Sage Reference Manual: General Hecke Algebras and Hecke Modules

Release 6.6

The Sage Development Team

April 18, 2015

CONTENTS

This chapter describes the basic functionality for modules over Hecke algebras, including decompositions, degeneracy maps and so on. For specific examples of Hecke algebras that use this functionality see Modular Symbols and Modular Forms.

CHAPTER

ONE

HECKE MODULES

class sage.modular.hecke.module.**HeckeModule_free_module**(*base_ring*, *level*, *weight*)

Bases: sage.modular.hecke.module.HeckeModule_generic

A Hecke module modeled on a free module over a commutative ring.

T(*n*)

Returns the n^{th} Hecke operator T_n. This function is a synonym for hecke_operator().

EXAMPLE:

```
sage: M = ModularSymbols(11,2)
sage: M.T(3)
Hecke operator T_3 on Modular Symbols ...
```

ambient()

Synonym for ambient_hecke_module. Return the ambient module associated to this module.

EXAMPLE:

```
sage: CuspForms(1, 12).ambient()
Modular Forms space of dimension 2 for Modular Group SL(2,Z) of weight 12 over Rational Fiel
```

ambient_hecke_module()

Return the ambient module associated to this module. As this is an abstract base class, return NotImplementedError.

EXAMPLE:

```
sage: sage.modular.hecke.module.HeckeModule_free_module(QQ, 10, 3).ambient_hecke_module()
Traceback (most recent call last):
...
NotImplementedError
```

ambient_module()

Synonym for ambient_hecke_module. Return the ambient module associated to this module.

EXAMPLE:

```
sage: CuspForms(1, 12).ambient_module()
Modular Forms space of dimension 2 for Modular Group SL(2,Z) of weight 12 over Rational Fiel
sage: sage.modular.hecke.module.HeckeModule_free_module(QQ, 10, 3).ambient_module()
Traceback (most recent call last):
...
NotImplementedError
```

atkin_lehner_operator(*d=None*)

Return the Atkin-Lehner operator W_d on this space, if defined, where d is a divisor of the level N such that N/d and d are coprime.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: w = M.atkin_lehner_operator()
sage: w
Hecke module morphism Atkin-Lehner operator W_11 defined by the matrix
[-1  0  0]
[ 0 -1  0]
[ 0  0 -1]
Domain: Modular Symbols space of dimension 3 for Gamma_0(11) of weight ...
Codomain: Modular Symbols space of dimension 3 for Gamma_0(11) of weight ...
sage: M = ModularSymbols(Gamma1(13))
sage: w = M.atkin_lehner_operator()
sage: w.fcp('x')
(x - 1)^7 * (x + 1)^8

sage: M = ModularSymbols(33)
sage: S = M.cuspidal_submodule()
sage: S.atkin_lehner_operator()
Hecke module morphism Atkin-Lehner operator W_33 defined by the matrix
[ 0 -1  0  1 -1  0]
[ 0 -1  0  0  0  0]
[ 0 -1  0  0 -1  1]
[ 1 -1  0  0 -1  0]
[ 0  0  0  0 -1  0]
[ 0 -1  1  0 -1  0]
Domain: Modular Symbols subspace of dimension 6 of Modular Symbols space ...
Codomain: Modular Symbols subspace of dimension 6 of Modular Symbols space ...

sage: S.atkin_lehner_operator(3)
Hecke module morphism Atkin-Lehner operator W_3 defined by the matrix
[ 0  1  0 -1  1  0]
[ 0  1  0  0  0  0]
[ 0  1  0  0  1 -1]
[-1  1  0  0  1  0]
[ 0  0  0  0  1  0]
[ 0  1 -1  0  1  0]
Domain: Modular Symbols subspace of dimension 6 of Modular Symbols space ...
Codomain: Modular Symbols subspace of dimension 6 of Modular Symbols space ...

sage: N = M.new_submodule()
sage: N.atkin_lehner_operator()
Hecke module morphism Atkin-Lehner operator W_33 defined by the matrix
[  1 2/5 4/5]
[  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 ...
```

basis()

Returns a basis for self.

EXAMPLES:

```
sage: m = ModularSymbols(43)
sage: m.basis()
((1,0), (1,31), (1,32), (1,38), (1,39), (1,40), (1,41))
```

basis_matrix()

Return the matrix of the basis vectors of self (as vectors in some ambient module)

EXAMPLE:

```
sage: CuspForms(1, 12).basis_matrix()
[1 0]
```

coordinate_vector(*x*)

Write x as a vector with respect to the basis given by self.basis().

EXAMPLES:

```
sage: S = ModularSymbols(11,2).cuspidal_submodule()
sage: S.0
(1,8)
sage: S.basis()
((1,8), (1,9))
sage: S.coordinate_vector(S.0)
(1, 0)
```

decomposition(*bound=None*, *anemic=True*, *height_guess=1*, *sort_by_basis=False*, *proof=None*)

Returns the maximal decomposition of this Hecke module under the action of Hecke operators of index coprime to the level. This is the finest decomposition of self that we can obtain using factors obtained by taking kernels of Hecke operators.

Each factor in the decomposition is a Hecke submodule obtained as the kernel of $f(T_n)^r$ acting on self, where n is coprime to the level and $r = 1$. If anemic is False, instead choose r so that $f(X)^r$ exactly divides the characteristic polynomial.

INPUT:

- `anemic` - bool (default: True), if True, use only Hecke operators of index coprime to the level.
- `bound` - int or None, (default: None). If None, use all Hecke operators up to the Sturm bound, and hence obtain the same result as one would obtain by using every element of the Hecke ring. If a fixed integer, decompose using only Hecke operators T_p, with p prime, up to bound.
- `sort_by_basis` - bool (default: `False`); If True the resulting decomposition will be sorted as if it was free modules, ignoring the Hecke module structure. This will save a lot of time.

OUTPUT:

- `list` - a list of subspaces of self.

EXAMPLES:

```
sage: ModularSymbols(17,2).decomposition()
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0(
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(
]
sage: ModularSymbols(Gamma1(10),4).decomposition()
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_1
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_1
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_1
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 18 for Gamma_1
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 18 for Gamma_1
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 18 for Gamma_1
]
sage: ModularSymbols(GammaH(12, [11])).decomposition()
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 9 for Congruen
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 9 for Congruen
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 9 for Congruen
```

```
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 9 for Congruen
Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension 9 for Congruen
]
```

TESTS:: sage: M = ModularSymbols(1000,2,sign=1).new_subspace().cuspidal_subspace() sage: M.decomposition(3, sort_by_basis = True) [Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 154 for Gamma_0(1000) of weight 2 with sign 1 over Rational Field]

degree()

The degree of this Hecke module (i.e. the rank of the ambient free module)

EXAMPLE:

```
sage: CuspForms(1, 12).degree()
2
```

diamond_bracket_matrix(*d*)

Return the matrix of the diamond bracket operator $\langle d \rangle$ on self.

EXAMPLES:

```
sage: M = ModularSymbols(DirichletGroup(5).0, 3)
sage: M.diamond_bracket_matrix(3)
[-zeta4      0]
[     0 -zeta4]
sage: ModularSymbols(Gamma1(5), 3).diamond_bracket_matrix(3)
[ 0 -1  0  0]
[ 1  0  0  0]
[ 0  0  0  1]
[ 0  0 -1  0]
```

diamond_bracket_operator(*d*)

Return the diamond bracket operator $\langle d \rangle$ on self.

EXAMPLES:

```
sage: M = ModularSymbols(DirichletGroup(5).0, 3)
sage: M.diamond_bracket_operator(3)
Diamond bracket operator <3> on Modular Symbols space of dimension 2 and level 5, weight 3,
```

dual_eigenvector(*names='alpha'*, *lift=True*, *nz=None*)

Return an eigenvector for the Hecke operators acting on the linear dual of this space. This eigenvector will have entries in an extension of the base ring of degree equal to the dimension of this space.

INPUT:

- `name` - print name of generator for eigenvalue field.
- `lift` - bool (default: True)
- `nz` - if not None, then normalize vector so dot product with this basis vector of ambient space is 1.

OUTPUT: A vector with entries possibly in an extension of the base ring. This vector is an eigenvector for all Hecke operators acting via their transpose.

If lift = False, instead return an eigenvector in the subspace for the Hecke operators on the dual space. I.e., this is an eigenvector for the restrictions of Hecke operators to the dual space.

Note:

1. The answer is cached so subsequent calls always return the same vector. However, the algorithm is randomized, so calls during another session may yield a different eigenvector. This function is used mainly for computing systems of Hecke eigenvalues.
2. One can also view a dual eigenvector as defining (via dot product) a functional phi from the ambient space of modular symbols to a field. This functional phi is an eigenvector for the dual action of Hecke operators on functionals.

EXAMPLE:

```
sage: ModularSymbols(14).cuspidal_subspace().simple_factors()[1].dual_eigenvector()
(0, 1, 0, 0, 0)
```

dual_hecke_matrix(*n*)

The matrix of the n^{th} Hecke operator acting on the dual embedded representation of self.

EXAMPLE:

```
sage: CuspForms(1, 24).dual_hecke_matrix(5)
[     79345647584250/2796203 50530996976060416/763363419]
[    195556757760000/2796203     124970165346810/2796203]
```

eigenvalue(*n*, *name='alpha'*)

Assuming that self is a simple space, return the eigenvalue of the n^{th} Hecke operator on self.

INPUT:

- n - index of Hecke operator
- name - print representation of generator of eigenvalue field

EXAMPLES:

```
sage: A = ModularSymbols(125,sign=1).new_subspace()[0]
sage: A.eigenvalue(7)
-3
sage: A.eigenvalue(3)
-alpha - 2
sage: A.eigenvalue(3,'w')
-w - 2
sage: A.eigenvalue(3,'z').charpoly('x')
x^2 + 3*x + 1
sage: A.hecke_polynomial(3)
x^2 + 3*x + 1

sage: M = ModularSymbols(Gamma1(17)).decomposition()[8].plus_submodule()
sage: M.eigenvalue(2,'a')
a
sage: M.eigenvalue(4,'a')
4/3*a^3 + 17/3*a^2 + 28/3*a + 8/3
```

Note:

1. In fact there are d systems of eigenvalues associated to self, where d is the rank of self. Each of the systems of eigenvalues is conjugate over the base field. This function chooses one of the systems and consistently returns eigenvalues from that system. Thus these are the coefficients a_n for $n \geq 1$ of a modular eigenform attached to self.

2. This function works even for Eisenstein subspaces, though it will not give the constant coefficient of one of the corresponding Eisenstein series (i.e., the generalized Bernoulli number).

TESTS:

This checks that trac ticket #15201 is fixed:

```
sage: M = ModularSymbols(5, 6, sign=1)
sage: f = M.decomposition()[0]
sage: f.eigenvalue(10)
50
```

factor_number()

If this Hecke module was computed via a decomposition of another Hecke module, this is the corresponding number. Otherwise return -1.

EXAMPLES:

```
sage: ModularSymbols(23)[0].factor_number()
0
sage: ModularSymbols(23).factor_number()
-1
```

gen(n)

Return the nth basis vector of the space.

EXAMPLE:

```
sage: ModularSymbols(23).gen(1)
(1,17)
```

hecke_matrix(n)

The matrix of the n^{th} Hecke operator acting on given basis.

EXAMPLE:

```
sage: C = CuspForms(1, 16)
sage: C.hecke_matrix(3)
[-3348]
```

hecke_operator(n)

Returns the n-th Hecke operator T_n.

INPUT:

- `ModularSymbols self` - Hecke equivariant space of modular symbols
- `int n` - an integer at least 1.

EXAMPLES:

```
sage: M = ModularSymbols(11,2)
sage: T = M.hecke_operator(3) ; T
Hecke operator T_3 on Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with
sage: T.matrix()
[ 4  0 -1]
[ 0 -1  0]
[ 0  0 -1]
```

```
sage: T(M.0)
4*(1,0) - (1,9)
sage: S = M.cuspidal_submodule()
sage: T = S.hecke_operator(3) ; T
Hecke operator T_3 on Modular Symbols subspace of dimension 2 of Modular Symbols space of di
sage: T.matrix()
[-1  0]
[ 0 -1]
sage: T(S.0)
-(1,8)
```

hecke_polynomial(*n*, *var='x'*)

Return the characteristic polynomial of the n^{th} Hecke operator acting on this space.

INPUT:

- n - integer

OUTPUT: a polynomial

EXAMPLE:

```
sage: ModularSymbols(11,2).hecke_polynomial(3)
x^3 - 2*x^2 - 7*x - 4
```

is_simple()

Return True if this space is simple as a module for the corresponding Hecke algebra. Raises NotImplementedError, as this is an abstract base class.

EXAMPLE:

```
sage: sage.modular.hecke.module.HeckeModule_free_module(QQ, 10, 3).is_simple()
Traceback (most recent call last):
...
NotImplementedError
```

is_splittable()

Returns True if and only if only it is possible to split off a nontrivial generalized eigenspace of self as the kernel of some Hecke operator (not necessarily prime to the level). Note that the direct sum of several copies of the same simple module is not splittable in this sense.

EXAMPLE:

```
sage: M = ModularSymbols(Gamma0(64)).cuspidal_subspace()
sage: M.is_splittable()
True
sage: M.simple_factors()[0].is_splittable()
False
```

is_splittable_anemic()

Returns true if and only if only it is possible to split off a nontrivial generalized eigenspace of self as the kernel of some Hecke operator of index coprime to the level. Note that the direct sum of several copies of the same simple module is not splittable in this sense.

EXAMPLE:

```
sage: M = ModularSymbols(Gamma0(64)).cuspidal_subspace()
sage: M.is_splittable_anemic()
True
```

```
sage: M.simple_factors()[0].is_splittable_anemic()
False
```

is_submodule(*other*)

Return True if self is a submodule of other.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma0(64))
sage: M[0].is_submodule(M)
True
sage: CuspForms(1, 24).is_submodule(ModularForms(1, 24))
True
sage: CuspForms(1, 12).is_submodule(CuspForms(3, 12))
False
```

ngens()

Number of generators of self (equal to the rank).

EXAMPLE:

```
sage: ModularForms(1, 12).ngens()
2
```

projection()

Return the projection map from the ambient space to self.

ALGORITHM: Let B be the matrix whose columns are obtained by concatenating together a basis for the factors of the ambient space. Then the projection matrix onto self is the submatrix of B^{-1} obtained from the rows corresponding to self, i.e., if the basis vectors for self appear as columns n through m of B, then the projection matrix is got from rows n through m of B^{-1}. This is because projection with respect to the B basis is just given by an $m - n + 1$ row slice P of a diagonal matrix D with 1's in the n through m positions, so projection with respect to the standard basis is given by $P \cdot B^{-1}$, which is just rows n through m of B^{-1}.

EXAMPLES:

```
sage: e = EllipticCurve('34a')
sage: m = ModularSymbols(34); s = m.cuspidal_submodule()
sage: d = s.decomposition(7)
sage: d
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 9 for Gamma_0(
]
sage: a = d[0]; a
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(
sage: pi = a.projection()
sage: pi(m([0,oo]))
-1/6*(2,7) + 1/6*(2,13) - 1/6*(2,31) + 1/6*(2,33)
sage: M = ModularSymbols(53,sign=1)
sage: S = M.cuspidal_subspace()[1] ; S
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 5 for Gamma_0(
sage: p = S.projection()
sage: S.basis()
((1,33) - (1,37), (1,35), (1,49))
sage: [ p(x) for x in S.basis() ]
[(1,33) - (1,37), (1,35), (1,49)]
```

system_of_eigenvalues(*n*, *name='alpha'*)

Assuming that self is a simple space of modular symbols, return the eigenvalues $[a_1, \ldots, a_n max]$ of the Hecke operators on self. See `self.eigenvalue(n)` for more details.

INPUT:

- `n` - number of eigenvalues
- `alpha` - name of generate for eigenvalue field

EXAMPLES: These computations use pseudo-random numbers, so we set the seed for reproducible testing.

```
sage: set_random_seed(0)
```

The computations also use cached results from other computations, so we clear the caches for reproducible testing.

```
sage: ModularSymbols_clear_cache()
```

We compute eigenvalues for newforms of level 62.

```
sage: M = ModularSymbols(62,2,sign=-1)
sage: S = M.cuspidal_submodule().new_submodule()
sage: [A.system_of_eigenvalues(3) for A in S.decomposition()]
[[1, 1, 0], [1, -1, 1/2*alpha + 1/2]]
```

Next we define a function that does the above:

```
sage: def b(N,k=2):
...     t=cputime()
...     S = ModularSymbols(N,k,sign=-1).cuspidal_submodule().new_submodule()
...     for A in S.decomposition():
...         print N, A.system_of_eigenvalues(5)

sage: b(63)
63 [1, 1, 0, -1, 2]
63 [1, alpha, 0, 1, -2*alpha]
```

This example illustrates finding field over which the eigenvalues are defined:

```
sage: M = ModularSymbols(23,2,sign=1).cuspidal_submodule().new_submodule()
sage: v = M.system_of_eigenvalues(10); v
[1, alpha, -2*alpha - 1, -alpha - 1, 2*alpha, alpha - 2, 2*alpha + 2, -2*alpha - 1, 2, -2*al
sage: v[0].parent()
Number Field in alpha with defining polynomial x^2 + x - 1
```

This example illustrates setting the print name of the eigenvalue field.

```
sage: A = ModularSymbols(125,sign=1).new_subspace()[0]
sage: A.system_of_eigenvalues(10)
[1, alpha, -alpha - 2, -alpha - 1, 0, -alpha - 1, -3, -2*alpha - 1, 3*alpha + 2, 0]
sage: A.system_of_eigenvalues(10,'x')
[1, x, -x - 2, -x - 1, 0, -x - 1, -3, -2*x - 1, 3*x + 2, 0]
```

weight()

Returns the weight of this Hecke module.

INPUT:

- `self` - an arbitrary Hecke module

OUTPUT:

- int - the weight

EXAMPLES:

```
sage: m = ModularSymbols(20, weight=2)
sage: m.weight()
2
```

zero_submodule()

Return the zero submodule of self.

EXAMPLES:

```
sage: ModularSymbols(11,4).zero_submodule()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 6 for Gamma_0(
sage: CuspForms(11,4).zero_submodule()
Modular Forms subspace of dimension 0 of Modular Forms space of dimension 4 for Congruence S
```

class sage.modular.hecke.module.**HeckeModule_generic**(*base_ring*, *level*, *category=None*)

Bases: sage.modules.module.Module_old

A very general base class for Hecke modules.

We define a Hecke module of weight k to be a module over a commutative ring equipped with an action of operators T_m for all positive integers m coprime to some integer $n`(thelevel), whichsatisfy`T_rT_s = T_{rs}$ for r, s coprime, and for powers of a prime p, $T_{p^r} = T_pT_{p^{r-1}} - \varepsilon(p)p^{k-1}T_{p^{r-2}}$, where $\varepsilon(p)$ is some endomorphism of the module which commutes with the T_m.

We distinguish between *full* Hecke modules, which also have an action of operators T_m for m not assumed to be coprime to the level, and *anemic* Hecke modules, for which this does not hold.

anemic_hecke_algebra()

Return the Hecke algebra associated to this Hecke module.

EXAMPLES:

```
sage: T = ModularSymbols(1,12).hecke_algebra()
sage: A = ModularSymbols(1,12).anemic_hecke_algebra()
sage: T == A
False
sage: A
Anemic Hecke algebra acting on Modular Symbols space of dimension 3 for Gamma_0(1) of weight
sage: A.is_anemic()
True
```

character()

The character of this space. As this is an abstract base class, return None.

EXAMPLE:

```
sage: sage.modular.hecke.module.HeckeModule_generic(QQ, 10).character() is None
True
```

dimension()

Synonym for rank.

EXAMPLE:

```
sage: M = sage.modular.hecke.module.HeckeModule_generic(QQ, 10).dimension()
Traceback (most recent call last):
...
NotImplementedError: Derived subclasses must implement rank
```

hecke_algebra()
Return the Hecke algebra associated to this Hecke module.

EXAMPLES:

```
sage: T = ModularSymbols(Gamma1(5),3).hecke_algebra()
sage: T
Full Hecke algebra acting on Modular Symbols space of dimension 4 for Gamma_1(5) of weight 3
sage: T.is_anemic()
False

sage: M = ModularSymbols(37,sign=1)
sage: E, A, B = M.decomposition()
sage: A.hecke_algebra() == B.hecke_algebra()
False
```

is_full_hecke_module()
Return True if this space is invariant under all Hecke operators.

Since self is guaranteed to be an anemic Hecke module, the significance of this function is that it also ensures invariance under Hecke operators of index that divide the level.

EXAMPLES:

```
sage: M = ModularSymbols(22); M.is_full_hecke_module()
True
sage: M.submodule(M.free_module().span([M.0.list()]), check=False).is_full_hecke_module()
False
```

is_hecke_invariant(n)
Return True if self is invariant under the Hecke operator T_n.

Since self is guaranteed to be an anemic Hecke module it is only interesting to call this function when n is not coprime to the level.

EXAMPLES:

```
sage: M = ModularSymbols(22).cuspidal_subspace()
sage: M.is_hecke_invariant(2)
True
```

We use check=False to create a nasty "module" that is not invariant under T_2:

```
sage: S = M.submodule(M.free_module().span([M.0.list()]), check=False); S
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 7 for Gamma_0(
sage: S.is_hecke_invariant(2)
False
sage: [n for n in range(1,12) if S.is_hecke_invariant(n)]
[1, 3, 5, 7, 9, 11]
```

is_zero()
Return True if this Hecke module has dimension 0.

EXAMPLES:

```
sage: ModularSymbols(11).is_zero()
False
sage: ModularSymbols(11).old_submodule().is_zero()
True
sage: CuspForms(10).is_zero()
True
sage: CuspForms(1,12).is_zero()
False
```

level()
Returns the level of this modular symbols space.

INPUT:

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

OUTPUT:

- `int` - the level

EXAMPLES:

```
sage: m = ModularSymbols(20)
sage: m.level()
20
```

rank()
Return the rank of this module over its base ring. Returns NotImplementedError, since this is an abstract base class.

EXAMPLES:

```
sage: sage.modular.hecke.module.HeckeModule_generic(QQ, 10).rank()
Traceback (most recent call last):
...
NotImplementedError: Derived subclasses must implement rank
```

submodule(*X*)
Return the submodule of self corresponding to X. As this is an abstract base class, this raises a NotImplementedError.

EXAMPLES:

```
sage: sage.modular.hecke.module.HeckeModule_generic(QQ, 10).submodule(0)
Traceback (most recent call last):
...
NotImplementedError: Derived subclasses should implement submodule
```

sage.modular.hecke.module.**is_HeckeModule**(*x*)
Return True if x is a Hecke module.

EXAMPLES:

```
sage: from sage.modular.hecke.module import is_HeckeModule
sage: is_HeckeModule(ModularForms(Gamma0(7), 4))
True
sage: is_HeckeModule(QQ^3)
False
sage: is_HeckeModule(J0(37).homology())
True
```

CHAPTER

TWO

AMBIENT HECKE MODULES

class sage.modular.hecke.ambient_module.**AmbientHeckeModule**(*base_ring*, *rank*, *level*, *weight*)

Bases: sage.modular.hecke.module.HeckeModule_free_module

An ambient Hecke module, i.e. a Hecke module that is isomorphic as a module over its base ring R to the standard free module R^k for some k. This is the base class for ambient spaces of modular forms and modular symbols, and for Brandt modules.

ambient_hecke_module()

Return the ambient space that contains this ambient space. This is, of course, just this space again.

EXAMPLE:

```
sage: M = ModularForms(11, 4); M.ambient_hecke_module() is M
True
```

complement()

Return the largest Hecke-stable complement of this space.

EXAMPLES:

```
sage: M=ModularSymbols(11,2,1)
sage: M
Modular Symbols space of dimension 2 for Gamma_0(11) of weight 2 with sign 1 over Rational F
sage: M.complement()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 2 for Gamma_0(
sage: C=M.cuspidal_subspace()
sage: C
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0(
sage: C.complement()
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0(
```

decomposition_matrix()

Returns the matrix whose columns form a basis for the canonical sorted decomposition of self coming from the Hecke operators.

If the simple factors are $D_0, \ldots, D_n$, then the first few columns are an echelonized basis for D_0, the next an echelonized basis for D_1, the next for D_2, etc.

EXAMPLE:

```
sage: S = ModularSymbols(37, 2)
sage: S.decomposition_matrix()
[   1    0    0    0 -1/3]
[   0    1   -1    0  1/2]
[   0    0    0    1 -1/2]
[   0    1    1    1    0]
[   0    0    0    0    1]
```

decomposition_matrix_inverse()

Returns the inverse of the matrix returned by decomposition_matrix().

EXAMPLE:

```
sage: S = ModularSymbols(37, 2)
sage: t = S.decomposition_matrix_inverse(); t
[   1    0    0    0  1/3]
[   0  1/2 -1/2  1/2 -1/2]
[   0 -1/2 -1/2  1/2    0]
[   0    0    1    0  1/2]
[   0    0    0    0    1]
sage: t * S.decomposition_matrix() == 1
True
```

degeneracy_map(*codomain*, *t=1*)

The t-th degeneracy map from self to the module `codomain`. The level of the codomain must be a divisor or multiple of level, and t must be a divisor of the quotient.

INPUT:

- `codomain` - a Hecke module, which should be of the same type as self, or a positive integer (in which case Sage will use `hecke_module_of_level()` to find the "natural" module of the corresponding level).
- `t` - int, the parameter of the degeneracy map, i.e., the map is related to $f(q)$ - $f(q^t)$.

OUTPUT: A morphism from self to codomain.

EXAMPLES:

```
sage: M = ModularSymbols(11,sign=1)
sage: d1 = M.degeneracy_map(33); d1
Hecke module morphism degeneracy map corresponding to f(q) |--> f(q) defined by the matrix
[ 1  0  0  0 -2 -1]
[ 0  0 -2  2  0  0]
Domain: Modular Symbols space of dimension 2 for Gamma_0(11) of weight ...
Codomain: Modular Symbols space of dimension 6 for Gamma_0(33) of weight ...
sage: M.degeneracy_map(33,3).matrix()
[ 3  2  2  0 -2  1]
[ 0  2  0 -2  0  0]
sage: M = ModularSymbols(33,sign=1)
sage: d2 = M.degeneracy_map(11); d2.matrix()
[  1   0]
[  0 1/2]
[  0  -1]
[  0   1]
[ -1   0]
[ -1   0]
sage: (d2*d1).matrix()
[4 0]
[0 4]

sage: M = ModularSymbols(3,12,sign=1)
sage: M.degeneracy_map(1)
Hecke module morphism degeneracy map corresponding to f(q) |--> f(q) defined by the matrix
[1 0]
[0 0]
[0 1]
[0 1]
```

```
[0 1]
Domain: Modular Symbols space of dimension 5 for Gamma_0(3) of weight ...
Codomain: Modular Symbols space of dimension 2 for Gamma_0(1) of weight ...

sage: S = M.cuspidal_submodule()
sage: S.degeneracy_map(1)
Hecke module morphism defined by the matrix
[1 0]
[0 0]
[0 0]
Domain: Modular Symbols subspace of dimension 3 of Modular Symbols space ...
Codomain: Modular Symbols space of dimension 2 for Gamma_0(1) of weight ...

sage: D = ModularSymbols(10,4).cuspidal_submodule().decomposition()
sage: D
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 10 for Gamma_0
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 10 for Gamma_0
]
sage: D[1].degeneracy_map(5)
Hecke module morphism defined by the matrix
[   0    0   -1    1]
[   0  1/2  3/2   -2]
[   0   -1    1    0]
[   0 -3/4 -1/4    1]
Domain: Modular Symbols subspace of dimension 4 of Modular Symbols space ...
Codomain: Modular Symbols space of dimension 4 for Gamma_0(5) of weight ...
```

We check for a subtle caching bug that came up in work on trac #10453:

```
sage: loads(dumps(J0(33).decomposition()[0].modular_symbols()))
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(
```

We check that certain absurd inputs are correctly caught:

```
sage: chi = kronecker_character(7)
sage: ModularSymbols(Gamma0(7), 4).degeneracy_map(ModularSymbols(chi, 4))
Traceback (most recent call last):
...
ValueError: The characters of the domain and codomain must match
```

dual_free_module()

The free module dual to self, as a submodule of the dual module of the ambient space. As this space is ambient anyway, this just returns self.free_module().

EXAMPLE:

```
sage: M = ModularForms(2,8); M.dual_free_module()
Vector space of dimension 3 over Rational Field
sage: M.dual_free_module() is M.free_module()
True
```

fcp(*n*, *var='x'*)

Returns the factorization of the characteristic polynomial of the Hecke operator T_n of index n acting on this space.

INPUT:

- `self` - Hecke module invariant under the Hecke operator of index n.

- int n - a positive integer.
- var - variable of polynomial (default x)

OUTPUT:

- list - list of the pairs (g,e), where g is an irreducible factor of the characteristic polynomial of T_n, and e is its multiplicity.

EXAMPLES:

```
sage: m = ModularSymbols(23, 2, sign=1)
sage: m.fcp(2)
(x - 3) * (x^2 + x - 1)
sage: m.hecke_operator(2).charpoly('x').factor()
(x - 3) * (x^2 + x - 1)
```

free_module()

Return the free module underlying this ambient Hecke module (the forgetful functor from Hecke modules to modules over the base ring)

EXAMPLE:

```
sage: ModularForms(59, 2).free_module()
Vector space of dimension 6 over Rational Field
```

hecke_bound()

Return an integer B such that the Hecke operators T_n, for $n \leq B$, generate the full Hecke algebra as a module over the base ring. Note that we include the n with n not coprime to the level.

At present this returns an unproven guess for non-cuspidal spaces which appears to be valid for $M_k(\Gamma_0(N))$, where k and N are the weight and level of self. (It is clearly valid for *cuspidal* spaces of any fixed character, as a consequence of the Sturm bound theorem.) It returns a hopelessly wrong answer for spaces of full level Γ_1.

TODO: Get rid of this dreadful bit of code.

EXAMPLE:

```
sage: ModularSymbols(17, 4).hecke_bound()
15
sage: ModularSymbols(Gamma1(17), 4).hecke_bound() # wrong!
15
```

hecke_images(i, v)

Return images of the i-th standard basis vector under the Hecke operators T_p for all integers in v.

INPUT:

- i - nonnegative integer
- v - a list of positive integer

OUTPUT:

- matrix - whose rows are the Hecke images

EXAMPLES:

```
sage: M = ModularSymbols(DirichletGroup(13).0, 3)
sage: M.T(2)(M.0).element()
(zeta12 + 4, 0, -1, 1)
sage: M.hecke_images(0, [1,2])
[         1          0          0          0]
[zeta12 + 4          0         -1          1]
```

hecke_module_of_level(*level*)

Return the Hecke module corresponding to self at the given level, which should be either a divisor or a multiple of the level of self. This raises NotImplementedError, and should be overridden in derived classes.

EXAMPLE:

```
sage: sage.modular.hecke.ambient_module.AmbientHeckeModule.hecke_module_of_level(ModularForm
Traceback (most recent call last):
...
NotImplementedError
```

intersection(*other*)

Returns the intersection of self and other, which must both lie in a common ambient space of modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(43, sign=1)
sage: A = M[0] + M[1]
sage: B = M[1] + M[2]
sage: A.rank(), B.rank()
(2, 3)
sage: C = A.intersection(B); C.rank()  # TODO
1
```

is_ambient()

Returns True if and only if self is an ambient Hecke module.

Warning: self can only be ambient by being of type AmbientHeckeModule.
For example, decomposing a simple ambient space yields a single factor, and that factor is *not* considered an ambient space.

EXAMPLES:

```
sage: m = ModularSymbols(10)
sage: m.is_ambient()
True

sage: a = m[0]  # the unique simple factor
sage: a == m
True
sage: a.is_ambient()
False
```

is_full_hecke_module(*compute=True*)

Returns True if this space is invariant under the action of all Hecke operators, even those that divide the level. This is always true for ambient Hecke modules, so return True.

EXAMPLE:

```
sage: ModularSymbols(11, 4).is_full_hecke_module()
True
```

is_new(*p=None*)

Return True if this module is entirely new.

EXAMPLE:

```
sage: ModularSymbols(11, 4).is_new()
False
sage: ModularSymbols(1, 12).is_new()
True
```

is_old(*p=None*)

Return True if this module is entirely old.

EXAMPLE:

```
sage: ModularSymbols(22).is_old()
True
sage: ModularSymbols(3, 12).is_old()
False
```

is_submodule(*V*)

Returns True if and only if self is a submodule of V. Since this is an ambient space, this returns True if and only if V is equal to self.

EXAMPLE:

```
sage: ModularSymbols(1, 4).is_submodule(ModularSymbols(11,4))
False
sage: ModularSymbols(11, 4).is_submodule(ModularSymbols(11,4))
True
```

linear_combination_of_basis(*v*)

Given a list or vector of length equal to the dimension of self, construct the appropriate linear combination of the basis vectors of self.

EXAMPLE:

```
sage: ModularForms(3, 12).linear_combination_of_basis([1,0,0,0,1])
2*q + 2049*q^2 + 177147*q^3 + 4196177*q^4 + 48830556*q^5 + O(q^6)
```

new_submodule(*p=None*)

Returns the new or p-new submodule of self.

INPUT:

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

OUTPUT: the new or p-new submodule of self, i.e. the intersection of the kernel of the degeneracy lowering maps to level N/p (for the given prime p, or for all prime divisors of N if p is not given).

If self is cuspidal this is a Hecke-invariant complement of the corresponding old submodule, but this may break down on Eisenstein subspaces (see the amusing example in William Stein's book of a form which is new and old at the same time).

EXAMPLES:

```
sage: m = ModularSymbols(33); m.rank()
9
sage: m.new_submodule().rank()
3
sage: m.new_submodule(3).rank()
4
sage: m.new_submodule(11).rank()
8
```

nonembedded_free_module()

Return the free module corresponding to self as an abstract free module (rather than as a submodule of an

ambient free module). As this module is ambient anyway, this just returns `self.free_module()`.

EXAMPLES:

```
sage: M = ModularSymbols(11, 2)
sage: M.nonembedded_free_module() is M.free_module()
True
```

old_submodule(*p=None*)

Returns the old or p-old submodule of self, i.e. the sum of the images of the degeneracy maps from level N/p (for the given prime p, or for all primes p dividing N if p is not given).

INPUT:

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

OUTPUT: the old or p-old submodule of self

EXAMPLES:

```
sage: m = ModularSymbols(33); m.rank()
9
sage: m.old_submodule().rank()
7
sage: m.old_submodule(3).rank()
6
sage: m.new_submodule(11).rank()
8

sage: e = DirichletGroup(16)([-1, 1])
sage: M = ModularSymbols(e, 3, sign=1); M
Modular Symbols space of dimension 4 and level 16, weight 3, character [-1, 1], sign 1, over
sage: M.old_submodule()
Modular Symbols subspace of dimension 3 of Modular Symbols space of dimension 4 and level 16
```

Illustrate that trac 10664 is fixed:

```
sage: ModularSymbols(DirichletGroup(42)[7], 6, sign=1).old_subspace(3)
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 40 and level 4
```

rank()

Return the rank of this ambient Hecke module.

OUTPUT:

Integer

EXAMPLES:

```
sage: M = sage.modular.hecke.ambient_module.AmbientHeckeModule(QQ, 3, 11, 2); M
Generic ambient Hecke module of rank 3, level 11 and weight 2 over Rational Field
sage: M.rank()
3
```

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

Return the Hecke submodule of self generated by M, which may be a submodule of the free module of self, or a list of elements of self.

EXAMPLE:

```
sage: M = ModularForms(37, 2)
sage: A = M.submodule([M.newforms()[0].element(), M.newforms()[1].element()]); A
Modular Forms subspace of dimension 2 of Modular Forms space of dimension 3 for Congruence S
```

submodule_from_nonembedded_module(*V*, *Vdual=None*, *check=True*)

Create a submodule of this module, from a submodule of an ambient free module of the same rank as the rank of self.

INPUT:

- `V` - submodule of ambient free module of the same rank as the rank of self.
- `Vdual` - used to pass in dual submodule (may be None)
- `check` - whether to check that submodule is Hecke equivariant

OUTPUT: Hecke submodule of self

EXAMPLE:

```
sage: V = QQ^8
sage: ModularForms(24, 2).submodule_from_nonembedded_module(V.submodule([0]))
Modular Forms subspace of dimension 0 of Modular Forms space of dimension 8 for Congruence S
```

submodule_generated_by_images(*M*)

Return the submodule of this ambient modular symbols space generated by the images under all degeneracy maps of M. The space M must have the same weight, sign, and group or character as this ambient space.

EXAMPLES:

```
sage: ModularSymbols(6, 12).submodule_generated_by_images(ModularSymbols(1,12))
Modular Symbols subspace of dimension 12 of Modular Symbols space of dimension 22 for Gamma_
```

sage.modular.hecke.ambient_module.**is_AmbientHeckeModule**(*x*)

Return True if x is of type AmbientHeckeModule.

EXAMPLES:

```
sage: from sage.modular.hecke.ambient_module import is_AmbientHeckeModule
sage: is_AmbientHeckeModule(ModularSymbols(6))
True
sage: is_AmbientHeckeModule(ModularSymbols(6).cuspidal_subspace())
False
sage: is_AmbientHeckeModule(ModularForms(11))
True
sage: is_AmbientHeckeModule(BrandtModule(2, 3))
True
```

CHAPTER

THREE

SUBMODULES OF HECKE MODULES

class sage.modular.hecke.submodule.**HeckeSubmodule**(*ambient*, *submodule*, *dual_free_module=None*, *check=True*)

Bases: sage.modular.hecke.module.HeckeModule_free_module

Submodule of a Hecke module.

ambient()

Synonym for ambient_hecke_module.

EXAMPLES:

```
sage: CuspForms(2, 12).ambient()
Modular Forms space of dimension 4 for Congruence Subgroup Gamma0(2) of weight 12 over Ratio
```

ambient_hecke_module()

Return the ambient Hecke module of which this is a submodule.

EXAMPLES:

```
sage: CuspForms(2, 12).ambient_hecke_module()
Modular Forms space of dimension 4 for Congruence Subgroup Gamma0(2) of weight 12 over Ratio
```

complement(*bound=None*)

Return the largest Hecke-stable complement of this space.

EXAMPLES:

```
sage: M = ModularSymbols(15, 6).cuspidal_subspace()
sage: M.complement()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 20 for Gamma_0
sage: E = EllipticCurve("128a")
sage: ME = E.modular_symbol_space()
sage: ME.complement()
Modular Symbols subspace of dimension 17 of Modular Symbols space of dimension 18 for Gamma_
```

degeneracy_map(*level*, *t=1*)

The t-th degeneracy map from self to the space of ambient modular symbols of the given level. The level of self must be a divisor or multiple of level, and t must be a divisor of the quotient.

INPUT:

- `level` - int, the level of the codomain of the map (positive int).
- `t` - int, the parameter of the degeneracy map, i.e., the map is related to $f(q)$ - $f(q^t)$.

OUTPUT: A linear function from self to the space of modular symbols of given level with the same weight, character, sign, etc., as this space.

EXAMPLES:

```
sage: D = ModularSymbols(10,4).cuspidal_submodule().decomposition(); D
[
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 10 for Gamma_0
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 10 for Gamma_0
]
sage: d = D[1].degeneracy_map(5); d
Hecke module morphism defined by the matrix
[   0    0   -1    1]
[   0  1/2  3/2   -2]
[   0   -1    1    0]
[   0 -3/4 -1/4    1]
Domain: Modular Symbols subspace of dimension 4 of Modular Symbols space ...
Codomain: Modular Symbols space of dimension 4 for Gamma_0(5) of weight ...

sage: d.rank()
2
sage: d.kernel()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 10 for Gamma_0
sage: d.image()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 for Gamma_0(
```

dual_free_module(*bound=None*, *anemic=True*, *use_star=True*)

Compute embedded dual free module if possible. In general this won't be possible, e.g., if this space is not Hecke equivariant, possibly if it is not cuspidal, or if the characteristic is not 0. In all these cases we raise a RuntimeError exception.

If use_star is True (which is the default), we also use the +/- eigenspaces for the star operator to find the dual free module of self. If self does not have a star involution, use_star will automatically be set to False.

EXAMPLES:

```
sage: M = ModularSymbols(11, 2)
sage: M.dual_free_module()
Vector space of dimension 3 over Rational Field
sage: Mpc = M.plus_submodule().cuspidal_submodule()
sage: Mcp = M.cuspidal_submodule().plus_submodule()
sage: Mcp.dual_free_module() == Mpc.dual_free_module()
True
sage: Mpc.dual_free_module()
Vector space of degree 3 and dimension 1 over Rational Field
Basis matrix:
[  1 5/2   5]

sage: M = ModularSymbols(35,2).cuspidal_submodule()
sage: M.dual_free_module(use_star=False)
Vector space of degree 9 and dimension 6 over Rational Field
Basis matrix:
[   1    0    0    0   -1    0    0    4   -2]
[   0    1    0    0    0    0    0 -1/2  1/2]
[   0    0    1    0    0    0    0 -1/2  1/2]
[   0    0    0    1   -1    0    0    1    0]
[   0    0    0    0    0    1    0   -2    1]
[   0    0    0    0    0    0    1   -2    1]

sage: M = ModularSymbols(40,2)
sage: Mmc = M.minus_submodule().cuspidal_submodule()
sage: Mcm = M.cuspidal_submodule().minus_submodule()
sage: Mcm.dual_free_module() == Mmc.dual_free_module()
```

```
True
sage: Mcm.dual_free_module()
Vector space of degree 13 and dimension 3 over Rational Field
Basis matrix:
[ 0  1  0  0  0  0  1  0 -1 -1  1 -1  0]
[ 0  0  1  0 -1  0 -1  0  1  0  0  0  0]
[ 0  0  0  0  0  1  1  0 -1  0  0  0  0]

sage: M = ModularSymbols(43).cuspidal_submodule()
sage: S = M[0].plus_submodule() + M[1].minus_submodule()
sage: S.dual_free_module(use_star=False)
Traceback (most recent call last):
...
RuntimeError: Computation of complementary space failed (cut down to rank 7, but should have
sage: S.dual_free_module().dimension() == S.dimension()
True
```

We test that #5080 is fixed:

```
sage: EllipticCurve('128a').congruence_number()
32
```

free_module()

Return the free module corresponding to self.

EXAMPLES:

```
sage: M = ModularSymbols(33,2).cuspidal_subspace() ; M
Modular Symbols subspace of dimension 6 of Modular Symbols space of dimension 9 for Gamma_0(
sage: M.free_module()
Vector space of degree 9 and dimension 6 over Rational Field
Basis matrix:
[ 0  1  0  0  0  0  0 -1  1]
[ 0  0  1  0  0  0  0 -1  1]
[ 0  0  0  1  0  0  0 -1  1]
[ 0  0  0  0  1  0  0 -1  1]
[ 0  0  0  0  0  1  0 -1  1]
[ 0  0  0  0  0  0  1 -1  0]
```

hecke_bound()

Compute the Hecke bound for self; that is, a number n such that the T_m for m = n generate the Hecke algebra.

EXAMPLES:

```
sage: M = ModularSymbols(24,8)
sage: M.hecke_bound()
53
sage: M.cuspidal_submodule().hecke_bound()
32
sage: M.eisenstein_submodule().hecke_bound()
53
```

intersection(*other*)

Returns the intersection of self and other, which must both lie in a common ambient space of modular symbols.

EXAMPLES:

```
sage: M = ModularSymbols(43, sign=1)
sage: A = M[0] + M[1]
sage: B = M[1] + M[2]
sage: A.dimension(), B.dimension()
(2, 3)
sage: C = A.intersection(B); C.dimension()
1
```

TESTS:

```
sage: M = ModularSymbols(1,80)
sage: M.plus_submodule().cuspidal_submodule().sign() # indirect doctest
1
```

is_ambient()

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

EXAMPLES:

```
sage: M = ModularSymbols(17,4)
sage: M.cuspidal_subspace().is_ambient()
False
sage: A = M.ambient_hecke_module()
sage: S = A.submodule(A.basis())
sage: sage.modular.hecke.submodule.HeckeSubmodule.is_ambient(S)
True
```

is_new(*p=None*)

Returns True if this Hecke module is p-new. If p is None, returns True if it is new.

EXAMPLES:

```
sage: M = ModularSymbols(1,16)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S.is_new()
True
```

is_old(*p=None*)

Returns True if this Hecke module is p-old. If p is None, returns True if it is old.

EXAMPLES:

```
sage: M = ModularSymbols(50,2)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.old_submodule().free_module())
sage: S.is_old()
True
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.new_submodule().free_module())
sage: S.is_old()
False
```

is_submodule(*V*)

Returns True if and only if self is a submodule of V.

EXAMPLES:

```
sage: M = ModularSymbols(30,4)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S.is_submodule(M)
True
sage: SS = sage.modular.hecke.submodule.HeckeSubmodule(M, M.old_submodule().free_module())
sage: S.is_submodule(SS)
False
```

linear_combination_of_basis(*v*)

Return the linear combination of the basis of self given by the entries of v.

EXAMPLES:

```
sage: M = ModularForms(Gamma0(2),12)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S.basis()
(q + 252*q^3 - 2048*q^4 + 4830*q^5 + O(q^6), q^2 - 24*q^4 + O(q^6))
sage: S.linear_combination_of_basis([3,10])
3*q + 10*q^2 + 756*q^3 - 6384*q^4 + 14490*q^5 + O(q^6)
```

module()

Alias for code{self.free_module()}.

EXAMPLES:

```
sage: M = ModularSymbols(17,4).cuspidal_subspace()
sage: M.free_module() is M.module()
True
```

new_submodule(*p=None*)

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

EXAMPLES:

```
sage: M = ModularSymbols(20,4)
sage: M.new_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_0
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S
Rank 12 submodule of a Hecke module of level 20
sage: S.new_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_0
```

nonembedded_free_module()

Return the free module corresponding to self as an abstract free module, i.e. not as an embedded vector space.

EXAMPLES:

```
sage: M = ModularSymbols(12,6)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S
Rank 14 submodule of a Hecke module of level 12
sage: S.nonembedded_free_module()
Vector space of dimension 14 over Rational Field
```

old_submodule(*p=None*)

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

EXAMPLES: We compute the old and new submodules of $\mathbf{S}_2(\Gamma_0(33))$.

```
sage: M = ModularSymbols(33); S = M.cuspidal_submodule(); S
Modular Symbols subspace of dimension 6 of Modular Symbols space of dimension 9 for Gamma_0(
sage: S.old_submodule()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 9 for Gamma_0(
sage: S.new_submodule()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(
```

rank()
Return the rank of self as a free module over the base ring.

EXAMPLE:

```
sage: ModularSymbols(6, 4).cuspidal_subspace().rank()
2
sage: ModularSymbols(6, 4).cuspidal_subspace().dimension()
2
```

submodule(*M*, *Mdual=None*, *check=True*)
Construct a submodule of self from the free module M, which must be a subspace of self.

EXAMPLES:

```
sage: M = ModularSymbols(18,4)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: S[0]
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_0
sage: S.submodule(S[0].free_module())
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_0
```

submodule_from_nonembedded_module(*V*, *Vdual=None*, *check=True*)
Construct a submodule of self from V. Here V should be a subspace of a vector space whose dimension is the same as that of self.

INPUT:

- V - submodule of ambient free module of the same rank as the rank of self.
- check - whether to check that V is Hecke equivariant.

OUTPUT: Hecke submodule of self

EXAMPLES:

```
sage: M = ModularSymbols(37,2)
sage: S = sage.modular.hecke.submodule.HeckeSubmodule(M, M.cuspidal_submodule().free_module(
sage: V = (QQ**4).subspace([[1,-1,0,1/2],[0,0,1,-1/2]])
sage: S.submodule_from_nonembedded_module(V)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 for Gamma_0(
```

sage.modular.hecke.submodule.**is_HeckeSubmodule**(*x*)
Return True if x is of type HeckeSubmodule.

EXAMPLES:

```
sage: sage.modular.hecke.submodule.is_HeckeSubmodule(ModularForms(1, 12))
False
sage: sage.modular.hecke.submodule.is_HeckeSubmodule(CuspForms(1, 12))
True
```

CHAPTER

FOUR

ELEMENTS OF HECKE MODULES

AUTHORS:

- William Stein

class sage.modular.hecke.element.**HeckeModuleElement**(*parent*, *x=None*)
Bases: sage.structure.element.ModuleElement

Element of a Hecke module.

ambient_module()
Return the ambient Hecke module that contains this element.

EXAMPLES:

```
sage: BrandtModule(37)([0,1,-1]).ambient_module()
Brandt module of dimension 3 of level 37 of weight 2 over Rational Field
```

element()
Return underlying vector space element that defines this Hecke module element.

EXAMPLES:

```
sage: z = BrandtModule(37)([0,1,-1]).element(); z
(0, 1, -1)
sage: type(z)
<type 'sage.modules.vector_rational_dense.Vector_rational_dense'>
```

is_cuspidal()
Return True if this element is cuspidal.

EXAMPLES:

```
sage: M = ModularForms(2, 22); M.0.is_cuspidal()
True
sage: (M.0 + M.4).is_cuspidal()
False
sage: EllipticCurve('37a1').newform().is_cuspidal()
True
```

It works for modular symbols too:

```
sage: M = ModularSymbols(19,2)
sage: M.0.is_cuspidal()
False
sage: M.1.is_cuspidal()
True
```

is_eisenstein()

Return True if this element is Eisenstein. This makes sense for both modular forms and modular symbols.

EXAMPLES:

```
sage: CuspForms(2,8).0.is_eisenstein()
False
sage: M = ModularForms(2,8);(M.0  + M.1).is_eisenstein()
False
sage: M.1.is_eisenstein()
True
sage: ModularSymbols(19,4).0.is_eisenstein()
False
sage: EllipticCurve('37a1').newform().is_eisenstein()
False
```

is_new(*p=None*)

Return True if this element is p-new. If p is None, return True if the element is new.

EXAMPLE:

```
sage: CuspForms(22, 2).0.is_new(2)
False
sage: CuspForms(22, 2).0.is_new(11)
True
sage: CuspForms(22, 2).0.is_new()
False
```

is_old(*p=None*)

Return True if this element is p-old. If p is None, return True if the element is old.

EXAMPLE:

```
sage: CuspForms(22, 2).0.is_old(11)
False
sage: CuspForms(22, 2).0.is_old(2)
True
sage: CuspForms(22, 2).0.is_old()
True
sage: EisensteinForms(144, 2).1.is_old()  # long time (3s on sage.math, 2011)
False
sage: EisensteinForms(144, 2).1.is_old(2) # not implemented
False
```

sage.modular.hecke.element.**is_HeckeModuleElement**(*x*)

Return True if x is a Hecke module element, i.e., of type HeckeModuleElement.

EXAMPLES:

```
sage: sage.modular.hecke.all.is_HeckeModuleElement(0)
False
sage: sage.modular.hecke.all.is_HeckeModuleElement(BrandtModule(37)([1,2,3]))
True
```

CHAPTER

FIVE

HOM SPACES BETWEEN HECKE MODULES

class sage.modular.hecke.homspace.**HeckeModuleHomspace**(*X*, *Y*, *category=None*)

Bases: sage.categories.homset.HomsetWithBase

A space of homomorphisms between two objects in the category of Hecke modules over a given base ring.

sage.modular.hecke.homspace.**is_HeckeModuleHomspace**(*x*)

Return True if x is a space of homomorphisms in the category of Hecke modules.

EXAMPLES:

```
sage: M = ModularForms(Gamma0(7), 4)
sage: sage.modular.hecke.homspace.is_HeckeModuleHomspace(Hom(M, M))
True
sage: sage.modular.hecke.homspace.is_HeckeModuleHomspace(Hom(M, QQ))
False
```

CHAPTER

SIX

MORPHISMS OF HECKE MODULES

AUTHORS:

- William Stein

class sage.modular.hecke.morphism.**HeckeModuleMorphism**
Bases: sage.categories.morphism.Morphism

Abstract base class for morphisms of Hecke modules.

class sage.modular.hecke.morphism.**HeckeModuleMorphism_matrix**(*parent*, *A*, *name=''*)
Bases: sage.modules.matrix_morphism.MatrixMorphism, sage.modular.hecke.morphism.HeckeModul

Morphisms of Hecke modules when the morphism is given by a matrix.

Note that care is needed when composing morphisms, because morphisms in Sage act on the left, but their matrices act on the right (!). So if F: A -> B and G : B -> C are morphisms, the composition A -> C is G*F, but its matrix is F.matrix() * G.matrix().

EXAMPLE:

```
sage: A = ModularForms(1, 4)
sage: B = ModularForms(1, 16)
sage: C = ModularForms(1, 28)
sage: F = A.Hom(B)(matrix(QQ,1,2,srange(1, 3)))
sage: G = B.Hom(C)(matrix(QQ,2,3,srange(1, 7)))
sage: G * F
Hecke module morphism defined by the matrix
[ 9 12 15]
Domain: Modular Forms space of dimension 1 for Modular Group SL(2,Z) ...
Codomain: Modular Forms space of dimension 3 for Modular Group SL(2,Z) ...
sage: F * G
Traceback (most recent call last):
...
TypeError: Incompatible composition of morphisms: domain of left morphism must be codomain of ri
```

name(*new=None*)
Return the name of this operator, or set it to a new name.

EXAMPLES:

```
sage: M = ModularSymbols(6)
sage: t = M.Hom(M)(matrix(QQ,3,3,srange(9)), name="spam"); t
Hecke module morphism spam defined by ...
sage: t.name()
'spam'
sage: t.name("eggs"); t
Hecke module morphism eggs defined by ...
```

sage.modular.hecke.morphism.**is_HeckeModuleMorphism**(*x*)

Return True if x is of type HeckeModuleMorphism.

EXAMPLES:

```
sage: sage.modular.hecke.morphism.is_HeckeModuleMorphism(ModularSymbols(6).hecke_operator(7).hec
True
```

sage.modular.hecke.morphism.**is_HeckeModuleMorphism_matrix**(*x*)

EXAMPLES:

```
sage: sage.modular.hecke.morphism.is_HeckeModuleMorphism_matrix(ModularSymbols(6).hecke_operator
True
```

CHAPTER

SEVEN

DEGENERACY MAPS

class sage.modular.hecke.degenmap.**DegeneracyMap**(*matrix*, *domain*, *codomain*, *t*)

Bases: sage.modular.hecke.morphism.HeckeModuleMorphism_matrix

A degeneracy map between Hecke modules of different levels.

EXAMPLES: We construct a number of degeneracy maps:

```
sage: M = ModularSymbols(33)
sage: d = M.degeneracy_map(11)
sage: d
Hecke module morphism degeneracy map corresponding to f(q) |--> f(q) defined by the matrix
[ 1  0  0]
[ 0  0  1]
[ 0  0 -1]
[ 0  1 -1]
[ 0  0  1]
[ 0 -1  1]
[-1  0  0]
[-1  0  0]
[-1  0  0]
Domain: Modular Symbols space of dimension 9 for Gamma_0(33) of weight ...
Codomain: Modular Symbols space of dimension 3 for Gamma_0(11) of weight ...
sage: d.t()
1
sage: d = M.degeneracy_map(11,3)
sage: d.t()
3
```

The parameter d must be a divisor of the quotient of the two levels:

```
sage: d = M.degeneracy_map(11,2)
Traceback (most recent call last):
...
ValueError: The level of self (=33) must be a divisor or multiple of level (=11), and t (=2) mus
```

Degeneracy maps can also go from lower level to higher level:

```
sage: M.degeneracy_map(66,2)
Hecke module morphism degeneracy map corresponding to f(q) |--> f(q^2) defined by the matrix
[ 2  0  0  0  0  0  1  0  0  0  1 -1  0  0  0 -1  1  0  0  0  0  0  0  0 -1]
[ 0  0  1 -1  0 -1  1  0 -1  2  0  0  0 -1  0  0 -1  1  2 -2  0  0  0 -1  1]
[ 0  0  1  0  0  0  0  0  1  0  0  0  1  0  0  0 -1  1  0  0 -1  1  0  0  0]
[ 0  0  0  0  0  0  0  0  0  2 -1  0  0  1  0  0 -1  1  0  0  1  0 -1 -1  1]
[ 0 -1  0  0  1  0  0  0  0  0  0  1  0  0  1  1 -1  0  0 -1  0  0  0  0  0]
[ 0  0  0  0  0  0  0  1 -1  0  0  2 -1  0  0  1  0  0  0 -1  0 -1  1 -1  1]
[ 0  0  0  0  1 -1  0  1 -1  0  0  0  0  0 -1  2  0  0  0  0  1  0  1  0  0]
[ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  1  1  0  0  0]
```

```
[ 0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  1  1  1  0  0  0]
Domain: Modular Symbols space of dimension 9 for Gamma_0(33) of weight ...
Codomain: Modular Symbols space of dimension 25 for Gamma_0(66) of weight ...
```

t()

Return the divisor of the quotient of the two levels associated to the degeneracy map.

EXAMPLES:

```
sage: M = ModularSymbols(33)
sage: d = M.degeneracy_map(11,3)
sage: d.t()
3
sage: d = M.degeneracy_map(11,1)
sage: d.t()
1
```

CHAPTER

EIGHT

HECKE ALGEBRAS

In Sage a "Hecke algebra" always refers to an algebra of endomorphisms of some explicit module, rather than the abstract Hecke algebra of double cosets attached to a subgroup of the modular group.

We distinguish between "anemic Hecke algebras", which are algebras of Hecke operators whose indices do not divide some integer N (the level), and "full Hecke algebras", which include Hecke operators coprime to the level. Morphisms in the category of Hecke modules are not required to commute with the action of the full Hecke algebra, only with the anemic algebra.

sage.modular.hecke.algebra.**AnemicHeckeAlgebra**(*M*)

Return the anemic Hecke algebra associated to the Hecke module M. This checks whether or not the object already exists in memory, and if so, returns the existing object rather than a new one.

EXAMPLES:

```
sage: CuspForms(1, 12).anemic_hecke_algebra() # indirect doctest
Anemic Hecke algebra acting on Cuspidal subspace of dimension 1 of Modular Forms space of dimens
```

We test uniqueness:

```
sage: CuspForms(1, 12).anemic_hecke_algebra() is CuspForms(1, 12).anemic_hecke_algebra()
True
```

We can't ensure uniqueness when loading and saving objects from files, but we can ensure equality:

```
sage: CuspForms(1, 12).anemic_hecke_algebra() is loads(dumps(CuspForms(1, 12).anemic_hecke_algeb
False
sage: CuspForms(1, 12).anemic_hecke_algebra() == loads(dumps(CuspForms(1, 12).anemic_hecke_algeb
True
```

sage.modular.hecke.algebra.**HeckeAlgebra**(*M*)

Return the full Hecke algebra associated to the Hecke module M. This checks whether or not the object already exists in memory, and if so, returns the existing object rather than a new one.

EXAMPLES:

```
sage: CuspForms(1, 12).hecke_algebra() # indirect doctest
Full Hecke algebra acting on Cuspidal subspace of dimension 1 of Modular Forms space of dimensic
```

We test uniqueness:

```
sage: CuspForms(1, 12).hecke_algebra() is CuspForms(1, 12).hecke_algebra()
True
```

We can't ensure uniqueness when loading and saving objects from files, but we can ensure equality:

```
sage: CuspForms(1, 12).hecke_algebra() is loads(dumps(CuspForms(1, 12).hecke_algebra()))
False
```

```
sage: CuspForms(1, 12).hecke_algebra() == loads(dumps(CuspForms(1, 12).hecke_algebra()))
True
```

class sage.modular.hecke.algebra.**HeckeAlgebra_anemic**(*M*)
Bases: sage.modular.hecke.algebra.HeckeAlgebra_base

An anemic Hecke algebra, generated by Hecke operators with index coprime to the level.

gens()
Return a generator over all Hecke operator T_n for $n = 1, 2, 3, \ldots$, with n coprime to the level. This is an infinite sequence.

EXAMPLES:

```
sage: T = ModularSymbols(12,2).anemic_hecke_algebra()
sage: g = T.gens()
sage: next(g)
Hecke operator T_1 on Modular Symbols space of dimension 5 for Gamma_0(12) of weight 2 with
sage: next(g)
Hecke operator T_5 on Modular Symbols space of dimension 5 for Gamma_0(12) of weight 2 with
```

hecke_operator(*n*)
Return the n-th Hecke operator, for n any positive integer coprime to the level.

EXAMPLES:

```
sage: T = ModularSymbols(Gamma1(5),3).anemic_hecke_algebra()
sage: T.hecke_operator(2)
Hecke operator T_2 on Modular Symbols space of dimension 4 for Gamma_1(5) of weight 3 with s
sage: T.hecke_operator(5)
Traceback (most recent call last):
...
IndexError: Hecke operator T_5 not defined in the anemic Hecke algebra
```

is_anemic()
Return True, since this the anemic Hecke algebra.

EXAMPLES:

```
sage: H = CuspForms(3, 12).anemic_hecke_algebra()
sage: H.is_anemic()
True
```

class sage.modular.hecke.algebra.**HeckeAlgebra_base**(*M*)
Bases: sage.rings.ring.CommutativeAlgebra

Base class for algebras of Hecke operators on a fixed Hecke module.

basis()
Return a basis for this Hecke algebra as a free module over its base ring.

EXAMPLE:

```
sage: ModularSymbols(Gamma1(3), 3).hecke_algebra().basis()
[Hecke operator on Modular Symbols space of dimension 2 for Gamma_1(3) of weight 3 with sign
[1 0]
[0 1],
Hecke operator on Modular Symbols space of dimension 2 for Gamma_1(3) of weight 3 with sign
[0 0]
[0 2]]
```

diamond_bracket_matrix(*d*)

Return the matrix of the diamond bracket operator $\langle d \rangle$.

EXAMPLE:

```
sage: T = ModularSymbols(Gamma1(7), 4).hecke_algebra()
sage: T.diamond_bracket_matrix(3)
[    0     0     1     0     0     0     0     0     0     0     0     0]
[    1     0     0     0     0     0     0     0     0     0     0     0]
[    0     1     0     0     0     0     0     0     0     0     0     0]
[    0     0     0 -11/9  -4/9     1   2/3   7/9   2/9   7/9  -5/9  -2/9]
[    0     0     0  58/9  17/9    -5 -10/3   4/9   5/9 -50/9  37/9  13/9]
[    0     0     0 -22/9  -8/9     2   4/3   5/9   4/9  14/9 -10/9  -4/9]
[    0     0     0  44/9  16/9    -4  -8/3   8/9   1/9 -28/9  20/9   8/9]
[    0     0     0     0     0     0     0     0     0     0     1     0]
[    0     0     0     0     0     0     0     0     0     0     0     1]
[    0     0     0     1     0     0     0     0     0     0     0     0]
[    0     0     0     2     0    -1     0     0     0     0     0     0]
[    0     0     0    -4     0     4     1     0     0     0     0     0]
```

diamond_bracket_operator(*d*)

Return the diamond bracket operator $\langle d \rangle$.

EXAMPLE:

```
sage: T = ModularSymbols(Gamma1(7), 4).hecke_algebra()
sage: T.diamond_bracket_operator(3)
Diamond bracket operator <3> on Modular Symbols space of dimension 12 for Gamma_1(7) of weig
```

discriminant()

Return the discriminant of this Hecke algebra, i.e. the determinant of the matrix $\mathrm{Tr}(x_i x_j)$ where $x_1, \ldots, x_d$ is a basis for self, and $\mathrm{Tr}(x)$ signifies the trace (in the sense of linear algebra) of left multiplication by x on the algebra (*not* the trace of the operator x acting on the underlying Hecke module!). For further discussion and conjectures see Calegari + Stein, *Conjectures about discriminants of Hecke algebras of prime level*, Springer LNCS 3076.

EXAMPLE:

```
sage: BrandtModule(3, 4).hecke_algebra().discriminant()
1
sage: ModularSymbols(65, sign=1).cuspidal_submodule().hecke_algebra().discriminant()
6144
sage: ModularSymbols(1,4,sign=1).cuspidal_submodule().hecke_algebra().discriminant()
1
sage: H = CuspForms(1, 24).hecke_algebra()
sage: H.discriminant()
83041344
```

gen(*n*)

Return the n-th Hecke operator.

EXAMPLES:

```
sage: T = ModularSymbols(11).hecke_algebra()
sage: T.gen(2)
Hecke operator T_2 on Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with
```

gens()

Return a generator over all Hecke operator T_n for $n = 1, 2, 3, \ldots$. This is infinite.

EXAMPLES:

```
sage: T = ModularSymbols(1,12).hecke_algebra()
sage: g = T.gens()
sage: next(g)
Hecke operator T_1 on Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with
sage: next(g)
Hecke operator T_2 on Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with
```

hecke_matrix(*n*, **args*, ***kwds*)

Return the matrix of the n-th Hecke operator T_n.

EXAMPLES:

```
sage: T = ModularSymbols(1,12).hecke_algebra()
sage: T.hecke_matrix(2)
[ -24    0    0]
[   0  -24    0]
[4860    0 2049]
```

hecke_operator(*n*)

Return the n-th Hecke operator T_n.

EXAMPLES:

```
sage: T = ModularSymbols(1,12).hecke_algebra()
sage: T.hecke_operator(2)
Hecke operator T_2 on Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with
```

is_noetherian()

Return True if this Hecke algebra is Noetherian as a ring. This is true if and only if the base ring is Noetherian.

EXAMPLES:

```
sage: CuspForms(1, 12).anemic_hecke_algebra().is_noetherian()
True
```

level()

Return the level of this Hecke algebra, which is (by definition) the level of the Hecke module on which it acts.

EXAMPLE:

```
sage: ModularSymbols(37).hecke_algebra().level()
37
```

matrix_space()

Return the underlying matrix space of this module.

EXAMPLES:

```
sage: CuspForms(3, 24, base_ring=Qp(5)).anemic_hecke_algebra().matrix_space()
Full MatrixSpace of 7 by 7 dense matrices over 5-adic Field with capped relative precision 2
```

module()

The Hecke module on which this algebra is acting.

EXAMPLES:

```
sage: T = ModularSymbols(1,12).hecke_algebra()
sage: T.module()
Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with sign 0 over Rational F
```

ngens()
The size of the set of generators returned by gens(), which is clearly infinity. (This is not necessarily a minimal set of generators.)

EXAMPLES:

```
sage: CuspForms(1, 12).anemic_hecke_algebra().ngens()
+Infinity
```

rank()
The rank of this Hecke algebra as a module over its base ring. Not implemented at present.

EXAMPLE:

```
sage: ModularSymbols(Gamma1(3), 3).hecke_algebra().rank()
Traceback (most recent call last):
...
NotImplementedError
```

class sage.modular.hecke.algebra.**HeckeAlgebra_full**(*M*)
Bases: sage.modular.hecke.algebra.HeckeAlgebra_base

A full Hecke algebra (including the operators T_n where n is not assumed to be coprime to the level).

anemic_subalgebra()
The subalgebra of self generated by the Hecke operators of index coprime to the level.

EXAMPLE:

```
sage: H = CuspForms(3, 12).hecke_algebra()
sage: H.anemic_subalgebra()
Anemic Hecke algebra acting on Cuspidal subspace of dimension 3 of Modular Forms space of di
```

is_anemic()
Return False, since this the full Hecke algebra.

EXAMPLES:

```
sage: H = CuspForms(3, 12).hecke_algebra()
sage: H.is_anemic()
False
```

sage.modular.hecke.algebra.**is_HeckeAlgebra**(*x*)
Return True if x is of type HeckeAlgebra.

EXAMPLES:

```
sage: from sage.modular.hecke.algebra import is_HeckeAlgebra
sage: is_HeckeAlgebra(CuspForms(1, 12).anemic_hecke_algebra())
True
sage: is_HeckeAlgebra(ZZ)
False
```

CHAPTER

NINE

HECKE OPERATORS

class sage.modular.hecke.hecke_operator.**DiamondBracketOperator**(*parent*, *d*)
Bases: sage.modular.hecke.hecke_operator.HeckeAlgebraElement_matrix

The diamond bracket operator $\langle d \rangle$ for some $d \in \mathbf{Z}/N\mathbf{Z}$ (which need not be a unit, although if it is not, the operator will be zero).

class sage.modular.hecke.hecke_operator.**HeckeAlgebraElement**(*parent*)
Bases: sage.structure.element.AlgebraElement

Base class for elements of Hecke algebras.

apply_sparse(*x*)
Apply this Hecke operator to x, where we avoid computing the matrix of x if possible.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: T = M.hecke_operator(23)
sage: T.apply_sparse(M.gen(0))
24*(1,0) - 5*(1,9)
```

charpoly(*var='x'*)
Return the characteristic polynomial of this Hecke operator.

INPUT:

- var - string (default: 'x')

OUTPUT: a monic polynomial in the given variable.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(6),4)
sage: M.hecke_operator(2).charpoly('x')
x^6 - 14*x^5 + 29*x^4 + 172*x^3 - 124*x^2 - 320*x + 256
```

codomain()
The codomain of this operator. This is the Hecke module associated to the parent Hecke algebra.

EXAMPLE:

```
sage: R = ModularForms(Gamma0(7), 4).hecke_algebra()
sage: sage.modular.hecke.hecke_operator.HeckeAlgebraElement(R).codomain()
Modular Forms space of dimension 3 for Congruence Subgroup Gamma0(7) of weight 4 over Ration
```

decomposition()
Decompose the Hecke module under the action of this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(11)
sage: t2 = M.hecke_operator(2)
sage: t2.decomposition()
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for Gamma_0(
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(
]

sage: M = ModularSymbols(33, sign=1).new_submodule()
sage: T = M.hecke_operator(2)
sage: T.decomposition()
[
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 6 for Gamma_0(
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 6 for Gamma_0(
]
```

det()

Return the determinant of this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(23)
sage: T = M.hecke_operator(3)
sage: T.det()
100
```

domain()

The domain of this operator. This is the Hecke module associated to the parent Hecke algebra.

EXAMPLE:

```
sage: R = ModularForms(Gamma0(7), 4).hecke_algebra()
sage: sage.modular.hecke.hecke_operator.HeckeAlgebraElement(R).domain()
 Modular Forms space of dimension 3 for Congruence Subgroup Gamma0(7) of weight 4 over Ratio
```

fcp(*var='x'*)

Return the factorization of the characteristic polynomial of this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(23)
sage: T = M.hecke_operator(3)
sage: T.fcp('x')
(x - 4) * (x^2 - 5)^2
```

hecke_module_morphism()

Return the endomorphism of Hecke modules defined by the matrix attached to this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(Gamma1(13))
sage: t = M.hecke_operator(2)
sage: t
Hecke operator T_2 on Modular Symbols space of dimension 15 for Gamma_1(13) of weight 2 with
sage: t.hecke_module_morphism()
Hecke module morphism T_2 defined by the matrix
[ 2  1  0  0  0  0  0  0  0  0  0  0  0  0 -1]
[ 0  2  0  1  0  0  0 -1  0  0  0  0  0  0  0]
[ 0  0  2  0  0  1 -1  1  0 -1  0  1 -1  0  0]
[ 0  0  0  2  1  0  1  0  0  0  1 -1  0  0  0]
[ 0  0  1  0  2  0  0  0  0  1 -1  0  0  0  1]
```

```
[ 1  0  0  0  0  2  0  0  0  0  0  0  1  0  0]
[ 0  0  0  0  0  0  0  1 -1  1 -1  0 -1  1  1]
[ 0  0  0  0  0  0  0 -1  1  1  0  0 -1  1  0]
[ 0  0  0  0  0  0 -1 -1  0  1 -1 -1  1  0 -1]
[ 0  0  0  0  0  0 -2  0  2 -2  0  2 -2  1 -1]
[ 0  0  0  0  0  0  0  0  2 -1  1  0  0  1 -1]
[ 0  0  0  0  0  0 -1  1  2 -1  1  0 -2  2  0]
[ 0  0  0  0  0  0  0  0  1  1  0 -1  0  0  0]
[ 0  0  0  0  0  0 -1  1  1  0  1  1 -1  0  0]
[ 0  0  0  0  0  0  2  0  0  0  2 -1  0  1 -1]
Domain: Modular Symbols space of dimension 15 for Gamma_1(13) of weight ...
Codomain: Modular Symbols space of dimension 15 for Gamma_1(13) of weight ...
```

image()

Return the image of this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(23)
sage: T = M.hecke_operator(3)
sage: T.fcp('x')
(x - 4) * (x^2 - 5)^2
sage: T.image()
Modular Symbols subspace of dimension 5 of Modular Symbols space of dimension 5 for Gamma_0(
sage: (T-4).image()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5 for Gamma_0(
sage: (T**2-5).image()
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for Gamma_0(
```

kernel()

Return the kernel of this Hecke operator.

EXAMPLES:

```
sage: M = ModularSymbols(23)
sage: T = M.hecke_operator(3)
sage: T.fcp('x')
(x - 4) * (x^2 - 5)^2
sage: T.kernel()
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 5 for Gamma_0(
sage: (T-4).kernel()
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 5 for Gamma_0(
sage: (T**2-5).kernel()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 5 for Gamma_0(
```

trace()

Return the trace of this Hecke operator.

```
sage: M = ModularSymbols(1,12)
sage: T = M.hecke_operator(2)
sage: T.trace()
2001
```

class sage.modular.hecke.hecke_operator.**HeckeAlgebraElement_matrix**(*parent*, *A*)

Bases: sage.modular.hecke.hecke_operator.HeckeAlgebraElement

An element of the Hecke algebra represented by a matrix.

matrix()

Return the matrix that defines this Hecke algebra element.

EXAMPLES:

```
sage: M = ModularSymbols(1,12)
sage: T = M.hecke_operator(2).matrix_form()
sage: T.matrix()
[ -24    0    0]
[   0  -24    0]
[4860    0 2049]
```

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

Bases: sage.modular.hecke.hecke_operator.HeckeAlgebraElement

The Hecke operator T_n for some n (which need not be coprime to the level). The matrix is not computed until it is needed.

index()

Return the index of this Hecke operator, i.e., if this Hecke operator is T_n, return the int n.

EXAMPLES:

```
sage: T = ModularSymbols(11).hecke_operator(17)
sage: T.index()
17
```

matrix(**args*, ***kwds*)

Return the matrix underlying this Hecke operator.

EXAMPLES:

```
sage: T = ModularSymbols(11).hecke_operator(17)
sage: T.matrix()
[18  0 -4]
[ 0 -2  0]
[ 0  0 -2]
```

matrix_form()

Return the matrix form of this element of a Hecke algebra.

```
sage: T = ModularSymbols(11).hecke_operator(17)
sage: T.matrix_form()
Hecke operator on Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign
[18  0 -4]
[ 0 -2  0]
[ 0  0 -2]
```

sage.modular.hecke.hecke_operator.**is_HeckeAlgebraElement**(*x*)

Return True if x is of type HeckeAlgebraElement.

EXAMPLES:

```
sage: from sage.modular.hecke.hecke_operator import is_HeckeAlgebraElement
sage: M = ModularSymbols(Gamma0(7), 4)
sage: is_HeckeAlgebraElement(M.T(3))
True
sage: is_HeckeAlgebraElement(M.T(3) + M.T(5))
True
```

sage.modular.hecke.hecke_operator.**is_HeckeOperator**(*x*)

Return True if x is of type HeckeOperator.

EXAMPLES:

```
sage: from sage.modular.hecke.hecke_operator import is_HeckeOperator
sage: M = ModularSymbols(Gamma0(7), 4)
sage: is_HeckeOperator(M.T(3))
True
sage: is_HeckeOperator(M.T(3) + M.T(5))
False
```

CHAPTER

TEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

m

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

W

Z