = Edge(7,0,Matrix(ZZ,2,2,[1,2,3,18]),v1,v2)
sage: e1.rep
[ 1  2]
[ 3 18]
```

AUTHORS:

- Marc Masdeu (2012-02-20)

class `sage.modular.btquotients.btquotient.`**`Vertex`**(*p*, *label*, *rep*, *leaving_edges=None*, *entering_edges=None*, *determinant=None*, *valuation=None*)

Bases: `sage.structure.sage_object.SageObject`

This is a structure to represent vertices of quotients of the Bruhat-Tits tree. It is useful to enrich the representation of the vertex as a matrix with extra data.

INPUT:

- `p` - a prime integer.
- `label` - An integer which uniquely identifies this vertex.
- `rep` - A 2x2 matrix in reduced form representing this vertex.
- `leaving_edges` - (default: empty list) A list of edges leaving this vertex.
- `entering_edges` - (default: empty list) A list of edges entering this vertex.
- `determinant` - (default: None) The determinant of `rep`, if known.
- `valuation` - (default: None) The valuation of the determinant of `rep`, if known.

EXAMPLES:

```
sage: from sage.modular.btquotients.btquotient import Vertex
sage: v1 = Vertex(5,0,Matrix(ZZ,2,2,[1,2,3,18]))
sage: v1.rep
[ 1  2]
[ 3 18]
sage: v1.entering_edges
[]
```

AUTHORS:

- Marc Masdeu (2012-02-20)

19.8 Spaces of p-adic automorphic forms

Compute with harmonic cocycles and p-adic automorphic forms, including overconvergent p-adic automorphic forms.

For a discussion of nearly rigid analytic modular forms and the rigid analytic Shimura-Maass operator, see [Fra2011]. It is worth also looking at [FM2014] for information on how these are implemented in this code.

EXAMPLES:

Create a quotient of the Bruhat-Tits tree:

```
sage: X = BruhatTitsQuotient(13,11)
```

Declare the corresponding space of harmonic cocycles:

```
sage: H = X.harmonic_cocycles(2,prec=5)
```

And the space of p-adic automorphic forms:

```
sage: A = X.padic_automorphic_forms(2,prec=5,overconvergent=True)
```

Harmonic cocycles, unlike p-adic automorphic forms, can be used to compute a basis:

```
sage: a = H.gen(0)
```

This can then be lifted to an overconvergent p-adic modular form:

```
sage: A.lift(a) # long time
p-adic automorphic form of cohomological weight 0
```

class sage.modular.btquotients.pautomorphicform.**BruhatTitsHarmonicCocycleElement**(*_parent*, *vec*)

Bases: sage.modular.hecke.element.HeckeModuleElement

Γ-invariant harmonic cocycles on the Bruhat-Tits tree. Γ-invariance is necessary so that the cocycle can be stored in terms of a finite amount of data.

More precisely, given a BruhatTitsQuotient T, harmonic cocycles are stored as a list of values in some coefficient module (e.g. for weight 2 forms can take $\mathbf{C}_p$) indexed by edges of a fundamental domain for T in the Bruhat-Tits tree. Evaluate the cocycle at other edges using Gamma invariance (although the values may not be equal over an orbit of edges as the coefficient module action may be nontrivial).

EXAMPLES:

Harmonic cocycles form a vector space, so they can be added and/or subtracted from each other:

```
sage: X = BruhatTitsQuotient(5,23)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: v1 = H.basis()[0]; v2 = H.basis()[1] # indirect doctest
sage: v3 = v1+v2
sage: v1 == v3-v2
True
```

and rescaled:

```
sage: v4 = 2*v1
sage: v1 == v4 - v1
True
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu

derivative(*z=None*, *level=0*, *order=1*)

Integrate Teitelbaum's p-adic Poisson kernel against the measure corresponding to `self` to evaluate the rigid analytic Shimura-Maass derivatives of the associated modular form at z.

If `z = None`, a function is returned that encodes the derivative of the modular form.

Note: This function uses the integration method of Riemann summation and is incredibly slow! It should only be used for testing and bug-finding. Overconvergent methods are quicker.

INPUT:

- `z` - an element in the quadratic unramified extension of $\mathbf{Q}_p$ that is not contained in $\mathbf{Q}_p$ (default = None). If `z = None` then a function encoding the derivative is returned.
- `level` - an integer. How fine of a mesh should the Riemann sum use.
- `order` - an integer. How many derivatives to take.

OUTPUT:

An element of the quadratic unramified extension of $\mathbf{Q}_p$, or a function encoding the derivative.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,23)
sage: H = X.harmonic_cocycles(2,prec=5)
sage: b = H.basis()[0]
sage: R.<a> = Qq(9,prec=10)
sage: b.modular_form(a,level=0) == b.derivative(a,level=0,order=0)
True
sage: b.derivative(a,level=1,order=1)
(2*a + 2)*3 + (a + 2)*3^2 + 2*a*3^3 + 2*3^4 + O(3^5)
sage: b.derivative(a,level=2,order=1)
(2*a + 2)*3 + 2*a*3^2 + 3^3 + a*3^4 + O(3^5)
```

evaluate(*e1*)

Evaluate a harmonic cocycle on an edge of the Bruhat-Tits tree.

INPUT:

- e1 - a matrix corresponding to an edge of the Bruhat-Tits tree

OUTPUT:

- An element of the coefficient module of the cocycle which describes the value of the cocycle on e1

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,17)
sage: e0 = X.get_edge_list()[0]
sage: e1 = X.get_edge_list()[1]
sage: H = X.harmonic_cocycles(2,prec=10)
sage: b = H.basis()[0]
sage: b.evaluate(e0.rep)
1 + O(5^10)
sage: b.evaluate(e1.rep)
4 + 4*5 + 4*5^2 + 4*5^3 + 4*5^4 + 4*5^5 + 4*5^6 + 4*5^7 + 4*5^8 + 4*5^9 + O(5^
↪10)
```

modular_form(*z=None*, *level=0*)

Integrate Teitelbaum's p-adic Poisson kernel against the measure corresponding to self to evaluate the associated modular form at z.

If z = None, a function is returned that encodes the modular form.

Note: This function uses the integration method of Riemann summation and is incredibly slow! It should only be used for testing and bug-finding. Overconvergent methods are quicker.

INPUT:

- z - an element in the quadratic unramified extension of $\mathbf{Q}_p$ that is not contained in $\mathbf{Q}_p$ (default = None).
- level - an integer. How fine of a mesh should the Riemann sum use.

OUTPUT:

An element of the quadratic unramified extension of $\mathbf{Q}_p$.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,23)
sage: H = X.harmonic_cocycles(2,prec = 8)
sage: b = H.basis()[0]
sage: R.<a> = Qq(9,prec=10)
sage: x1 = b.modular_form(a,level = 0); x1
a + (2*a + 1)*3 + (a + 1)*3^2 + (a + 1)*3^3 + 3^4 + (a + 2)*3^5 + a*3^7 + O(3^8)
sage: x2 = b.modular_form(a,level = 1); x2
a + (a + 2)*3 + (2*a + 1)*3^3 + (2*a + 1)*3^4 + 3^5 + (a + 2)*3^6 + a*3^7 + O(3^
↪8)
sage: x3 = b.modular_form(a,level = 2); x3
a + (a + 2)*3 + (2*a + 2)*3^2 + 2*a*3^4 + (a + 1)*3^5 + 3^6 + O(3^8)
sage: x4 = b.modular_form(a,level = 3);x4
a + (a + 2)*3 + (2*a + 2)*3^2 + (2*a + 2)*3^3 + 2*a*3^5 + a*3^6 + (a + 2)*3^7 +␣
↪O(3^8)
sage: (x4-x3).valuation()
3
```

monomial_coefficients()

Void method to comply with pickling.

EXAMPLES:

```
sage: M = BruhatTitsQuotient(3,5).harmonic_cocycles(2,prec=10)
sage: M.monomial_coefficients()
{}
```

print_values()

Print the values of the cocycle on all of the edges.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,23)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: H.basis()[0].print_values()
0   |1 + O(5^10)
1   |0
2   |0
3   |4 + 4*5 + 4*5^2 + 4*5^3 + 4*5^4 + 4*5^5 + 4*5^6 + 4*5^7 + 4*5^8 + 4*5^9 +
↪O(5^10)
4   |0
5   |0
6   |0
7   |0
8   |0
9   |0
10  |0
11  |0
```

riemann_sum(*f*, *center=1*, *level=0*, *E=None*)

Evaluate the integral of the function `f` with respect to the measure determined by `self` over $\mathbf{P}^1(\mathbf{Q}_p)$.

INPUT:

- `f` - a function on $\mathbf{P}^1(\mathbf{Q}_p)$.
- `center` - An integer (default = 1). Center of integration.
- `level` - An integer (default = 0). Determines the size of the covering when computing the Riemann sum. Runtime is exponential in the level.
- `E` - A list of edges (default = None). They should describe a covering of $\mathbf{P}^1(\mathbf{Q}_p)$.

OUTPUT:

A p-adic number.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,7)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: b = H.basis()[0]
sage: R.<z> = PolynomialRing(QQ,1)
sage: f = z^2
```

Note that f has a pole at infinity, so that the result will be meaningless:

```
sage: b.riemann_sum(f,level=0)
1 + 5 + 2*5^3 + 4*5^4 + 2*5^5 + 3*5^6 + 3*5^7 + 2*5^8 + 4*5^9 + O(5^10)
```

valuation()

Return the valuation of the cocycle, defined as the minimum of the values it takes on a set of representatives.

OUTPUT:

An integer.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,17)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: b1 = H.basis()[0]
sage: b2 = 3*b1
sage: b1.valuation()
0
sage: b2.valuation()
1
sage: H(0).valuation()
+Infinity
```

class sage.modular.btquotients.pautomorphicform.**BruhatTitsHarmonicCocycles**(*X*, *k*, *prec=None*, *basis_matrix=None*, *base_field=None*)

Bases: sage.modular.hecke.ambient_module.AmbientHeckeModule, sage.structure.unique_representation.UniqueRepresentation

Ensure unique representation

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,5)
sage: M1 = X.harmonic_cocycles( 2, prec = 10)
sage: M2 = X.harmonic_cocycles( 2, 10)
sage: M1 is M2
True
```

Element

alias of *BruhatTitsHarmonicCocycleElement*

base_extend(*base_ring*)

Extend the base ring of the coefficient module.

INPUT:

- base_ring - a ring that has a coerce map from the current base ring

OUTPUT:

A new space of HarmonicCocycles with the base extended.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,19)
sage: H = X.harmonic_cocycles(2,10)
sage: H.base_ring()
```

(continues on next page)

(continued from previous page)

```
3-adic Field with capped relative precision 10
sage: H1 = H.base_extend(Qp(3,prec=15))
sage: H1.base_ring()
3-adic Field with capped relative precision 15
```

basis_matrix()

Return a basis of `self` in matrix form.

If the coefficient module M is of finite rank then the space of Gamma invariant M valued harmonic cocycles can be represented as a subspace of the finite rank space of all functions from the finitely many edges in the corresponding BruhatTitsQuotient into M. This function computes this representation of the space of cocycles.

OUTPUT:

- A basis matrix describing the cocycles in the spaced of all M valued Gamma invariant functions on the tree.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,3)
sage: M = X.harmonic_cocycles(4,prec = 20)
sage: B = M.basis() # indirect doctest
sage: len(B) == X.dimension_harmonic_cocycles(4)
True
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu (2012-02-20)

change_ring(*new_base_ring*)

Change the base ring of the coefficient module.

INPUT:

- `new_base_ring` - a ring that has a coerce map from the current base ring

OUTPUT:

New space of HarmonicCocycles with different base ring

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5,17)
sage: H = X.harmonic_cocycles(2,10)
sage: H.base_ring()
5-adic Field with capped relative precision 10
sage: H1 = H.base_extend(Qp(5,prec=15)) # indirect doctest
sage: H1.base_ring()
5-adic Field with capped relative precision 15
```

character()

The trivial character.

OUTPUT:

The identity map.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,7)
sage: H = X.harmonic_cocycles(2,prec = 10)
sage: f = H.character()
sage: f(1)
1
sage: f(2)
2
```

embed_quaternion(*g*, *scale=1*, *exact=None*)

Embed the quaternion element `g` into the matrix algebra.

INPUT:

- `g` - A quaternion, expressed as a 4x1 matrix.

OUTPUT:

A 2x2 matrix with p-adic entries.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,2)
sage: q = X.get_stabilizers()[0][1][0]
sage: H = X.harmonic_cocycles(2,prec = 5)
sage: Hmat = H.embed_quaternion(q)
sage: Hmat.matrix().trace() == X._conv(q).reduced_trace() and Hmat.matrix().
↪determinant() == 1
True
```

free_module()

Return the underlying free module

OUTPUT:

A free module.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,7)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: H.free_module()
Vector space of dimension 1 over 3-adic Field with
capped relative precision 10
```

is_simple()

Whether `self` is irreducible.

OUTPUT:

Boolean. True if and only if `self` is irreducible.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,29)
sage: H = X.harmonic_cocycles(4,prec =10)
sage: H.rank()
14
sage: H.is_simple()
False
```

(continues on next page)

(continued from previous page)

```
sage: X = BruhatTitsQuotient(7,2)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: H.rank()
1
sage: H.is_simple()
True
```

monomial_coefficients()

Void method to comply with pickling.

EXAMPLES:

```
sage: M = BruhatTitsQuotient(3,5).harmonic_cocycles(2,prec=10)
sage: M.monomial_coefficients()
{}
```

rank()

Return the rank (dimension) of `self`.

OUTPUT:

An integer.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,11)
sage: H = X.harmonic_cocycles(2,prec = 10)
sage: X.genus() == H.rank()
True
sage: H1 = X.harmonic_cocycles(4,prec = 10)
sage: H1.rank()
16
```

submodule(*v*, *check=False*)

Return the submodule of `self` spanned by `v`.

INPUT:

- `v` - Submodule of self.free_module().
- `check` - Boolean (default = False).

OUTPUT:

Subspace of harmonic cocycles.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(3,17)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: H.rank()
3
sage: v = H.gen(0)
sage: N = H.free_module().span([v.element()])
sage: H1 = H.submodule(N)
Traceback (most recent call last):
...
NotImplementedError
```

sage.modular.btquotients.pautomorphicform.**eval_dist_at_powseries**(*phi*, *f*)

Evaluate a distribution on a powerseries.

A distribution is an element in the dual of the Tate ring. The elements of coefficient modules of overconvergent modular symbols and overconvergent p-adic automorphic forms give examples of distributions in Sage.

INPUT:

- `phi` - a distribution
- `f` - a power series over a ring coercible into a p-adic field

OUTPUT:

The value of `phi` evaluated at `f`, which will be an element in the ring of definition of `f`

EXAMPLES:

```
sage: from sage.modular.btquotients.pautomorphicform import eval_dist_at_powseries
sage: R.<X> = PowerSeriesRing(ZZ,10)
sage: f = (1 - 7*X)^(-1)

sage: D = OverconvergentDistributions(0,7,10)
sage: phi = D(list(range(1,11)))
sage: eval_dist_at_powseries(phi,f)
1 + 2*7 + 3*7^2 + 4*7^3 + 5*7^4 + 6*7^5 + 2*7^7 + 3*7^8 + 4*7^9 + O(7^10)
```

class sage.modular.btquotients.pautomorphicform.**pAdicAutomorphicFormElement**(*parent*, *vec*)

Bases: `sage.structure.element.ModuleElement`

Rudimentary implementation of a class for a p-adic automorphic form on a definite quaternion algebra over $\mathbf{Q}$. These are required in order to compute moments of measures associated to harmonic cocycles on the Bruhat-Tits tree using the overconvergent modules of Darmon-Pollack and Matt Greenberg. See Greenberg's thesis [Gr2006] for more details.

INPUT:

- `vec` - A preformatted list of data

EXAMPLES:

```
sage: X = BruhatTitsQuotient(17,3)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: h = H.an_element()
sage: HH = X.padic_automorphic_forms(2,10)
sage: a = HH(h)
sage: a
p-adic automorphic form of cohomological weight 0
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu

coleman(*t1*, *t2*, *E=None*, *method='moments'*, *mult=False*, *delta=- 1*)

If `self` is a p-adic automorphic form that corresponds to a rigid modular form, then this computes the Coleman integral of this form between two points on the boundary $P^1(\mathbf{Q}_p)$ of the p-adic upper half plane.

INPUT:

- `t1`, `t2` - elements of $P^1(\mathbf{Q}_p)$ (the endpoints of integration)

- E - (default: None). If specified, will not compute the covering adapted to t1 and t2 and instead use the given one. In that case, E should be a list of matrices corresponding to edges describing the open balls to be considered.
- method - string (default: 'moments'). Tells which algorithm to use (alternative is 'riemann_sum', which is unsuitable for computations requiring high precision)
- mult - boolean (default: False). Whether to compute the multiplicative version.

OUTPUT:

The result of the Coleman integral

EXAMPLES:

```
sage: p = 7
sage: lev = 2
sage: prec = 10
sage: X = BruhatTitsQuotient(p,lev, use_magma = True) # optional - magma
sage: k = 2 # optional - magma
sage: M = X.harmonic_cocycles(k,prec) # optional - magma
sage: B = M.basis() # optional - magma
sage: f = 3*B[0] # optional - magma
sage: MM = X.padic_automorphic_forms(k,prec,overconvergent = True) # optional -␣
↪magma
sage: D = -11 # optional - magma
sage: X.is_admissible(D) # optional - magma
True
sage: K.<a> = QuadraticField(D) # optional - magma
sage: Kp.<g> = Qq(p**2,prec) # optional - magma
sage: P = Kp.gen() # optional - magma
sage: Q = 2+Kp.gen()+ p*(Kp.gen() +1) # optional - magma
sage: F = MM.lift(f) # long time, optional - magma
sage: J0 = F.coleman(P,Q,mult = True) # long time, optional - magma
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu (2012-02-20)

derivative(*z=None*, *level=0*, *method='moments'*, *order=1*)

Return the derivative of the modular form corresponding to self.

INPUT:

- z - (default: None). If specified, evaluates the derivative at the point z in the p-adic upper half plane.
- level - integer (default: 0). If method is 'riemann_sum', will use a covering of $P^1(\mathbf{Q}_p)$ with balls of size p^-level.
- method - string (default: moments). It must be either moments or riemann_sum.
- order - integer (default: 1). The order of the derivative to be computed.

OUTPUT:

- A function from the p-adic upper half plane to $\mathbf{C}_p$. If an argument z was passed, returns instead the value of the derivative at that point.

EXAMPLES:

Integrating the Poisson kernel against a measure yields a value of the associated modular form. Such values can be computed efficiently using the overconvergent method, as long as one starts with an ordinary form:

```
sage: X = BruhatTitsQuotient(7, 2)
sage: X.genus()
1
```

Since the genus is 1, the space of weight 2 forms is 1 dimensional. Hence any nonzero form will be a U_7 eigenvector. By Jacquet-Langlands and Cerednik-Drinfeld, in this case the Hecke eigenvalues correspond to that of any nonzero form on $\Gamma_0(14)$ of weight 2. Such a form is ordinary at 7, and so we can apply the overconvergent method directly to this form without p-stabilizing:

```
sage: H = X.harmonic_cocycles(2,prec=5)
sage: h = H.gen(0)
sage: A = X.padic_automorphic_forms(2,prec=5,overconvergent=True)
sage: f0 = A.lift(h)
```

Now that we've lifted our harmonic cocycle to an overconvergent automorphic form, we extract the associated modular form as a function and test the modular property:

```
sage: T.<x> = Qq(49,prec=10)
sage: f = f0.modular_form()
sage: g = X.get_embedding_matrix()*X.get_units_of_order()[1]
sage: a,b,c,d = g.change_ring(T).list()
sage: (c*x +d)^2*f(x)-f((a*x + b)/(c*x + d))
O(7^5)
```

We can also compute the Shimura-Maass derivative, which is a nearly rigid analytic modular forms of weight 4:

```
sage: f = f0.derivative()
sage: (c*x + d)^4*f(x)-f((a*x + b)/(c*x + d))
O(7^5)
```

evaluate(*e1*)

Evaluate a p-adic automorphic form on a matrix in $GL_2(\mathbf{Q}_p)$.

INPUT:

- e1 - a matrix in $GL_2(\mathbf{Q}_p)$

OUTPUT:

- the value of self evaluated on e1

EXAMPLES:

```
sage: X = BruhatTitsQuotient(7,5)
sage: M = X.harmonic_cocycles(2,prec=5)
sage: A = X.padic_automorphic_forms(2,prec=5)
sage: a = A(M.basis()[0])
sage: a.evaluate(Matrix(ZZ,2,2,[1,2,3,1]))
4 + 6*7 + 6*7^2 + 6*7^3 + 6*7^4 + O(7^5)
sage: a.evaluate(Matrix(ZZ,2,2,[17,0,0,1]))
1 + O(7^5)
```

integrate(*f*, *center=1*, *level=0*, *method='moments'*)
Calculate

$$\int_{\mathbf{P}^1(\mathbf{Q}_p)} f(x)d\mu(x)$$

were μ is the measure associated to `self`.

INPUT:

- `f` - An analytic function.
- `center` - 2x2 matrix over $\mathbf{Q}_p$ (default: 1)
- `level` - integer (default: 0)
- `method` - string (default: 'moments'). Which method of integration to use. Either 'moments' or 'riemann_sum'.

EXAMPLES:

Integrating the Poisson kernel against a measure yields a value of the associated modular form. Such values can be computed efficiently using the overconvergent method, as long as one starts with an ordinary form:

```
sage: X = BruhatTitsQuotient(7,2)
sage: X.genus()
1
```

Since the genus is 1, the space of weight 2 forms is 1 dimensional. Hence any nonzero form will be a U_7 eigenvector. By Jacquet-Langlands and Cerednik-Drinfeld, in this case the Hecke eigenvalues correspond to that of any nonzero form on $\Gamma_0(14)$ of weight 2. Such a form is ordinary at 7, and so we can apply the overconvergent method directly to this form without p-stabilizing:

```
sage: H = X.harmonic_cocycles(2,prec = 5)
sage: h = H.gen(0)
sage: A = X.padic_automorphic_forms(2,prec = 5,overconvergent=True)
sage: a = A.lift(h)
sage: a._value[0].moment(2)
2 + 6*7 + 4*7^2 + 4*7^3 + 6*7^4 + O(7^5)
```

Now that we've lifted our harmonic cocycle to an overconvergent automorphic form we simply need to define the Teitelbaum-Poisson Kernel, and then integrate:

```
sage: Kp.<x> = Qq(49,prec = 5)
sage: z = Kp['z'].gen()
sage: f = 1/(z-x)
sage: a.integrate(f)
(5*x + 5) + (4*x + 4)*7 + (5*x + 5)*7^2 + (5*x + 6)*7^3 + O(7^5)
```

AUTHORS:

- Cameron Franc (2012-02-20)
- Marc Masdeu (2012-02-20)

modular_form(*z=None*, *level=0*, *method='moments'*)
Return the modular form corresponding to `self`.

INPUT:

- `z` - (default: None). If specified, returns the value of the form at the point `z` in the p-adic upper half plane.

- `level` - integer (default: 0). If `method` is 'riemann_sum', will use a covering of $P^1(\mathbf{Q}_p)$ with balls of size p^-level.
- `method` - string (default: `moments`). It must be either `moments` or `riemann_sum`.

OUTPUT:

- A function from the p-adic upper half plane to $\mathbf{C}_p$. If an argument `z` was passed, returns instead the value at that point.

EXAMPLES:

Integrating the Poisson kernel against a measure yields a value of the associated modular form. Such values can be computed efficiently using the overconvergent method, as long as one starts with an ordinary form:

```
sage: X = BruhatTitsQuotient(7, 2)
sage: X.genus()
1
```

Since the genus is 1, the space of weight 2 forms is 1 dimensional. Hence any nonzero form will be a U_7 eigenvector. By Jacquet-Langlands and Cerednik-Drinfeld, in this case the Hecke eigenvalues correspond to that of any nonzero form on $\Gamma_0(14)$ of weight 2. Such a form is ordinary at 7, and so we can apply the overconvergent method directly to this form without p-stabilizing:

```
sage: H = X.harmonic_cocycles(2,prec = 5)
sage: A = X.padic_automorphic_forms(2,prec = 5,overconvergent=True)
sage: f0 = A.lift(H.basis()[0])
```

Now that we've lifted our harmonic cocycle to an overconvergent automorphic form, we extract the associated modular form as a function and test the modular property:

```
sage: T.<x> = Qq(7^2,prec = 5)
sage: f = f0.modular_form(method = 'moments')
sage: a,b,c,d = X.embed_quaternion(X.get_units_of_order()[1]).change_ring(T.
↪base_ring()).list()
sage: ((c*x + d)^2*f(x)-f((a*x + b)/(c*x + d))).valuation()
5
```

valuation()

The valuation of `self`, defined as the minimum of the valuations of the values that it takes on a set of edge representatives.

OUTPUT:

An integer.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(17,3)
sage: M = X.harmonic_cocycles(2,prec=10)
sage: A = X.padic_automorphic_forms(2,prec=10)
sage: a = A(M.gen(0))
sage: a.valuation()
0
sage: (17*a).valuation()
1
```

class sage.modular.btquotients.pautomorphicform.**pAdicAutomorphicForms**(*domain*, *U*, *prec=None*, *t=None*, *R=None*, *overconvergent=False*)

Bases: sage.modules.module.Module, sage.structure.unique_representation.UniqueRepresentation

Create a space of p-automorphic forms

EXAMPLES:

```
sage: X = BruhatTitsQuotient(11,5)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: A = X.padic_automorphic_forms(2,prec=10)
sage: TestSuite(A).run()
```

Element

alias of *pAdicAutomorphicFormElement*

lift(*f*)

Lift the harmonic cocycle f to a p-automorphic form.

If one is using overconvergent coefficients, then this will compute all of the moments of the measure associated to f.

INPUT:

- f - a harmonic cocycle

OUTPUT:

A p-adic automorphic form

EXAMPLES:

If one does not work with an overconvergent form then lift does nothing:

```
sage: X = BruhatTitsQuotient(13,5)
sage: H = X.harmonic_cocycles(2,prec=10)
sage: h = H.gen(0)
sage: A = X.padic_automorphic_forms(2,prec=10)
sage: A.lift(h) # long time
p-adic automorphic form of cohomological weight 0
```

With overconvergent forms, the input is lifted naively and its moments are computed:

```
sage: X = BruhatTitsQuotient(13,11)
sage: H = X.harmonic_cocycles(2,prec=5)
sage: A2 = X.padic_automorphic_forms(2,prec=5,overconvergent=True)
sage: a = H.gen(0)
sage: A2.lift(a) # long time
p-adic automorphic form of cohomological weight 0
```

precision_cap()

Return the precision of self.

OUTPUT:

An integer.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(13,11)
sage: A = X.padic_automorphic_forms(2,prec=10)
sage: A.precision_cap()
10
```

prime()

Return the underlying prime.

OUTPUT:

- p - a prime integer

EXAMPLES:

```
sage: X = BruhatTitsQuotient(11,5)
sage: H = X.harmonic_cocycles(2,prec = 10)
sage: A = X.padic_automorphic_forms(2,prec = 10)
sage: A.prime()
11
```

zero()

Return the zero element of `self`.

EXAMPLES:

```
sage: X = BruhatTitsQuotient(5, 7)
sage: H1 = X.padic_automorphic_forms( 2, prec=10)
sage: H1.zero() == 0
True
```

CHAPTER

TWENTY

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

m

INDEX

A

B

C

D

E

H

I

N

O

P

Q

R

S

T

U

V

W

Z