Sage Reference Manual: Modular Symbols

Release 6.10

The Sage Development Team

December 21, 2015

1 Creation of modular symbols spaces 1

2 Space of modular symbols (base class) 7

3 Ambient spaces of modular symbols 29

4 Subspace of ambient spaces of modular symbols 47

5 A single element of an ambient space of modular symbols 51

6 Modular symbols {alpha, beta} 53

7 Manin symbols 57

8 Manin symbol lists 61

9 Space of boundary modular symbols 73

10 Heilbronn matrix computation 79

11 Lists of Manin symbols (elements of $\mathbb{P}^1(\mathbf{Z}/N\mathbf{Z})$) over Q 83

12 List of coset representatives for $\Gamma_1(N)$ in $\mathrm{SL}_2(\mathbf{Z})$ 91

13 List of coset representatives for $\Gamma_H(N)$ in $\mathrm{SL}_2(\mathbf{Z})$ 93

14 Relation matrices for ambient modular symbols spaces 95

15 Lists of Manin symbols (elements of $\mathbb{P}^1(R/N)$) over number fields 101

16 File: sage/modular/modsym/apply.pyx (starting at line 1) 113

17 MISSING TITLE 115

18 Optimized Cython code for computing relation matrices in certain cases. 117

19 Indices and Tables 119

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: print M.T(2).charpoly('x').factor()
(x - 3) * (x^2 + x - 1)^2
sage: print M.decomposition(2)
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for Gamma_0(23) of we
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5 for Gamma_0(23) of we
]


sage: M = ModularSymbols(23,2,base_ring=QuadraticField(5, 'sqrt5'))
sage: print M.T(2).charpoly('x').factor()
(x - 3) * (x - 1/2*sqrt5 + 1/2)^2 * (x + 1/2*sqrt5 + 1/2)^2
sage: print M.decomposition(2)
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for Gamma_0(23) of we
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for Gamma_0(23) of we
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for Gamma_0(23) of we
]
```

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
```

```
sage: G.modular_symbols().cuspidal_subspace()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 13 for Congruence Subgr
```

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
```

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 and over Rational
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 S
```

```
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 5 for Congruence S
]
```

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 Cyc
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
```

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 Rationa

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 of modulus 13 over 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 Fin
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)],
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'
```

TESTS:

We test use_cache:

```
sage: ModularSymbols_clear_cache()
sage: M = ModularSymbols(11,use_cache=False)
sage: sage.modular.modsym.modsym._cache
{}
sage: M = ModularSymbols(11,use_cache=True)
sage: sage.modular.modsym.modsym._cache
{(Congruence Subgroup Gamma0(11), 2, 0, Rational Field): <weakref at ...; to 'ModularSymbolsAmbi
sage: M is ModularSymbols(11,use_cache=True)
True
sage: M is ModularSymbols(11,use_cache=False)
False
```

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

TESTS:

Make sure #10548 is fixed sage: import gc sage: m=ModularSymbols(Gamma1(29)) sage: m=[] sage: ModularSymbols_clear_cache() sage: gc.collect() # random 3422 sage: a=[x for x in gc.get_objects() if isinstance(x,sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g1)] sage: a []

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])
<type '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

Standard initialisation function.

INPUT:

- modsym - a space of modular symbols
- A - matrix of the associated period map

EXAMPLE:

```
sage: ModularSymbols(2, 8).cuspidal_submodule().integral_period_mapping() # indirect doctest
Integral period mapping associated to Modular Symbols subspace of dimension 2 of Modular Symbols
```

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

TESTS:

Verify that trac ticket #12772 is fixed:

```
sage: M = ModularSymbols(1,12,sign=1).cuspidal_subspace().new_subspace()
sage: A = M.decomposition()[0]
sage: A.compact_system_of_eigenvalues(prime_range(10))
(
[   -24]
[   252]
[  4830]
[-16744], (1)
)
```

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
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(
```

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
]
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
Traceback (most recent call last):
...
NotImplementedError: computation of dual star involution matrix not yet implemented for this
sage: ModularSymbols(Gamma0(11),2).dual_star_involution_matrix()
[ 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()

Returns 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)`.

EXAMPLE:

```
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 F
```

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()
((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)]
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/
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
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)
```

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
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'th Hecke operator acting on the integral structure on self (as returned by "self.integral_structu
This is often (but not always) different from the matrix returned by `self.hecke_matrix`, even if the latter has integral entries.

EXAMPLE:

```
sage: M = ModularSymbols(6,4)
sage: M.hecke_matrix(3)
[27  0  0  0  6 -6]
[ 0  1 -4  4  8 10]
[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 Sym
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()
Traceback (most recent call last):
...
ValueError: integral mapping only defined for cuspidal spaces
```

integral_structure()

Return the **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
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
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 not 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(
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 6 for Gamma_0(
]
sage: C=ModularSymbols(1,14,0,GF(5)).cuspidal_submodule()
sage: C
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0(
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
sage: ModularSymbols(14,4).minus_submodule()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 12 for Gamma_0
```

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

Returns 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(
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

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(
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(

sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 7 for Gamma_0(

sage: S = ModularSymbols(389,sign=1)[3]; S
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 33 for Gamma_0
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 32 for Gamma_0
sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 6 of Modular Symbols space of dimension 65 for Gamma_0

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(
sage: S.modular_symbols_of_sign(1) is S
True
sage: S.modular_symbols_of_sign(0)
Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension 12 for Gamma_0
sage: S.modular_symbols_of_sign(-1)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5 for Gamma_0(
```

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()

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
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 F
sage: ModularSymbols(17,2).plus_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(
```

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

Returns the q-expansion to precision prec of a new eigenform associated to self, where 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 of modulus 13 over Number Field in a with defining polynomial
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
sage: f
Modular Symbols subspace of dimension 8 of Modular Symbols space of dimension 10 and level 7
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'*)

Returns 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)
[
```

```
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(
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(
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*)

Returns 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)
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, 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:
[ 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
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 +
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(
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0(
]
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
sage: D = M.decomposition(); D
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 and level 25
]
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                                   0
```

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(
sage: r = A.rational_period_mapping(); r
Rational period mapping associated to Modular Symbols subspace of dimension 2 of Modular Sym
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 F
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()
Returns 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(
sage: M.sign_submodule(-1)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for Gamma_0(
sage: M.sign_submodule(-1).sign()
-1
```

simple_factors()

Returns 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
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
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0(
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0(
```

star_decomposition()

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

EXAMPLE:

```
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
Modular Symbols subspace of dimension 7 of Modular Symbols space of dimension 31 for Gamma_1
]
```

star_eigenvalues()

Returns 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_involuti
Traceback (most recent call last):
...
NotImplementedError
```

sturm_bound()

Returns 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.

EXAMPLE:

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).

EXAMPLE:
```
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 F
```

matrix()
Return the matrix of this period mapping.

EXAMPLE:
```
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 F
```

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

Standard initialisation function.

INPUT:

- modsym - a space of modular symbols
- A - matrix of the associated period map

EXAMPLE:
```
sage: ModularSymbols(2, 8).cuspidal_submodule().integral_period_mapping() # indirect doctest
Integral period mapping associated to Modular Symbols subspace of dimension 2 of Modular Symbols
```

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 and over Ra
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 and over Finit
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, ove
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, ove
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
```

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)
)]
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*ze
```

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 Fie
sage: M.cuspidal_submodule()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 5 for Gamma_0(
sage: ModularSymbols(1,24,-1).cuspidal_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 2 for Gamma_0(
```

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(
```

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
sage: M.T(3)(M.0)
28*[X^2,(0,1)] + 2*[X^2,(1,7)] - [X^2,(1,9)] - [X^2,(1,10)]
sage: M.T(3)(M.0).element()
(28, 2, -1, -1)
```

factor()

Returns 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
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0
(Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 7 for Gamma_0

sage: ModularSymbols(1,6,0,GF(2)).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0

sage: ModularSymbols(18,2).factorization()
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 for Gamma_0
(Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension 7 for Gamma_0

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
sage: M.factorization()                    # long time (about 8 seconds)
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
```

factorization()

Returns 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
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0
(Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 7 for Gamma_0

sage: ModularSymbols(1,6,0,GF(2)).factorization()
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0

sage: ModularSymbols(18,2).factorization()
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 for Gamma_0
(Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension 7 for Gamma_0

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
```

```
sage: M.factorization()                    # long time (about 8 seconds)
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
(Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 7 and level 3
```

integral_structure(*algorithm='default'*)

Return the **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]
[  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  1/102      0  5/204  1/136  23/24   3/17 43/136 69/136]
[     0      0      0   1/48      0   1/48  23/24    1/6    1/8  17/24]
[     0      0      0      0   1/24      0  23/24    1/3    1/6    1/2]
[     0      0      0      0      0   1/24  23/24    1/3  11/24   5/24]
[     0      0      0      0      0      0      1      0      0      0]
[     0      0      0      0      0      0      0    1/2      0    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()
Returns 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()
Returns 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 $0 \leq i \leq k-2$ 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 monomial Manin Symbol associated to $[i; (u, v)]$, with $i = 0$ if not supplied, corresponding to the symbol $[X^i * Y^{k-2-i}, (u, v)]$.

EXAMPLES:

```
sage: M = ModularSymbols(11,4,1)
sage: M.manin_symbol([2,5,6])
[X^2,(1,10)]
```

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 $0 \leq i \leq k-2$ 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}
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

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_sign(*sign*)

Returns 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 F
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
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 and over Ratio
```

modular_symbols_of_weight(*k*)

Returns 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 and over Rationa
```

new_submodule(*p=None*)

Returns 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
```

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

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()

Returns 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
[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]
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(
```

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 and over R
```

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 Rationa
sage: ModularSymbols(eps, 3)
Modular Symbols space of dimension 2 and level 4, weight 3, character [-1], sign 0, over Rationa
```

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
```

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*)

Returns 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
sage: M.modular_symbols_of_level(15)
Modular Symbols space of dimension 0 and level 15, weight 2, character [1, zeta4], sign 0, o
```

modular_symbols_of_sign(*sign*)

Returns 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
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
sage: M.modular_symbols_of_sign(-1)
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4], sign -1, over
```

modular_symbols_of_weight(*k*)

Returns 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
Modular Symbols space of dimension 0 and level 5, weight 2, character [zeta4], sign 0, over
sage: M.modular_symbols_of_weight(3)
Modular Symbols space of dimension 2 and level 5, weight 3, character [zeta4], sign 0, over
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 and over R
```

manin_symbols()

Return the Manin symbol list of this modular symbol space.

EXAMPLES:

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

modular_symbols_of_level(N)

Returns 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.

For example, if self is the space of modular symbols of weight 2 for $\Gamma_0(22)$, and level is 11, then this function returns the modular symbol space of weight 2 for $\Gamma_0(11)$.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: M.modular_symbols_of_level(22)
Modular Symbols space of dimension 7 for Gamma_0(22) of weight 2 with sign 0 over Rational F
sage: M = ModularSymbols(Gamma1(6))
sage: M.modular_symbols_of_level(12)
Modular Symbols space of dimension 9 for Gamma_1(12) of weight 2 with sign 0 and over Ration
```

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 and over Rational
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 and over Rational Fi
```

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 and over R
```

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
```

modular_symbols_of_level(N)

Returns 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.

For example, if self is the space of modular symbols of weight 2 for $\Gamma_0(22)$, and level is 11, then this function returns the modular symbol space of weight 2 for $\Gamma_0(11)$.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(30),4); M
Modular Symbols space of dimension 144 for Gamma_1(30) of weight 4 with sign 0 and over Rati
sage: M.modular_symbols_of_level(22)
Modular Symbols space of dimension 90 for Gamma_1(22) of weight 4 with sign 0 and over Ratio
```

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]
```

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] o
```

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
```

modular_symbols_of_level(N)

Returns a space of modular symbols with the same parameters as this space except with level N, which should be either a divisor or a multiple of the level of self.

TESTS:

```
sage: M = ModularSymbols(GammaH(15,[7]),6)
sage: M.modular_symbols_of_level(5)
Modular Symbols space of dimension 4 for Gamma_0(5) of weight 6 with sign 0 over Rational Fi
sage: M.modular_symbols_of_level(30)
Traceback (most recent call last):
...
ValueError: N (=30) should be a factor of the level of this space (=15)
sage: M.modular_symbols_of_level(73)
Traceback (most recent call last):
...
ValueError: N (=73) should be a factor of the level of this space (=15)
```

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 F
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_
sage: S.is_cuspidal()
True
sage: S.cuspidal_submodule()
Modular Symbols subspace of dimension 40 of Modular Symbols space of dimension 48 for Gamma_
```

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
```

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
sage: ModularSymbols(20,2).cuspidal_subspace().eisenstein_subspace()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 7 for Gamma_0(
```

factorization()

Returns 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
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0
(Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0
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

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(
```

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)
[ 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.

TESTS:

```
sage: x = ModularSymbols(3, 12).cuspidal_submodule().gen(0)
sage: x == loads(dumps(x))
True
```

list()

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

EXAMPLE:

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

manin_symbol_rep()

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

EXAMPLE:

```
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()

Returns a representation of self as a formal sum of modular symbols.

EXAMPLE:

```
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

EXAMPLE:

```
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} - 248832*X^6*{-1/12, 0} + 893101*X^5*Y*{-1/11, 0} - 103680*X^5*Y*{-1/12,
sage: set_modsym_print_mode('vector'); x
(1, 1, 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)

TESTS:

```
sage: s = ModularSymbols(11).2.modular_symbol_rep()[0][1]; s
{-1/9, 0}
sage: loads(dumps(s)) == s
True
```

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()

Returns 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()
-[Y^2,(1,1)] - 2*[X*Y,(-1,0)] - [X^2,(-6,1)] - [X^2,(-1,0)]
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 $X^i Y^{k-2-i}\{\alpha, \beta\}$, return $X^i Y^{k-2-i}$.

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 $X^i Y^{k-2-i}\{\alpha, \beta\}$, 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 $P(X,Y) \in \mathbb{Z}[X,Y]$ is homogeneous of weight $k-2$ and $(u:v) \in \mathbb{P}^1(\mathbb{Z}/N\mathbb{Z})$. 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.

EXAMPLE:

```
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 $SL_2(\mathbb{Z})$.

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 $c = c'$ (mod N) and $d = d'$ (mod 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}
```

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 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):
...
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 xrange(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)
<type '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

EXAMPLE:

```
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 of the j'th symbol.

EXAMPLE:

```
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 xrange(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.

EXAMPLE:

```
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 xrange(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.

EXAMPLE:

```
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 xrange(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.

EXAMPLE:

```
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 xrange(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.

EXAMPLE:

```
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.

EXAMPLE:

```
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.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.

EXAMPLE:

```
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.

EXAMPLE:

```
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.

EXAMPLE:

```
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))
<class 'sage.modular.modsym.manin_symbol_list.ManinSymbolList_group_with_category'>
```

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 $[P(aX + bY, cX + dY), (u, v)g]$.

EXAMPLE:

```
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 of the j'th symbol (a Python `int`, either 1 or -1)

EXAMPLE:

```
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 xrange(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 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)
(40, 1)
sage: [m.apply_S(i) for i in xrange(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()

EXAMPLE:

```
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 xrange(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()

EXAMPLE:

```
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 xrange(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

EXAMPLE:

```
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),
 (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. 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 2 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)
Traceback (most recent call last):
...
ValueError: only 0 generators known for Space of Boundary Modular Symbols for Congruence Sub
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
```

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 Rationa
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 we
sage: B == loads(dumps(B))
True
```

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, 5), 1), ((1, 6), 1), ((1, 1), 1), ((1, 4), 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_t
sage: A
[ 1  0  0]
[ 3  0 -1]
[ 4 -2 -1]
[ 7 -2 -2]
[ 6  0 -2]
[12 -2 -4]
sage: z = M((1,0))
sage: [M.T(n)(z).element() for n in [1..6]]
[(1, 0, 0), (3, 0, -1), (4, -2, -1), (7, -2, -2), (6, 0, -2), (12, -2, -4)]
```

TESTS:

```
sage: M = ModularSymbols(389,2,1,GF(7))
sage: C = M.cuspidal_subspace()
sage: N = C.new_subspace()
sage: D = N.decomposition()
sage: D[1].q_eigenform(10, 'a') # indirect doctest
q + 4*q^2 + 2*q^3 + 6*q^5 + q^6 + 5*q^7 + 6*q^8 + q^9 + O(q^10)
```

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: sage.modular.modsym.heilbronn.hecke_images_gamma0_weight_k(4,1,3,15,6,[1,11,12], R)
[       0        0      1/8     -1/8        0        0        0        0]
[-4435/22 -1483/22     -112 -4459/22  2151/22 -5140/11  4955/22  2340/11]
[ 1253/22  1981/22       -2  3177/22 -1867/22  6560/11 -7549/22  -612/11]
sage: x = M((3,4,1)) ; x.element()
(0, 0, 1/8, -1/8, 0, 0, 0, 0)
sage: M.T(11)(x).element()
(-4435/22, -1483/22, -112, -4459/22, 2151/22, -5140/11, 4955/22, 2340/11)
sage: M.T(12)(x).element()
(1253/22, 1981/22, -2, 3177/22, -1867/22, 6560/11, -7549/22, -612/11)
```

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)]
```

Saving and loading works.

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

N()

Returns 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)
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 xrange(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 xrange(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 xrange(len(L))])
True
```

index(*u*, *v*)

Returns 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*)

Returns 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()

Returns the underlying list of this P1List object.

EXAMPLES:

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

normalize(u, v)

Returns 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)

Returns 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)
[1L, 1L, 2L, 3L] # 32-bit
[1, 1, 2, 3]     # 64-bit
```

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))
sage: 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)
[1L, 8L, 2L, 17L] # 32-bit
[1, 8, 2, 17]     # 64-bit
sage: m=Matrix(Integers(),2,2,lift_to_sl2z_llong(2,6,11))
sage: 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
sage: (7*77154) % 90000
78
```

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

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

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

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.

TESTS:

```
sage: L = sage.modular.modsym.g1list.G1list(18)
sage: loads(dumps(L)) == L
True
```

list()

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

EXAMPLE:

```
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,
```

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.

EXAMPLE:

```
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*)

A class representing a list of coset representatives for $\Gamma_H(N)$ in $\mathrm{SL}_2(\mathbf{Z})$.

TESTS:

```
sage: L = sage.modular.modsym.ghlist.GHlist(GammaH(18,[13]))
sage: loads(dumps(L)) == L
True
```

list()

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

EXAMPLE:

```
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,
```

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.

EXAMPLE:

```
sage: from sage.modular.modsym.relation_matrix import *
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.

EXAMPLE:

```
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), (
)
```

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.

EXAMPLE:

```
sage: from sage.modular.modsym.relation_matrix import *
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)

EXAMPLE:

```
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)),
 ((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)),
 ((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

EXAMPLE:

```
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
)
sage: A[0].is_sparse()
True
```

sage.modular.modsym.relation_matrix.**sparse_2term_quotient**(*rels*, *n*, *F*)

Performs 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` - set 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.

EXAMPLE: 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: v = [((int(0),3), (int(1),-1)), ((int(1),1), (int(3),1)), ((int(2),1),(int(3),1)), ((int(4
sage: rels = set(v)
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()
```

```
[1, -4*a^2 + 9*a - 21, 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))
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()

Returns 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

Returns 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

Returns 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)
sage: alpha = MSymbol(N, 3*a + 1, a)
sage: alpha.lift_to_sl2_Ok()
[0, -1, 1, a]
```

normalize(*with_scalar=False*)

Returns 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()

Returns 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()

Returns 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*)

Applies 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
[-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*)

Returns 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*)

Returns 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 $c = c'$ (mod N) and $d = d'$ (mod 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)
[1, -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()

Returns 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())
<type 'list'>
```

normalize(*c*, *d=None*, *with_scalar=False*)

Returns 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)
sage: Ok = k.ring_of_integers(k)
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(k)
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(k)
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*)

Returns (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*)

Returns 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

FILE: SAGE/MODULAR/MODSYM/APPLY.PYX (STARTING AT LINE 1)

class sage.modular.modsym.apply.**Apply**
Bases: object

sage.modular.modsym.apply.**apply_to_monomial**(*i*, *j*, *a*, *b*, *c*, *d*)
Returns 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.

EXAMPLE:

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

MISSING TITLE

class sage.modular.modsym.hecke_operator.**HeckeOperator**(*parent*, *n*)

Bases: sage.modular.hecke.hecke_operator.HeckeOperator

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: H = M.hecke_operator(2005); H
Hecke operator T_2005 on Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with s
sage: H == loads(dumps(H))
True
```

We create a Hecke operator of large index (greater than 32 bits):

```
sage: M1 =  ModularSymbols(21,2)
sage: M1.hecke_operator(13^9)
Hecke operator T_10604499373 on Modular Symbols space of dimension 5 for Gamma_0(21) of weight 2
```

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) 64*[X^2,(1,8)] + 24*[X^2,(1,10)] - 9*[X^2,(1,13)] + 37*[X^2,(1,16)] 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*)

Performs 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` - set 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.

EXAMPLES:

```
sage: v = [((0,1), (1,-1)), ((1,1), (3,1)), ((2,1),(3,1)), ((4,1),(5,-1))]
sage: rels = set(v)
sage: n = 6
sage: from sage.modular.modsym.relation_matrix_pyx import sparse_2term_quotient_only_pm1
sage: sparse_2term_quotient_only_pm1(rels, n)
[(3, -1), (3, -1), (3, -1), (3, 1), (5, 1), (5, 1)]
```

CHAPTER

NINETEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

m

sage.modular.modsym.ambient, 29
sage.modular.modsym.apply, 113
sage.modular.modsym.boundary, 73
sage.modular.modsym.element, 51
sage.modular.modsym.g1list, 91
sage.modular.modsym.ghlist, 93
sage.modular.modsym.hecke_operator, 115
sage.modular.modsym.heilbronn, 79
sage.modular.modsym.manin_symbol, 57
sage.modular.modsym.manin_symbol_list, 61
sage.modular.modsym.modsym, 1
sage.modular.modsym.modular_symbols, 53
sage.modular.modsym.p1list, 83
sage.modular.modsym.p1list_nf, 101
sage.modular.modsym.relation_matrix, 95
sage.modular.modsym.relation_matrix_pyx, 117
sage.modular.modsym.space, 7
sage.modular.modsym.subspace, 47

A

abelian_variety() (sage.modular.modsym.space.ModularSymbolsSpace method), 7
abvarquo_cuspidal_subgroup() (sage.modular.modsym.space.ModularSymbolsSpace method), 8
abvarquo_rational_cuspidal_subgroup() (sage.modular.modsym.space.ModularSymbolsSpace method), 8
alpha() (sage.modular.modsym.modular_symbols.ModularSymbol method), 53
Apply (class in sage.modular.modsym.apply), 113
apply() (sage.modular.modsym.heilbronn.Heilbronn method), 79
apply() (sage.modular.modsym.manin_symbol.ManinSymbol method), 57
apply() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 61
apply() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 65
apply() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 70
apply() (sage.modular.modsym.modular_symbols.ModularSymbol method), 53
apply_I() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 61
apply_I() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 65
apply_I() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 70
apply_I() (sage.modular.modsym.p1list.P1List method), 83
apply_J_epsilon() (sage.modular.modsym.p1list_nf.P1NFList method), 105
apply_S() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 61
apply_S() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 65
apply_S() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 70
apply_S() (sage.modular.modsym.p1list.P1List method), 84
apply_S() (sage.modular.modsym.p1list_nf.P1NFList method), 105
apply_sparse() (sage.modular.modsym.hecke_operator.HeckeOperator method), 115
apply_T() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 62
apply_T() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 66
apply_T() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 71
apply_T() (sage.modular.modsym.p1list.P1List method), 84
apply_T_alpha() (sage.modular.modsym.p1list_nf.P1NFList method), 106
apply_to_monomial() (in module sage.modular.modsym.apply), 113
apply_TS() (sage.modular.modsym.p1list_nf.P1NFList method), 106
apply_TT() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 62
apply_TT() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 66
apply_TT() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 71

B

beta() (sage.modular.modsym.modular_symbols.ModularSymbol method), 54
boundary_map() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 30

boundary_map() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 47
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 30
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wt2_g0 method), 41
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_eps method), 42
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g0 method), 44
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g1 method), 45
boundary_space() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_gamma_h method), 46
BoundarySpace (class in sage.modular.modsym.boundary), 74
BoundarySpace_wtk_eps (class in sage.modular.modsym.boundary), 76
BoundarySpace_wtk_g0 (class in sage.modular.modsym.boundary), 76
BoundarySpace_wtk_g1 (class in sage.modular.modsym.boundary), 77
BoundarySpace_wtk_gamma_h (class in sage.modular.modsym.boundary), 77
BoundarySpaceElement (class in sage.modular.modsym.boundary), 75

C

c (sage.modular.modsym.p1list_nf.MSymbol attribute), 103
canonical_parameters() (in module sage.modular.modsym.modsym), 4
change_ring() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 30
character() (sage.modular.modsym.boundary.BoundarySpace method), 74
character() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 66
character() (sage.modular.modsym.space.ModularSymbolsSpace method), 9
codomain() (sage.modular.modsym.space.PeriodMapping method), 26
compact_newform_eigenvalues() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 31
compact_system_of_eigenvalues() (sage.modular.modsym.space.ModularSymbolsSpace method), 9
compute_presentation() (in module sage.modular.modsym.relation_matrix), 95
compute_presentation() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 31
congruence_number() (sage.modular.modsym.space.ModularSymbolsSpace method), 10
coordinate_vector() (sage.modular.modsym.boundary.BoundarySpaceElement method), 76
cuspidal_submodule() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 32
cuspidal_submodule() (sage.modular.modsym.space.ModularSymbolsSpace method), 10
cuspidal_submodule() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 47
cuspidal_subspace() (sage.modular.modsym.space.ModularSymbolsSpace method), 10
cusps() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 32

D

d (sage.modular.modsym.p1list_nf.MSymbol attribute), 103
default_prec() (sage.modular.modsym.space.ModularSymbolsSpace method), 10
dimension_of_associated_cuspform_space() (sage.modular.modsym.space.ModularSymbolsSpace method), 11
domain() (sage.modular.modsym.space.PeriodMapping method), 26
dual_star_involution_matrix() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 32
dual_star_involution_matrix() (sage.modular.modsym.space.ModularSymbolsSpace method), 11
dual_star_involution_matrix() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 48

E

eisenstein_submodule() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 32
eisenstein_subspace() (sage.modular.modsym.space.ModularSymbolsSpace method), 11
eisenstein_subspace() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 48
Element (sage.modular.modsym.manin_symbol_list.ManinSymbolList attribute), 61
Element (sage.modular.modsym.space.ModularSymbolsSpace attribute), 7
element() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 32

endpoints() (sage.modular.modsym.manin_symbol.ManinSymbol method), 58
export (class in sage.modular.modsym.p1list), 86

F

factor() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 33
factorization() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 33
factorization() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 48
free_module() (sage.modular.modsym.boundary.BoundarySpace method), 74

G

G1list (class in sage.modular.modsym.g1list), 91
gen() (sage.modular.modsym.boundary.BoundarySpace method), 74
gens_to_basis_matrix() (in module sage.modular.modsym.relation_matrix), 96
GHlist (class in sage.modular.modsym.ghlist), 93
group() (sage.modular.modsym.boundary.BoundarySpace method), 74
group() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_gamma_h method), 69
group() (sage.modular.modsym.space.ModularSymbolsSpace method), 11

H

hecke_images_gamma0_weight2() (in module sage.modular.modsym.heilbronn), 80
hecke_images_gamma0_weight_k() (in module sage.modular.modsym.heilbronn), 80
hecke_images_nonquad_character_weight2() (in module sage.modular.modsym.heilbronn), 81
hecke_images_quad_character_weight2() (in module sage.modular.modsym.heilbronn), 82
hecke_module_of_level() (sage.modular.modsym.space.ModularSymbolsSpace method), 12
HeckeOperator (class in sage.modular.modsym.hecke_operator), 115
Heilbronn (class in sage.modular.modsym.heilbronn), 79
HeilbronnCremona (class in sage.modular.modsym.heilbronn), 79
HeilbronnMerel (class in sage.modular.modsym.heilbronn), 80

I

i (sage.modular.modsym.manin_symbol.ManinSymbol attribute), 58
i() (sage.modular.modsym.modular_symbols.ModularSymbol method), 54
index() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 62
index() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 67
index() (sage.modular.modsym.p1list.P1List method), 84
index() (sage.modular.modsym.p1list_nf.P1NFList method), 107
index_of_normalized_pair() (sage.modular.modsym.p1list.P1List method), 85
index_of_normalized_pair() (sage.modular.modsym.p1list_nf.P1NFList method), 107
integral_basis() (sage.modular.modsym.space.ModularSymbolsSpace method), 12
integral_hecke_matrix() (sage.modular.modsym.space.ModularSymbolsSpace method), 13
integral_period_mapping() (sage.modular.modsym.space.ModularSymbolsSpace method), 13
integral_structure() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 34
integral_structure() (sage.modular.modsym.space.ModularSymbolsSpace method), 14
IntegralPeriodMapping (class in sage.modular.modsym.space), 7
intersection_number() (sage.modular.modsym.space.ModularSymbolsSpace method), 14
is_ambient() (sage.modular.modsym.boundary.BoundarySpace method), 75
is_ambient() (sage.modular.modsym.space.ModularSymbolsSpace method), 14
is_cuspidal() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 34
is_cuspidal() (sage.modular.modsym.space.ModularSymbolsSpace method), 15
is_cuspidal() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 49

is_eisenstein() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 35
is_eisenstein() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 49
is_ManinSymbol() (in module sage.modular.modsym.manin_symbol), 59
is_ModularSymbolsElement() (in module sage.modular.modsym.element), 51
is_ModularSymbolsSpace() (in module sage.modular.modsym.space), 27
is_simple() (sage.modular.modsym.space.ModularSymbolsSpace method), 15

L

level() (sage.modular.modsym.manin_symbol.ManinSymbol method), 58
level() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 67
level() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 72
lift_to_sl2_Ok() (in module sage.modular.modsym.p1list_nf), 109
lift_to_sl2_Ok() (sage.modular.modsym.p1list_nf.MSymbol method), 103
lift_to_sl2_Ok() (sage.modular.modsym.p1list_nf.P1NFList method), 108
lift_to_sl2z() (in module sage.modular.modsym.p1list), 86
lift_to_sl2z() (sage.modular.modsym.manin_symbol.ManinSymbol method), 58
lift_to_sl2z() (sage.modular.modsym.p1list.P1List method), 85
lift_to_sl2z_int() (in module sage.modular.modsym.p1list), 87
lift_to_sl2z_llong() (in module sage.modular.modsym.p1list), 87
list() (sage.modular.modsym.element.ModularSymbolsElement method), 51
list() (sage.modular.modsym.g1list.G1list method), 91
list() (sage.modular.modsym.ghlist.GHlist method), 93
list() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 62
list() (sage.modular.modsym.p1list.P1List method), 85
list() (sage.modular.modsym.p1list_nf.P1NFList method), 108

M

make_coprime() (in module sage.modular.modsym.p1list_nf), 110
manin_basis() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 35
manin_generators() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 35
manin_gens_to_basis() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 35
manin_symbol() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 36
manin_symbol() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 63
manin_symbol_list() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 63
manin_symbol_rep() (sage.modular.modsym.element.ModularSymbolsElement method), 51
manin_symbol_rep() (sage.modular.modsym.modular_symbols.ModularSymbol method), 54
manin_symbols() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 36
manin_symbols() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_eps method), 42
manin_symbols() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g0 method), 44
manin_symbols() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g1 method), 45
manin_symbols() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_gamma_h method), 46
manin_symbols_basis() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 36
ManinSymbol (class in sage.modular.modsym.manin_symbol), 57
ManinSymbolList (class in sage.modular.modsym.manin_symbol_list), 61
ManinSymbolList_character (class in sage.modular.modsym.manin_symbol_list), 64
ManinSymbolList_gamma0 (class in sage.modular.modsym.manin_symbol_list), 68
ManinSymbolList_gamma1 (class in sage.modular.modsym.manin_symbol_list), 68
ManinSymbolList_gamma_h (class in sage.modular.modsym.manin_symbol_list), 69
ManinSymbolList_group (class in sage.modular.modsym.manin_symbol_list), 69
matrix() (sage.modular.modsym.space.PeriodMapping method), 27

minus_submodule() (sage.modular.modsym.space.ModularSymbolsSpace method), 15
modI_relations() (in module sage.modular.modsym.relation_matrix), 97
modS_relations() (in module sage.modular.modsym.relation_matrix), 97
modular_symbol() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 36
modular_symbol_rep() (sage.modular.modsym.element.ModularSymbolsElement method), 51
modular_symbol_rep() (sage.modular.modsym.manin_symbol.ManinSymbol method), 58
modular_symbol_sum() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 37
modular_symbols_of_level() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_eps method), 42
modular_symbols_of_level() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g0 method), 44
modular_symbols_of_level() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_g1 method), 45
modular_symbols_of_level() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_gamma_h method), 46
modular_symbols_of_sign() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 38
modular_symbols_of_sign() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_eps method), 42
modular_symbols_of_sign() (sage.modular.modsym.space.ModularSymbolsSpace method), 15
modular_symbols_of_weight() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 38
modular_symbols_of_weight() (sage.modular.modsym.ambient.ModularSymbolsAmbient_wtk_eps method), 43
modular_symbols_space() (sage.modular.modsym.space.PeriodMapping method), 27
ModularSymbol (class in sage.modular.modsym.modular_symbols), 53
ModularSymbols() (in module sage.modular.modsym.modsym), 2
ModularSymbols_clear_cache() (in module sage.modular.modsym.modsym), 4
ModularSymbolsAmbient (class in sage.modular.modsym.ambient), 29
ModularSymbolsAmbient_wt2_g0 (class in sage.modular.modsym.ambient), 40
ModularSymbolsAmbient_wtk_eps (class in sage.modular.modsym.ambient), 41
ModularSymbolsAmbient_wtk_g0 (class in sage.modular.modsym.ambient), 43
ModularSymbolsAmbient_wtk_g1 (class in sage.modular.modsym.ambient), 44
ModularSymbolsAmbient_wtk_gamma_h (class in sage.modular.modsym.ambient), 45
ModularSymbolsElement (class in sage.modular.modsym.element), 51
ModularSymbolsSpace (class in sage.modular.modsym.space), 7
ModularSymbolsSubspace (class in sage.modular.modsym.subspace), 47
MSymbol (class in sage.modular.modsym.p1list_nf), 102
multiplicity() (sage.modular.modsym.space.ModularSymbolsSpace method), 16

N

n (sage.modular.modsym.heilbronn.HeilbronnMerel attribute), 80
N() (sage.modular.modsym.p1list.P1List method), 83
N() (sage.modular.modsym.p1list_nf.MSymbol method), 103
N() (sage.modular.modsym.p1list_nf.P1NFList method), 105
new_submodule() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 38
new_subspace() (sage.modular.modsym.space.ModularSymbolsSpace method), 16
ngens() (sage.modular.modsym.space.ModularSymbolsSpace method), 17
normalize() (sage.modular.modsym.g1list.G1list method), 91
normalize() (sage.modular.modsym.ghlist.GHlist method), 93
normalize() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 64
normalize() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_character method), 67
normalize() (sage.modular.modsym.manin_symbol_list.ManinSymbolList_group method), 72
normalize() (sage.modular.modsym.p1list.P1List method), 85
normalize() (sage.modular.modsym.p1list_nf.MSymbol method), 103
normalize() (sage.modular.modsym.p1list_nf.P1NFList method), 109
normalize_with_scalar() (sage.modular.modsym.p1list.P1List method), 86

O

old_subspace() (sage.modular.modsym.space.ModularSymbolsSpace method), 17

P

p (sage.modular.modsym.heilbronn.HeilbronnCremona attribute), 80
p1_normalize() (in module sage.modular.modsym.p1list), 87
p1_normalize_int() (in module sage.modular.modsym.p1list), 88
p1_normalize_llong() (in module sage.modular.modsym.p1list), 89
P1List (class in sage.modular.modsym.p1list), 83
p1list() (in module sage.modular.modsym.p1list), 89
p1list() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 39
p1list_int() (in module sage.modular.modsym.p1list), 89
p1list_llong() (in module sage.modular.modsym.p1list), 90
P1NFList (class in sage.modular.modsym.p1list_nf), 104
p1NFlist() (in module sage.modular.modsym.p1list_nf), 111
P1NFList_clear_level_cache() (in module sage.modular.modsym.p1list_nf), 109
PeriodMapping (class in sage.modular.modsym.space), 26
plus_submodule() (sage.modular.modsym.space.ModularSymbolsSpace method), 17
polynomial_part() (sage.modular.modsym.modular_symbols.ModularSymbol method), 55
psi() (in module sage.modular.modsym.p1list_nf), 111

Q

q_eigenform() (sage.modular.modsym.space.ModularSymbolsSpace method), 17
q_eigenform_character() (sage.modular.modsym.space.ModularSymbolsSpace method), 18
q_expansion_basis() (sage.modular.modsym.space.ModularSymbolsSpace method), 18
q_expansion_cuspforms() (sage.modular.modsym.space.ModularSymbolsSpace method), 20
q_expansion_module() (sage.modular.modsym.space.ModularSymbolsSpace method), 20

R

rank() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 39
rank() (sage.modular.modsym.boundary.BoundarySpace method), 75
rational_period_mapping() (sage.modular.modsym.space.ModularSymbolsSpace method), 23
RationalPeriodMapping (class in sage.modular.modsym.space), 27
relation_matrix_wtk_g0() (in module sage.modular.modsym.relation_matrix), 98

S

sage.modular.modsym.ambient (module), 29
sage.modular.modsym.apply (module), 113
sage.modular.modsym.boundary (module), 73
sage.modular.modsym.element (module), 51
sage.modular.modsym.g1list (module), 91
sage.modular.modsym.ghlist (module), 93
sage.modular.modsym.hecke_operator (module), 115
sage.modular.modsym.heilbronn (module), 79
sage.modular.modsym.manin_symbol (module), 57
sage.modular.modsym.manin_symbol_list (module), 61
sage.modular.modsym.modsym (module), 1
sage.modular.modsym.modular_symbols (module), 53
sage.modular.modsym.p1list (module), 83

sage.modular.modsym.p1list_nf (module), 101
sage.modular.modsym.relation_matrix (module), 95
sage.modular.modsym.relation_matrix_pyx (module), 117
sage.modular.modsym.space (module), 7
sage.modular.modsym.subspace (module), 47
set_default_prec() (sage.modular.modsym.space.ModularSymbolsSpace method), 23
set_modsym_print_mode() (in module sage.modular.modsym.element), 52
set_precision() (sage.modular.modsym.space.ModularSymbolsSpace method), 24
sign() (sage.modular.modsym.boundary.BoundarySpace method), 75
sign() (sage.modular.modsym.space.ModularSymbolsSpace method), 24
sign_submodule() (sage.modular.modsym.space.ModularSymbolsSpace method), 25
simple_factors() (sage.modular.modsym.space.ModularSymbolsSpace method), 25
space() (sage.modular.modsym.modular_symbols.ModularSymbol method), 55
sparse_2term_quotient() (in module sage.modular.modsym.relation_matrix), 99
sparse_2term_quotient_only_pm1() (in module sage.modular.modsym.relation_matrix_pyx), 117
star_decomposition() (sage.modular.modsym.space.ModularSymbolsSpace method), 25
star_eigenvalues() (sage.modular.modsym.space.ModularSymbolsSpace method), 25
star_involution() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 39
star_involution() (sage.modular.modsym.space.ModularSymbolsSpace method), 26
star_involution() (sage.modular.modsym.subspace.ModularSymbolsSubspace method), 49
sturm_bound() (sage.modular.modsym.space.ModularSymbolsSpace method), 26
submodule() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 39
symbol_list() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 64

T

T_relation_matrix_wtk_g0() (in module sage.modular.modsym.relation_matrix), 95
to_list() (sage.modular.modsym.heilbronn.Heilbronn method), 79
tuple() (sage.modular.modsym.manin_symbol.ManinSymbol method), 59
tuple() (sage.modular.modsym.p1list_nf.MSymbol method), 104
twisted_winding_element() (sage.modular.modsym.ambient.ModularSymbolsAmbient method), 40

U

u (sage.modular.modsym.manin_symbol.ManinSymbol attribute), 59

V

v (sage.modular.modsym.manin_symbol.ManinSymbol attribute), 59

W

weight() (sage.modular.modsym.boundary.BoundarySpace method), 75
weight() (sage.modular.modsym.manin_symbol.ManinSymbol method), 59
weight() (sage.modular.modsym.manin_symbol_list.ManinSymbolList method), 64
weight() (sage.modular.modsym.modular_symbols.ModularSymbol method), 55