CONTENTS

1 Constructors for certain modular abelian varieties 1
2 Base class for modular abelian varieties 3
3 Ambient Jacobian Abelian Variety 43
4 Finite subgroups of modular abelian varieties 47
5 Torsion subgroups of modular abelian varieties 55
6 Cuspidal subgroups of modular abelian varieties 61
7 Homology of modular abelian varieties 65
8 Spaces of homomorphisms between modular abelian varieties 73
9 Morphisms between modular abelian varieties, including Hecke operators acting on modular abelian varieties 81
10 Abelian varieties attached to newforms 89
11 $L$-series of modular abelian varieties 91
12 Indices and Tables 95
Python Module Index 97
Index 99
CONSTRUCTORS FOR CERTAIN MODULAR ABELIAN VARIETIES

AUTHORS:

- William Stein (2007-03)

```
sage.modular.abvar.constructor.AbelianVariety(X)
Create the abelian variety corresponding to the given defining data.

INPUT:

- X - an integer, string, newform, modsym space, congruence subgroup or tuple of congruence subgroups

OUTPUT: a modular abelian variety

EXAMPLES:

sage: AbelianVariety(Gamma0(37))
Abelian variety J0(37) of dimension 2
sage: AbelianVariety('37a')
Newform abelian subvariety 37a of dimension 1 of J0(37)

sage: AbelianVariety(ModularSymbols(37).cuspidal_submodule())
Abelian variety J0(37) of dimension 2
```

```
sage.modular.abvar.constructor.J0(N)
Return the Jacobian \( J_0(N) \) of the modular curve \( X_0(N) \).

EXAMPLES:

sage: J0(389)
Abelian variety J0(389) of dimension 32

The result is cached:

sage: J0(33) is J0(33)
True
```
sage.modular.abvar.constructor.J1(N)
Return the Jacobian \( J_1(N) \) of the modular curve \( X_1(N) \).

EXAMPLES:

```
sage: J1(389)
Abelian variety J1(389) of dimension 6112
```

sage.modular.abvar.constructor.JH(N,H)
Return the Jacobian \( J_H(N) \) of the modular curve \( X_H(N) \).

EXAMPLES:

```
sage: JH(389,[16])
Abelian variety JH(389,[16]) of dimension 64
```
CHAPTER
TWO

BASE CLASS FOR MODULAR ABELIAN VARIETIES

AUTHORS:

• William Stein (2007-03)

class sage.modular.abvar.abvar.ModularAbelianVariety(groups, lattice=None, base_field=Rational Field, is_simple=None, newform_level=None, isogeny_number=None, number=None, check=True)

Bases: sage.modular.abvar.abvar.ModularAbelianVariety_abstract

Create a modular abelian variety with given level and base field.

INPUT:

• groups - a tuple of congruence subgroups
• lattice - (default: \( \mathbb{Z}^n \)) a full lattice in \( \mathbb{Z}^n \), where \( n \) is the sum of the dimensions of the spaces of cuspidal modular symbols corresponding to each \( \Gamma \in \text{groups} \)
• base_field - a field (default: \( \mathbb{Q} \))

EXAMPLES:

```sage
J0(23)
Abelian variety J0(23) of dimension 2
```

```sage
J0(23).lattice()
Return the lattice that defines this abelian variety.
```

OUTPUT:

• lattice - a lattice embedded in the rational homology of the ambient product Jacobian

EXAMPLES:

```sage
A = (J0(11) \times J0(37))[1]; A
Simple abelian subvariety 37a(1,37) of dimension 1 of J0(11) \times J0(37)
```

```sage
type(A)
<class 'sage.modular.abvar.abvar.ModularAbelianVariety_with_category'>
```

```sage
A.lattice()
Free module of degree 6 and rank 2 over Integer Ring
```

Echelon basis matrix:
```
[ 0 0 1 -1 1 0]
[ 0 0 0 2 -1]
```
class sage.modular.abvar.abvar.ModularAbelianVariety_abstract(groups, base_field, is_simple=None, newform_level=None, isogeny_number=None, number=None, check=True)

Bases: sage.structure.parent_base.ParentWithBase

Abstract base class for modular abelian varieties.

INPUT:

• groups - a tuple of congruence subgroups
• base_field - a field
• is_simple - bool; whether or not self is simple
• newform_level - if self is isogenous to a newform abelian variety, returns the level of that abelian variety
• isogeny_number - which isogeny class the corresponding newform is in; this corresponds to the Cremona letter code
• number - the t number of the degeneracy map that this abelian variety is the image under
• check - whether to do some type checking on the defining data

EXAMPLES: One should not create an instance of this class, but we do so anyways here as an example:

```
sage: A = sage.modular.abvar.abvar.ModularAbelianVariety_abstract((Gamma0(37),), QQ)
sage: type(A)
<class 'sage.modular.abvar.abvar.ModularAbelianVariety_abstract_with_category'>
```

All hell breaks loose if you try to do anything with $A$:

```
sage: A
<repr('<sage.modular.abvar.abvar.ModularAbelianVariety_abstract_with_category at 0x...>') failed: NotImplementedError: BUG -- lattice method must be defined in derived class>  
```

All instances of this class are in the category of modular abelian varieties:

```
sage: A.category()
Category of modular abelian varieties over Rational Field
sage: J0(23).category()
Category of modular abelian varieties over Rational Field
```

ambient_morphism()

Return the morphism from self to the ambient variety. This is injective if self is natural a subvariety of the ambient product Jacobian.

OUTPUT: morphism

The output is cached.

EXAMPLES: We compute the ambient structure morphism for an abelian subvariety of $J_0(33)$:
\begin{verbatim}
  sage: A, B, C = J0(33)
  sage: phi = A.ambient_morphism()
  sage: phi.domain()
  Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
  sage: phi.codomain()
  Abelian variety J0(33) of dimension 3
  sage: phi.matrix()
  [ 1  1 -2  0  2 -1]
  [ 0  3 -2 -1  2  0]

  phi is of course injective

  sage: phi.kernel()
  (Finite subgroup with invariants [] over QQ of Simple abelian subvariety
  \rightarrow 11a(1,33) of dimension 1 of J0(33),
  Abelian subvariety of dimension 0 of J0(33))

  This is the same as the basis matrix for the lattice corresponding to self:

  sage: A.lattice()
  Free module of degree 6 and rank 2 over Integer Ring
  Echelon basis matrix:
  [ 1  1 -2  0  2 -1]
  [ 0  3 -2 -1  2  0]

  We compute a non-injective map to an ambient space:

  sage: Q, pi = J0(33)/A
  sage: phi = Q.ambient_morphism()
  sage: phi.matrix()
  [ 1  4  1  9 -1 -1]
  [ 0 15  0  0 30 -75]
  [ 0  0  5 10 -5 15]
  [ 0  0  0 15 -15 30]
  sage: phi.kernel()[0]
  Finite subgroup with invariants [5, 15, 15] over QQ of Abelian variety factor
  \rightarrow of dimension 2 of J0(33)

  ambient_variety()
  Return the ambient modular abelian variety that contains this abelian variety. The ambient variety is always
  a product of Jacobians of modular curves.

  OUTPUT: abelian variety

  EXAMPLES:

  sage: A = J0(33)[0]; A
  Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
  sage: A.ambient_variety()
  Abelian variety J0(33) of dimension 3

  base_extend(K)
  EXAMPLES:

  sage: A = J0(37); A
  Abelian variety J0(37) of dimension 2
  sage: A.base_extend(QQbar)
  Abelian variety J0(37) over Algebraic Field of dimension 2
\end{verbatim}
sage: A.base_extend(GF(7))
Abelian variety J0(37) over Finite Field of size 7 of dimension 2

base_field()
Synonym for self.base_ring().

EXAMPLES:

sage: J0(11).base_field()
Rational Field

change_ring(R)
Change the base ring of this modular abelian variety.

EXAMPLES:

sage: A = J0(23)
sage: A.change_ring(QQ)
Abelian variety J0(23) of dimension 2

complement(A=None)
Return a complement of this abelian variety.

INPUT:

• A - (default: None); if given, A must be an abelian variety that contains self, in which case the complement of self is taken inside A. Otherwise the complement is taken in the ambient product Jacobian.

OUTPUT: abelian variety

EXAMPLES:

sage: a,b,c = J0(33)
sage: (a+b).complement()
Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)
sage: (a+b).complement() == c
True
sage: a.complement(a+b)
Abelian subvariety of dimension 1 of J0(33)

conductor()
Return the conductor of this abelian variety.

EXAMPLES:

sage: A = J0(23)
sage: A.conductor().factor()
23^2
sage: A = J1(25)
sage: A.conductor().factor()
5^24
sage: A = J0(11^2); A.decomposition()
[
Simple abelian subvariety 11a(1,121) of dimension 1 of J0(121),
Simple abelian subvariety 11a(11,121) of dimension 1 of J0(121),
(continues on next page)
Simple abelian subvariety 121a(1,121) of dimension 1 of J0(121),
Simple abelian subvariety 121b(1,121) of dimension 1 of J0(121),
Simple abelian subvariety 121c(1,121) of dimension 1 of J0(121),
Simple abelian subvariety 121d(1,121) of dimension 1 of J0(121)
]  
sage: A.conductor().factor()
11^10

sage: A = J0(33)[0]; A
Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
sage: A.conductor()
11

sage: A.elliptic_curve().conductor()
11

cuspidal_subgroup() 
Return the cuspidal subgroup of this modular abelian variety. This is the subgroup generated by rational cusps.

EXAMPLES:

sage: J = J0(54)
sage: C = J.cuspidal_subgroup()
sage: C.gens()
[[[1/3, 0, 0, 0, 0, 1/3, 0, 2/3]], [[(0, 1/3, 0, 0, 0, 2/3, 0, 1/3)], [[(0, 0, 0, 0, 0, 0, 0, 1/3, 2/3)]]

sage: C.invariants()
[3, 3, 3, 3, 3, 9]
sage: J1(13).cuspidal_subgroup()
Finite subgroup with invariants [19, 19] over QQ of Abelian variety J1(13) of dimension 2

sage: A = J0(33)[0]
sage: A.cuspidal_subgroup()
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)

decomposition (simple=True, bound=None)
Return a sequence of abelian subvarieties of self that are all simple, have finite intersection and sum to self.

INPUT: simple- bool (default: True) if True, all factors are simple. If False, each factor returned is isogenous to a power of a simple and the simples in each factor are distinct.

• bound - int (default: None) if given, only use Hecke operators up to this bound when decomposing.
  This can give wrong answers, so use with caution!

EXAMPLES:

sage: m = ModularSymbols(11).cuspidal_submodule()
sage: dl = m.degeneracy_map(33,1).matrix(); d3=m.degeneracy_map(33,3).matrix()
sage: w = ModularSymbols(33).submodule((dl + d3).image(), check=False)
sage: A = w.abelian_variety(); A
Abelian subvariety of dimension 1 of J0(33)
sage: D = A.decomposition(); D
[ Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33) ]
We decompose a product of two Jacobians:

```
sage: (J0(33) * J0(11)).decomposition()
[Simple abelian subvariety 11a(1,11) of dimension 1 of J0(33) x J0(11),
 Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33) x J0(11),
 Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33) x J0(11)]
```

**degen_t (none_if_not_known=False)**

If this abelian variety is obtained via decomposition then it gets labeled with the newform label along with some information about degeneracy maps. In particular, the label ends in a pair \((t, N)\), where \(N\) is the ambient level and \(t\) is an integer that divides the quotient of \(N\) by the newform level. This function returns the tuple \((t, N)\), or raises a ValueError if self isn’t simple.

**Note:** It need not be the case that self is literally equal to the image of the newform abelian variety under the \(t^{th}\) degeneracy map. See the documentation for the label method for more details.

**INPUT:**

- none_if_not_known - (default: False) - if True, return None instead of attempting to compute the degen map’s \(t\), if it isn’t known. This None result is not cached.

**OUTPUT:** a pair (integer, integer)

**EXAMPLES:**

```
sage: D = J0(33).decomposition(); D
[Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33),
 Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33),
 Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)]
sage: D[0].degen_t()
(1, 33)
sage: D[1].degen_t()
```
\texttt{degeneracy_map}(M_{ls}, t_{ls})

Return the degeneracy map with domain \textit{self} and given level/parameter. If \textit{self.ambient_variety()} is a product of Jacobians (as opposed to a single Jacobian), then one can provide a list of new levels and parameters, corresponding to the ambient Jacobians in order. (See the examples below.)

\textbf{INPUT:}

• \texttt{M, t} - integers level and \(t\), or

• \texttt{Mlist, tlist} - if \textit{self} is in a nontrivial product ambient Jacobian, input consists of a list of levels and corresponding list of \(t\)'s.

\textbf{OUTPUT:} a degeneracy map

\textbf{EXAMPLES:} We make several degeneracy maps related to \(J_0(11)\) and \(J_0(33)\) and compute their matrices.

\begin{verbatim}
sage: d1 = J0(11).degeneracy_map(33, 1); d1
Degeneracy map from Abelian variety J0(11) of dimension 1 to Abelian variety
\rightarrow J0(33) of dimension 3 defined by [1]
sage: d1.matrix()
[ 0 -3 2 1 -2 0]
[ 1 -2 0 1 0 -1]
sage: d2 = J0(11).degeneracy_map(33, 3); d2
Degeneracy map from Abelian variety J0(11) of dimension 1 to Abelian variety
\rightarrow J0(33) of dimension 3 defined by [3]
sage: d2.matrix()
[-1 0 0 0 1 -2]
[-1 -1 1 -1 1 0]
sage: d3 = J0(33).degeneracy_map(11, 1); d3
Degeneracy map from Abelian variety J0(33) of dimension 3 to Abelian variety
\rightarrow J0(11) of dimension 1 defined by [1]
sage: d1.matrix() * d3.matrix()
[4 0]
[0 4]
\end{verbatim}

We verify that first mapping from level 11 to level 33, then back is multiplication by 4:

\begin{verbatim}
sage: d1.matrix() * d3.matrix()
[4 0]
[0 4]
\end{verbatim}

We compute a more complicated degeneracy map involving nontrivial product ambient Jacobians; note that this is just the block direct sum of the two matrices at the beginning of this example:

\begin{verbatim}
sage: d = (J0(11)*J0(11)).degeneracy_map([33,33], [1,3]); d
Degeneracy map from Abelian variety J0(11) x J0(11) of dimension 2 to Abelian
\rightarrow variety J0(33) x J0(33) of dimension 6 defined by [1, 3]
sage: d.matrix()
[ 0 -3 2 1 -2 0 0 0 0 0 0 0]
[ 1 -2 0 1 0 -1 0 0 0 0 0 0]
[ 0 0 0 0 0 0 -1 0 0 0 1 -2]
[ 0 0 0 0 0 0 -1 -1 1 -1 1 0]
\end{verbatim}
degree()  
Return the degree of this abelian variety, which is the dimension of the ambient Jacobian product.

EXAMPLES:
```
sage: A = J0(23)
sage: A.dimension()
2
```

dimension()  
Return the dimension of this abelian variety.

EXAMPLES:
```
sage: A = J0(23)
sage: A.dimension()
2
```

direct_product(other)  
Compute the direct product of self and other.

INPUT:
- `self`, `other` - modular abelian varieties

OUTPUT: abelian variety

EXAMPLES:
```
sage: J0(11).direct_product(J1(13))
Abelian variety J0(11) x J1(13) of dimension 3
sage: A = J0(33)[0].direct_product(J0(33)[1]); A  
Abelian subvariety of dimension 2 of J0(33) x J0(33)
sage: A.lattice()
Free module of degree 12 and rank 4 over Integer Ring
Echelon basis matrix:
[ 1 1 -2 0 2 -1 0 0 0 0 0 0]
[ 0 3 -2 -1 2 0 0 0 0 0 0 0]
[ 0 0 0 0 0 0 1 0 0 0 -1 2]
[ 0 0 0 0 0 0 0 1 -1 1 0 -2]
```

dual()  
Return the dual of this abelian variety.

OUTPUT:
- dual abelian variety
- morphism from self to dual
- covering morphism from J to dual

**Warning:** This is currently only implemented when self is an abelian subvariety of the ambient Jacobian product, and the complement of self in the ambient product Jacobian share no common factors. A more general implementation will require implementing computation of the intersection pairing on integral homology and the resulting Weil pairing on torsion.

EXAMPLES: We compute the dual of the elliptic curve newform abelian variety of level 33, and find the kernel of the modular map, which has structure \((\mathbb{Z}/3)^2\).
sage: A, B, C = J0(33)
sage: C
Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)
sage: C.d, f, pi = C.dual()
sage: f.matrix()
\[
\begin{bmatrix}
3 & 0 \\
0 & 3 \\
\end{bmatrix}
\]
sage: f.ker()
Finite subgroup with invariants [3, 3] over QQ of Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)

By a theorem the modular degree must thus be 3:

sage: E = EllipticCurve('33a')
sage: E.modular_degree()
3

Next we compute the dual of a 2-dimensional new simple abelian subvariety of \( J_0(43) \).

sage: A = AbelianVariety('43b'); A
Newform abelian subvariety 43b of dimension 2 of J0(43)
sage: A.d, f, pi = A.dual()

The kernel shows that the modular degree is 2:

sage: f.ker()
Finite subgroup with invariants [2, 2] over QQ of Newform abelian subvariety 43b of dimension 2 of J0(43)

Unfortunately, the dual is not implemented in general:

sage: A = J0(22)[0]; A
Simple abelian subvariety 11a(1,22) of dimension 1 of J0(22)
sage: A.dual()
Traceback (most recent call last):
...     NotImplementedError: dual not implemented unless complement shares no simple factors with self.

elliptic_curve()

Return an elliptic curve isogenous to self. If self is not dimension 1 with rational base ring, raise a ValueError.

The elliptic curve is found by looking it up in the CremonaDatabase. The CremonaDatabase contains all curves up to some large conductor. If a curve is not found in the CremonaDatabase, a RuntimeError will be raised. In practice, only the most committed users will see this RuntimeError.

OUTPUT: an elliptic curve isogenous to self.

EXAMPLES:
Elliptic curves are of dimension 1.

```sage
sage: J = J0(23)
sage: J.elliptic_curve()
Traceback (most recent call last):
  ... ValueError: self must be of dimension 1
```

This is only implemented for curves over QQ.

```sage
sage: J = J0(11).change_ring(CC)
sage: J.elliptic_curve()
Traceback (most recent call last):
  ... ValueError: base ring must be QQ
```

**endomorphism_ring**(category=None)

Return the endomorphism ring of self.

**OUTPUT:** b = self.sturm_bound()

**EXAMPLES:** We compute a few endomorphism rings:

```sage
sage: J0(11).endomorphism_ring()
Endomorphism ring of Abelian variety J0(11) of dimension 1
sage: J0(37).endomorphism_ring()
Endomorphism ring of Abelian variety J0(37) of dimension 2
sage: J0(33)[2].endomorphism_ring()
Endomorphism ring of Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)
```

No real computation is done:

```sage
sage: J1(123456).endomorphism_ring()
Endomorphism ring of Abelian variety J1(123456) of dimension 423185857
```

**finite_subgroup**(X, field_of_definition=None, check=True)

Return a finite subgroup of this modular abelian variety.

**INPUT:**

- X - list of elements of other finite subgroups of this modular abelian variety or elements that coerce into the rational homology (viewed as a rational vector space); also X could be a finite subgroup itself that is contained in this abelian variety.

- field_of_definition - (default: None) field over which this group is defined. If None try to figure out the best base field.

**OUTPUT:** a finite subgroup of a modular abelian variety

**EXAMPLES:**
```python
sage: J = J0(11)
sage: J.finite_subgroup([[1/5,0], [0,1/3]])
Finite subgroup with invariants [15] over QQbar of Abelian variety J0(11) of dimension 1
```
This method gives a way of changing the ambient abelian variety of a finite subgroup. This caused an issue in trac ticket #6392 but should be fixed now.

```python
sage: J = J0(33); C = J[0].cuspidal_subgroup(); C
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
sage: J.finite_subgroup([[0,0,0,0,0,1/6]])
Finite subgroup with invariants [6] over QQbar of Abelian variety J0(33) of dimension 3
sage: J.finite_subgroup(C)
Finite subgroup with invariants [5] over QQ of Abelian variety J0(33) of dimension 3
```

### `free_module()`

Synonym for `self.lattice()`.

**OUTPUT:** a free module over \( \mathbb{Z} \)

**EXAMPLES:**

```python
sage: J0(37).free_module()  
Ambient free module of rank 4 over the principal ideal domain Integer Ring
sage: J0(37)[0].free_module()  
Free module of degree 4 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1 -1 1 0]
[ 0 0 2 -1]
```

### `frobenius_polynomial(p, var='x')`

Computes the Frobenius polynomial at \( p \).

**INPUT:**

- \( p \) – prime number

**OUTPUT:**

a monic integral polynomial

**EXAMPLES:**

```python
sage: f = Newform('39b','a')
sage: A=AbelianVariety(f)
sage: A.frobenius_polynomial(5)
x^4 + 2*x^2 + 25
```
(continues on next page)
sage: J = J0(23)
sage: J.frobenius_polynomial(997)
x^4 + 20*x^3 + 1374*x^2 + 19940*x + 994009

sage: J = J0(33)
sage: J.frobenius_polynomial(7)
x^6 + 9*x^4 - 16*x^3 + 63*x^2 + 343

sage: J = J0(19)
sage: J.frobenius_polynomial(3, var='y')
y^2 + 2*y + 3

sage: J = J0(3); J
Abelian variety J0(3) of dimension 0
sage: J.frobenius_polynomial(11)
1

sage: A = J1(27)[1]; A
Simple abelian subvariety 27bG1(1,27) of dimension 12 of J1(27)
sage: A.groups()
(Congruence Subgroup Gamma0(27), Congruence Subgroup Gamma1(27))

sage: J = J1(33)
sage: J.frobenius_polynomial(11)
Traceback (most recent call last):
...  
ValueError: p must not divide the level of self
sage: J.frobenius_polynomial(4)
Traceback (most recent call last):
...  
ValueError: p must be prime

groups()
Return an ordered tuple of the congruence subgroups that the ambient product Jacobian is attached to.

Every modular abelian variety is a finite quotient of an abelian subvariety of a product of modular Jacobians \( \mathcal{J}_\Gamma \). This function returns a tuple containing the groups \( \Gamma \).

EXAMPLES:

sage: A = (J0(37) * J1(13))[0]; A
Simple abelian subvariety 13aG1(1,13) of dimension 2 of J0(37) x J1(13)
sage: A.groups()
(Congruence Subgroup Gamma0(37), Congruence Subgroup Gamma1(13))

hecke_operator(n)
Return the \( n \)th Hecke operator on the modular abelian variety, if this makes sense [[elaborate]]. Otherwise raise a ValueError.

EXAMPLES: We compute \( T_2 \) on \( J_0(37) \).
sage: t2 = J0(37).hecke_operator(2); t2
Hecke operator T_2 on Abelian variety J0(37) of dimension 2
sage: t2.charpoly().factor()
x * (x + 2)
sage: t2.index()
2

Note that there is no matrix associated to Hecke operators on modular abelian varieties. For a matrix, instead consider, e.g., the Hecke operator on integral or rational homology.

sage: t2.action_on_homology().matrix()
[-1 1 1 -1]
[ 1 -1 1 0]
[ 0 0 -2 1]
[ 0 0 0 0]

hecke_polynomial \( (n, \text{var}='x') \)
Return the characteristic polynomial of the \( n \)th Hecke operator \( T_n \) acting on self. Raises an ArithmeticError if self is not Hecke equivariant.

INPUT:
• n - integer \( \geq 1 \)
• var - string (default: ‘x’); valid variable name

EXAMPLES:

sage: J0(33).hecke_polynomial(2)
x^3 + 3*x^2 - 4
sage: f = J0(33).hecke_polynomial(2, 'y'); f
y^3 + 3*y^2 - 4
sage: f.parent()
Univariate Polynomial Ring in y over Rational Field
sage: J0(33)[2].hecke_polynomial(3)
x + 1
sage: J0(33)[0].hecke_polynomial(5)
x - 1
sage: J0(33)[0].hecke_polynomial(11)
x - 1
sage: J0(33)[0].hecke_polynomial(3)
Traceback (most recent call last):
...
ArithmeticError: subspace is not invariant under matrix

homology (base_ring=Integer Ring)
Return the homology of this modular abelian variety.

Warning: For efficiency reasons the basis of the integral homology need not be the same as the basis for the rational homology.

EXAMPLES:

sage: J0(389).homology(GF(7))
Homology with coefficients in Finite Field of size 7 of Abelian variety J0(389) of dimension 32

(continues on next page)
sage: J0(389).homology(QQ)
Rational Homology of Abelian variety J0(389) of dimension 32
sage: J0(389).homology(ZZ)
Integral Homology of Abelian variety J0(389) of dimension 32

in_same_ambient Variety(\texttt{other})
Return True if self and other are abelian subvarieties of the same ambient product Jacobian.

EXAMPLES:

sage: A, B, C = J0(33)
sage: A.in_same_ambient Variety(B)
True
sage: A.in_same_ambient Variety(J0(11))
False

integral_homology()
Return the integral homology of this modular abelian variety.

EXAMPLES:

sage: H = J0(43).integral_homology(); H
Integral Homology of Abelian variety J0(43) of dimension 3
sage: H.rank()
6
sage: H = J1(17).integral_homology(); H
Integral Homology of Abelian variety J1(17) of dimension 5
sage: H.rank()
10

If you just ask for the rank of the homology, no serious calculations are done, so the following is fast:

sage: H = J0(50000).integral_homology(); H
Integral Homology of Abelian variety J0(50000) of dimension 7351
sage: H.rank()
14702

A product:

sage: H = (J0(11) * J1(13)).integral_homology()
sage: H.hecke_operator(2)
Hecke operator T_2 on Integral Homology of Abelian variety J0(11) x J1(13) of dimension 3
sage: H.hecke_operator(2).matrix()
\begin{bmatrix}
-2 & 0 & 0 & 0 & 0 \\
0 & -2 & 0 & 0 & 0 \\
0 & 0 & -1 & -1 & -1 \\
0 & 0 & 1 & -2 & -1 \\
0 & 0 & 0 & 0 & -2 \\
0 & 0 & 0 & 0 & -1
\end{bmatrix}

intersection(\texttt{other})
Return the intersection of self and other inside a common ambient Jacobian product.

When \texttt{other} is a modular abelian variety, the output will be a tuple \((G, A)\), where \(G\) is a finite subgroup that surjects onto the component group and \(A\) is the identity component. So in particular, the intersection is the variety \(G+A\). Note that \(G\) is not chosen in any canonical way. When \texttt{other} is a finite group, the intersection will be returned as a finite group.
INPUT:

- **other** - a modular abelian variety or a finite group

OUTPUT: If other is a modular abelian variety:

- **G** - finite subgroup of self
- **A** - abelian variety (identity component of intersection)

If other is a finite group:

- **G** - a finite group

EXAMPLES: We intersect some abelian varieties with finite intersection.

```
sage: J = J0(37)
sage: J[0].intersection(J[1])
(Finite subgroup with invariants [2, 2] over QQ of Simple abelian subvariety ˓→37a(1,37) of dimension 1 of J0(37), Simple abelian subvariety of dimension 0 of J0(37))
```

```
sage: D = list(J0(65)); D
[Simple abelian subvariety 65a(1,65) of dimension 1 of J0(65), Simple abelian subvariety 65b(1,65) of dimension 2 of J0(65), Simple abelian subvariety 65c(1,65) of dimension 2 of J0(65)]
sage: D[0].intersection(D[1])
(Finite subgroup with invariants [2] over QQ of Simple abelian subvariety ˓→65a(1,65) of dimension 1 of J0(65), Simple abelian subvariety of dimension 0 of J0(65))
sage: (D[0]+D[1]).intersection(D[1]+D[2])
(Finite subgroup with invariants [2] over QQbar of Abelian subvariety of dimension 3 of J0(65), Abelian subvariety of dimension 2 of J0(65))
```

```
sage: J = J0(33)
sage: J[0].intersection(J[1])
(Finite subgroup with invariants [5] over QQ of Simple abelian subvariety ˓→11a(1,33) of dimension 1 of J0(33), Simple abelian subvariety of dimension 0 of J0(33))
```

```
sage: J = J0(67); D = J.decomposition(); D
[Simple abelian subvariety 67a(1,67) of dimension 1 of J0(67), Simple abelian subvariety 67b(1,67) of dimension 2 of J0(67), Simple abelian subvariety 67c(1,67) of dimension 2 of J0(67)]
sage: (D[0] + D[1]).intersection(D[1] + D[2])
(Finite subgroup with invariants [5, 10] over QQbar of Abelian subvariety of dimension 3 of J0(67), Abelian subvariety of dimension 2 of J0(67))
```

Next we intersect two abelian varieties with non-finite intersection:

```
sage: J = J0(67); D = J.decomposition(); D
[Simple abelian subvariety 67a(1,67) of dimension 1 of J0(67), Simple abelian subvariety 67b(1,67) of dimension 2 of J0(67), Simple abelian subvariety 67c(1,67) of dimension 2 of J0(67)]
sage: (D[0] + D[1]).intersection(D[1] + D[2])
(Finite subgroup with invariants [5, 10] over QQbar of Abelian subvariety of dimension 3 of J0(67), Abelian subvariety of dimension 2 of J0(67))
```

When the intersection is infinite, the output is \((G, A)\), where \(G\) surjects onto the component group. This choice of \(G\) is not canonical (see trac ticket #26189). In this following example, \(B\) is a subvariety of \(J\):

```
sage: d1 = J0(11).degeneracy_map(22, 1)
sage: d2 = J0(11).degeneracy_map(22, 2)
sage: B = (d1-d2).image()
sage: J = J0(22)
sage: J.intersection(B)
```

(continues on next page)
is_J0()

Return whether or not self is of the form J0(N).

OUTPUT: bool

EXAMPLES:

```python
sage: J0(23).is_J0()
True
sage: J1(11).is_J0()
False
sage: (J0(23) * J1(11)).is_J0()
False
sage: J0(37)[0].is_J0()
False
sage: (J0(23) * J0(21)).is_J0()
False
```

is_J1()

Return whether or not self is of the form J1(N).

OUTPUT: bool

EXAMPLES:

```python
sage: J1(23).is_J1()
True
sage: J0(23).is_J1()
False
sage: (J1(11) * J1(13)).is_J1()
False
sage: (J1(11) * J0(13)).is_J1()
False
sage: J1(23)[0].is_J1()
False
```

is_ambient()

Return True if self equals the ambient product Jacobian.

OUTPUT: bool

EXAMPLES:

```python
sage: A,B,C = J0(33)
sage: A.is_ambient()
False
sage: J0(33).is_ambient()
True
```
```
sage: (A+B).is_ambient()
False
sage: (A+B+C).is_ambient()
True
```

**is_hecke_stable()**

Return True if self is stable under the Hecke operators of its ambient Jacobian.

OUTPUT: bool

EXAMPLES:

```
sage: J0(11).is_hecke_stable()
True
sage: J0(33)[2].is_hecke_stable()
True
sage: J0(33)[0].is_hecke_stable()
False
sage: (J0(33)[0] + J0(33)[1]).is_hecke_stable()
True
```

**is_simple**(none_if_not_known=False)

Return whether or not this modular abelian variety is simple, i.e., has no proper nonzero abelian subvarieties.

INPUT:

- none_if_not_known - bool (default: False); if True then this function may return None instead of True or False if we don’t already know whether or not self is simple.

EXAMPLES:

```
sage: J0(5).is_simple(none_if_not_known=True) is None # this may fail if J0(5) comes up elsewhere...
True
sage: J0(33).is_simple()
False
sage: J0(33).is_simple(none_if_not_known=True)
False
sage: J0(33)[1].is_simple()
True
sage: J1(17).is_simple()
False
```

**is_subvariety**(other)

Return True if self is a subvariety of other as they sit in a common ambient modular Jacobian. In particular, this function will only return True if self and other have exactly the same ambient Jacobians.

EXAMPLES:

```
sage: J = J0(37); J
Abelian variety J0(37) of dimension 2
sage: A = J[0]; A
Simple abelian subvariety 37a(1,37) of dimension 1 of J0(37)
sage: A.is_subvariety(A)
True
sage: A.is_subvariety(J)
True
```
is_subvariety_of_ambient_jacobian()

Return True if self is (presented as) a subvariety of the ambient product Jacobian.

Every abelian variety in Sage is a quotient of a subvariety of an ambient Jacobian product by a finite subgroup.

EXAMPLES:

```python
sage: J0(33).is_subvariety_of_ambient_jacobian()
True
sage: A = J0(33)[0]; A
Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
sage: A.is_subvariety_of_ambient_jacobian()
True
sage: B, phi = A / A.torsion_subgroup(2)
sage: B
Abelian variety factor of dimension 1 of J0(33)
sage: phi.matrix()
[2 0]
[0 2]
sage: B.is_subvariety_of_ambient_jacobian()
False
```

isogeny_number(none_if_not_known=False)

Return the number (starting at 0) of the isogeny class of new simple abelian varieties that self is in. If self is not simple, raises a ValueError exception.

INPUT:

- none_if_not_known - bool (default: False); if True then this function may return None instead of True or False if we don’t already know the isogeny number of self.

EXAMPLES: We test the none_if_not_known flag first:

```python
sage: J0(33).isogeny_number(none_if_not_known=True) is None
True
```

Of course, \(J_0(33)\) is not simple, so this function raises a ValueError:

```python
sage: J0(33).isogeny_number()
Traceback (most recent call last):
  ... ValueError: self must be simple
```

Each simple factor has isogeny number 1, since that’s the number at which the factor is new.

```python
sage: J0(33)[1].isogeny_number()
0
sage: J0(33)[2].isogeny_number()
0
```

Next consider \(J_0(37)\) where there are two distinct newform factors:

```python
sage: J0(37)[1].isogeny_number()
1
```

label()

Return the label associated to this modular abelian variety.

The format of the label is [level][isogeny class][group][t, ambient level]
If this abelian variety $B$ has the above label, this implies only that $B$ is isogenous to the newform abelian variety $A_f$ associated to the newform with label [level][isogeny class][group]. The [group] is empty for $\Gamma_0(N)$, is $G1$ for $\Gamma_1(N)$ and is $GH[...]$ for $\Gamma_H(N)$.

**Warning:** The sum of $\delta_s(A_f)$ for all $s \mid t$ contains $A$, but no sum for a proper divisor of $t$ contains $A$. It need *not* be the case that $B$ is equal to $\delta_t(A_f)$!!!

**OUTPUT:** string

**EXAMPLES:**

```sage
sage: J0(11).label()
'11a(1,11)'
sage: J0(11)[0].label()
'11a(1,11)'
sage: J0(33)[2].label()
'33a(1,33)'
sage: J0(22).label()
Traceback (most recent call last):
  ... ValueError: self must be simple
```

We illustrate that self need not equal $\delta_t(A_f)$:

```sage
sage: J = J0(11); phi = J.degeneracy_map(33, 1) + J.degeneracy_map(33,3)
sage: B = phi.image(); B
Abelian subvariety of dimension 1 of J0(33)
sage: B.decomposition()
[
  Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33)
]
sage: C = J.degeneracy_map(33,3).image(); C
Abelian subvariety of dimension 1 of J0(33)
sage: C == B
False
```

**lattice()**

Return lattice in ambient cuspidal modular symbols product that defines this modular abelian variety.

This must be defined in each derived class.

**OUTPUT:** a free module over $\mathbb{Z}$

**EXAMPLES:**

```sage
sage: A = sage.modular.abvar.abvar.ModularAbelianVariety_abstract((Gamma0(37), →), QQ)
sage: A
<repr('<sage.modular.abvar.abvar.ModularAbelianVariety_abstract_with_category
→at 0x...'>) failed: NotImplementedError: BUG -- lattice method must be
→defined in derived class>
```

**level()**

Return the level of this modular abelian variety, which is an integer $N$ (usually minimal) such that this modular abelian variety is a quotient of $J_1(N)$. In the case that the ambient variety of self is a product of Jacobians, return the LCM of their levels.

**EXAMPLES:**


lseries()

Return the complex $L$-series of this modular abelian variety.

EXAMPLES:

```
sage: A = J0(37)
sage: A.lseries()
Complex L-series attached to Abelian variety J0(37) of dimension 2
```

modular_degree()

Return the modular degree of this abelian variety, which is the square root of the degree of the modular kernel.

EXAMPLES:

```
sage: A = AbelianVariety('37a')
sage: A.modular_degree()
2
```

modular_kernel()

Return the modular kernel of this abelian variety, which is the kernel of the canonical polarization of self.

EXAMPLES:

```
sage: A = AbelianVariety('33a'); A
Newform abelian subvariety 33a of dimension 1 of J0(33)
sage: A.modular_kernel()
Finite subgroup with invariants [3, 3] over QQ of Newform abelian subvariety 33a of dimension 1 of J0(33)
```

newform(names=None)

Return the newform $f$ such that this abelian variety is isogenous to the newform abelian variety $A_f$. If this abelian variety is not simple, raise a ValueError.

INPUT:

- names – (default: None) If the newform has coefficients in a number field, then a generator name must be specified.

OUTPUT: A newform $f$ so that self is isogenous to $A_f$.

EXAMPLES:

```
sage: J0(11).newform()
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
sage: f = J0(23).newform(names='a')
sage: AbelianVariety(f) == J0(23)
True
sage: J = J0(33)
sage: [s.newform('a') for s in J.decomposition()]
(continues on next page)
```
The following fails since \( J_0(33) \) is not simple:

```
sage: J0(33).newform()
Traceback (most recent call last):
...  
ValueError: self must be simple
```

### newform_decomposition \((names=None)\)

Return the newforms of the simple subvarieties in the decomposition of self as a product of simple subvarieties, up to isogeny.

**OUTPUT:**
- an array of newforms

**EXAMPLES:**

```
sage: J = J1(11) * J0(23)
sage: J.newform_decomposition('a')
[q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),  
q + a0*q^2 + (-2*a0 - 1)*q^3 + (-a0 - 1)*q^4 + 2*a0*q^5 + O(q^6)]
```

### newform_label()

Return the label \([\text{level}][\text{isogeny class}][\text{group}]\) of the newform \( f \) such that this abelian variety is isogenous to the newform abelian variety \( A_f \). If this abelian variety is not simple, raise a \( \text{ValueError} \).

**OUTPUT:** string

**EXAMPLES:**

```
sage: J0(11).newform_label()
'11a'
sage: J0(33)[2].newform_label()
'33a'
```

The following fails since \( J_0(33) \) is not simple:

```
sage: J0(33).newform_label()
Traceback (most recent call last):  
...  
ValueError: self must be simple
```

### newform_level \((\text{none\_if\_not\_known}=\text{False})\)

Write self as a product (up to isogeny) of newform abelian varieties \( A_f \). Then this function return the least common multiple of the levels of the newforms \( f \), along with the corresponding group or list of groups (the groups do not appear with multiplicity).

**INPUT:**
- none_if_not_known - (default: False) if True, return None instead of attempting to compute the newform level, if it’s not already known. This None result is not cached.

**OUTPUT:** integer group or list of distinct groups

**EXAMPLES:**

```
Here there are multiple groups since there are in fact multiple newforms:

```
sage: (J0(11) * J1(13)).newform_level()
(143, [Congruence Subgroup Gamma0(11), Congruence Subgroup Gamma(13)])
```

### number_of_rational_points()

Return the number of rational points of this modular abelian variety.

It is not always possible to compute the order of the torsion subgroup. The BSD conjecture is assumed to compute the algebraic rank.

**OUTPUT:** a positive integer or infinity

**EXAMPLES:**

```
sage: J0(23).number_of_rational_points()
11
sage: J0(29).number_of_rational_points()
7
sage: J0(37).number_of_rational_points()
+Infinity
sage: J0(12); J0(12).number_of_rational_points()
Abelian variety J0(12) of dimension 0
1
sage: J1(17).number_of_rational_points()
584
sage: J1(16).number_of_rational_points()
Traceback (most recent call last):
...
TypeError: Unable to compute order of torsion subgroup (it is in [1, 2, 4, ...
→5, 10, 20])
```

### padic_lseries(p)

Return the $p$-adic $L$-series of this modular abelian variety.

**EXAMPLES:**

```
sage: A = J0(37)
sage: A.padic_lseries(7)
7-adic L-series attached to Abelian variety J0(37) of dimension 2
```

### project_to_factor(n)

If self is an ambient product of Jacobians, return a projection from self to the nth such Jacobian.

**EXAMPLES:**

```
sage: J = J0(33)
sage: J.project_to_factor(0)
Abelian variety endomorphism of Abelian variety J0(33) of dimension 3
```
```python
sage: J = J0(33) * J0(37) * J0(11)
sage: J.project_to_factor(2)
Abelian variety morphism:
  From: Abelian variety J0(33) x J0(37) x J0(11) of dimension 6
  To:    Abelian variety J0(11) of dimension 1
sage: J.project_to_factor(2).matrix()
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 0]
[0 1]
[0 0]
projection (A, check=True)
Given an abelian subvariety A of self, return a projection morphism from self to A. Note that this morphism need not be unique.

INPUT:
  • A - an abelian variety

OUTPUT: a morphism

EXAMPLES:
```}

We project onto a factor in a product of two Jacobians:
```python
sage: a, b, c = J0(33)
sage: pi = J0(33).projection(a); pi.matrix()
[ 3 -2]
[ 3 -2]
[ 5 5]
[ 5 0]
[ 1 1]
sage: pi = (a+b).projection(a); pi.matrix()
[ 0 0]
[-3 2]
[-4 1]
[-1 -1]
sage: pi = a.projection(a); pi.matrix()
[1 0]
[0 1]
```
qbar_torsion_subgroup()

Return the group of all points of finite order in the algebraic closure of this abelian variety.

EXAMPLES:

```
sage: T = J0(33).qbar_torsion_subgroup(); T
Group of all torsion points in QQbar on Abelian variety J0(33) of dimension 3
```

The field of definition is the same as the base field of the abelian variety.

```
sage: T.field_of_definition()
Rational Field
```

On the other hand, T is a module over \(\mathbb{Z}\).

```
sage: T.base_ring()
Integer Ring
```

quotient(other, **kwds)

Compute the quotient of self and other, where other is either an abelian subvariety of self or a finite subgroup of self.

INPUT:

- other - a finite subgroup or subvariety
- further named arguments, that are currently ignored.

OUTPUT: a pair (A, phi) with phi the quotient map from self to A

EXAMPLES: We quotient \(J_0(33)\) out by an abelian subvariety:

```
sage: Q, f = J0(33).quotient(J0(33)[0])
sage: Q
Abelian variety factor of dimension 2 of J0(33)
sage: f
Abelian variety morphism:
  From: Abelian variety J0(33) of dimension 3
  To:  Abelian variety factor of dimension 2 of J0(33)
```

We quotient \(J_0(33)\) by the cuspidal subgroup:

```
sage: C = J0(33).cuspidal_subgroup()
sage: Q, f = J0(33).quotient(C)
sage: Q
Abelian variety factor of dimension 3 of J0(33)
sage: f.kernel()[0]
Finite subgroup with invariants [10, 10] over QQ of Abelian variety J0(33) of dimension 3
```

(continues on next page)
sage: C
Finite subgroup with invariants [10, 10] over QQ of Abelian variety $J_0(33)$ of dimension 3
sage: J0(11).direct_product(J1(13))
Abelian variety $J_0(11) \times J_1(13)$ of dimension 3

\textbf{rank()}

Return the rank of the underlying lattice of self.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: J = J0(33)
sage: J.rank()
6
sage: J[1]
Simple abelian subvariety $11a(3,33)$ of dimension 1 of $J_0(33)$
sage: (J[1] * J[1]).rank()
4
\end{verbatim}

\textbf{rational_cusp_subgroup()}

Return the subgroup of this modular abelian variety generated by rational cusps.

This is a subgroup of the group of rational points in the cuspidal subgroup.

\textbf{Warning:} This is only currently implemented for $\Gamma_0(N)$.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: J = J0(54)
sage: CQ = J.rational_cusp_subgroup(); CQ
Finite subgroup with invariants [3, 3, 9] over QQ of Abelian variety $J_0(54)$ of dimension 4
sage: CQ.gens()
[[[1/3, 0, 0, 1/3, 2/3, 1/3, 0, 1/3]], [[0, 0, 1/9, 1/9, 7/9, 7/9, 1/9, 8/9]],
[[0, 0, 0, 0, 0, 0, 1/3, 2/3]]]
sage: factor(CQ.order())
3^4
sage: CQ.invariants()
[3, 3, 9]
\end{verbatim}

In this example the rational cuspidal subgroup and the cuspidal subgroup differ by a lot.

\begin{verbatim}
sage: J = J0(49)
sage: J.cuspidal_subgroup()
Finite subgroup with invariants [2, 14] over QQ of Abelian variety $J_0(49)$ of dimension 1
sage: J.rational_cusp_subgroup()
Finite subgroup with invariants [2] over QQ of Abelian variety $J_0(49)$ of dimension 1
\end{verbatim}

Note that computation of the rational cusp subgroup isn’t implemented for $\Gamma_1$. 

\begin{verbatim}
sage: J = J1(13)
sage: J.cuspidal_subgroup()
Finite subgroup with invariants [19, 19] over QQ of Abelian variety $J_1(13)$ of dimension 2
\end{verbatim}
sage: J.rational_cusp_subgroup()
Traceback (most recent call last):
...
NotImplementedError: computation of rational cusps only implemented in Gamma0
˓→case.

**rational_cuspidal_subgroup()**

Return the rational subgroup of the cuspidal subgroup of this modular abelian variety.

This is a subgroup of the group of rational points in the cuspidal subgroup.

**Warning:** This is only currently implemented for $\Gamma_0(N)$.

**EXAMPLES:**

```python
sage: J = J0(54)
sage: CQ = J.rational_cuspidal_subgroup(); CQ
Finite subgroup with invariants [3, 3, 9] over QQ of Abelian variety J0(54)
˓→of dimension 4
sage: CQ.gens()
[[(1/3, 0, 0, 1/3, 2/3, 1/3, 0, 1/3)], [(0, 0, 1/9, 1/9, 7/9, 7/9, 1/9, 8/9)],
˓→[(0, 0, 0, 0, 0, 0, 1/3, 2/3)]
sage: factor(CQ.order())
3^4
sage: CQ.invariants()
[3, 3, 9]
```

In this example the rational cuspidal subgroup and the cuspidal subgroup differ by a lot.

```python
sage: J = J0(49)
sage: J.cuspidal_subgroup()
Finite subgroup with invariants [2, 14] over QQ of Abelian variety J0(49) of
˓→dimension 1
sage: J.rational_cuspidal_subgroup()
Finite subgroup with invariants [2] over QQ of Abelian variety J0(49) of
˓→dimension 1
```

Note that computation of the rational cusp subgroup isn’t implemented for $\Gamma_1$.

```python
sage: J = J1(13)
sage: J.cuspidal_subgroup()
Finite subgroup with invariants [19, 19] over QQ of Abelian variety J1(13) of
˓→dimension 2
sage: J.rational_cuspidal_subgroup()
Traceback (most recent call last):
...
NotImplementedError: only implemented when group is Gamma0
```

**rational_homology()**

Return the rational homology of this modular abelian variety.

**EXAMPLES:**

```python
sage: H = J0(37).rational_homology(); H
Rational Homology of Abelian variety J0(37) of dimension 2
```
sage: H.rank()
4
sage: H.base_ring()
Rational Field
sage: H = J1(17).rational_homology(); H
Rational Homology of Abelian variety J1(17) of dimension 5
sage: H.rank()
10
sage: H.base_ring()
Rational Field

**rational_torsion_order**(proof=True)

Return the order of the rational torsion subgroup of this modular abelian variety.

This function is really an alias for order() See the docstring there for a more in-depth reference and more interesting examples.

**INPUT:**

* proof – a boolean (default: True)

**OUTPUT:**

The order of the rational torsion subgroup of this modular abelian variety.

**EXAMPLES:**

sage: J0(11).rational_torsion_subgroup().order()
5
sage: J0(11).rational_torsion_order()
5

**rational_torsion_subgroup()**

Return the maximal torsion subgroup of self defined over QQ.

**EXAMPLES:**

sage: J = J0(33)
sage: A = J.new_subvariety()
sage: A
Abelian subvariety of dimension 1 of J0(33)
sage: t = A.rational_torsion_subgroup(); t
Torsion subgroup of Abelian subvariety of dimension 1 of J0(33)
sage: t.multiple_of_order()
4
sage: t.divisor_of_order()
4
sage: t.order()
4
sage: t.gens()
[[[1/2, 0, 0, -1/2, 0, 0]], [[0, 0, 1/2, 0, 1/2, -1/2]]]

**shimura_subgroup()**

Return the Shimura subgroup of this modular abelian variety. This is the kernel of $J_0(N) \to J_1(N)$ under the natural map. Here we compute the Shimura subgroup as the kernel of $J_0(N) \to J_0(Np)$ where the map is the difference between the two degeneracy maps.

**EXAMPLES:**
```python
sage: J=J0(11)
sage: J.shimura_subgroup()
Finite subgroup with invariants [5] over QQ of Abelian variety J0(11) of
dimension 1

sage: J=J0(17)
sage: G=J.cuspidal_subgroup(); G
Finite subgroup with invariants [4] over QQ of Abelian variety J0(17) of
dimension 1

sage: S=J.shimura_subgroup(); S
Finite subgroup with invariants [4] over QQ of Abelian variety J0(17) of
dimension 1

sage: G.intersection(S)
Finite subgroup with invariants [2] over QQ of Abelian variety J0(17) of
dimension 1

sage: J=J0(33)
sage: A=J.decomposition()[0]
sage: A.shimura_subgroup()
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety
11a(1,33) of dimension 1 of J0(33)

sage: J.shimura_subgroup()
Finite subgroup with invariants [10] over QQ of Abelian variety J0(33) of
dimension 3
```

**sturm_bound()**

Return a bound $B$ such that all Hecke operators $T_n$ for $n \leq B$ generate the Hecke algebra.

**OUTPUT:** integer

**EXAMPLES:**

```python
sage: J0(11).sturm_bound()
2
sage: J0(33).sturm_bound()
8
sage: J1(17).sturm_bound()
48
sage: J1(123456).sturm_bound()
1693483008
sage: JH(37,[2,3]).sturm_bound()
7
sage: J1(37).sturm_bound()
228
```

**torsion_subgroup**(n)

If $n$ is an integer, return the subgroup of points of order $n$. Return the $n$-torsion subgroup of elements of order dividing $n$ of this modular abelian variety $A$, i.e., the group $A[n]$.

**EXAMPLES:**

```python
sage: J1(13).torsion_subgroup(19)
Finite subgroup with invariants [19, 19, 19, 19] over QQ of Abelian variety
J1(13) of dimension 2

sage: A = J0(23)
sage: G = A.torsion_subgroup(5); G
Finite subgroup with invariants [5, 5, 5, 5] over QQ of Abelian variety
J0(23) of dimension 2
```
vector_space()

Return vector space corresponding to the modular abelian variety.

This is the lattice tensored with \( \mathbb{Q} \).

EXAMPLES:

```
sage: J0(37).vector_space()
Vector space of dimension 4 over Rational Field
sage: J0(37)[0].vector_space()
Vector space of degree 4 and dimension 2 over Rational Field
Basis matrix:
[ 1 -1 0 1/2]
[ 0 0 1 -1/2]
```

zero_subgroup()

Return the zero subgroup of this modular abelian variety, as a finite group.

EXAMPLES:

```
sage: A = J0(54); G = A.zero_subgroup(); G
Finite subgroup with invariants [] over QQ of Abelian variety J0(54) of dimension 4
sage: G.is_subgroup(A)
True
```

zero_subvariety()

Return the zero subvariety of self.

EXAMPLES:

```
sage: J = J0(37)
sage: J.zero_subvariety()
Simple abelian subvariety of dimension 0 of J0(37)
sage: J.zero_subvariety().level()
37
sage: J.zero_subvariety().newform_level()
(1, [])
```

class sage.modular.abvar.abvar.ModularAbelianVariety_modsym(modsym, **kwds)

Bases: sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract

Modular abelian variety that corresponds to a Hecke stable space of cuspidal modular symbols.

EXAMPLES: We create a modular abelian variety attached to a space of modular symbols.
```
sage: M = ModularSymbols(23).cuspidal_submodule()
sage: A = M.abelian_variety(); A
Abelian variety J0(23) of dimension 2
```

**brandt_module** (*p*)

Return the Brandt module at *p* that corresponds to self. This is the factor of the vector space on the ideal class set in an order of level *N* in the quaternion algebra ramified at *p* and infinity.

**INPUT:**

- *p* – prime that exactly divides the level

**OUTPUT:**

- Brandt module space that corresponds to self.

**EXAMPLES:**

```
sage: J0(43)[1].brandt_module(43)
Subspace of dimension 2 of Brandt module of dimension 4 of level 43 of weight →2 over Rational Field
sage: J0(43)[1].brandt_module(43).basis()
(((1, 0, -1/2, -1/2), (0, 1, -1/2, -1/2))
sage: J0(43)[0].brandt_module(43).basis()
(((0, 0, 1, -1),)
sage: J0(35)[0].brandt_module(5).basis()
(((1, 0, -1, 0),)
sage: J0(35)[0].brandt_module(7).basis()
(((1, -1, 1, -1),)
```

**component_group_order** (*p*)

Return the order of the component group of the special fiber at *p* of the Neron model of self.

**NOTE:** For bad primes, this is only implemented when the group if Gamma0 and *p* exactly divides the level.

**NOTE:** the input abelian variety must be simple

**ALGORITHM:** See “Component Groups of Quotients of J0(N)” by Kohel and Stein. That paper is about optimal quotients; however, section 4.1 of Conrad-Stein “Component Groups of Purely Toric Quotients”, one sees that the component group of an optimal quotient is the same as the component group of its dual (which is the subvariety).

**INPUT:**

- *p* – a prime number

**OUTPUT:**

- Integer

**EXAMPLES:**

```
sage: A = J0(37)[1]
sage: A.component_group_order(37)
3
sage: A = J0(43)[1]
sage: A.component_group_order(37)
1
sage: A.component_group_order(43)
7
```

(continues on next page)
tamagawa_number(p)
Return the Tamagawa number of this abelian variety at p.

NOTE: For bad primes, this is only implemented when the group if Gamma0 and p exactly divides the level and Atkin-Lehner acts diagonally on this abelian variety (e.g., if this variety is new and simple). See the self.component_group command for more information.

NOTE: the input abelian variety must be simple.

In cases where this function doesn’t work, consider using the self.tamagawa_number_bounds functions.

INPUT:
• p – a prime number

OUTPUT:
• Integer

EXAMPLES:

```
sage: A = J0(37)[1]
sage: A.tamagawa_number(37)
3
sage: A = J0(43)[1]
sage: A.tamagawa_number(37)
1
sage: A = J0(43)[1]
sage: A.tamagawa_number(43)
7
sage: A = J0(23)[0]
sage: A.tamagawa_number(23)
11
```

tamagawa_number_bounds(p)
Return a divisor and multiple of the Tamagawa number of self at p.

NOTE: the input abelian variety must be simple.

INPUT:
• p – a prime number

OUTPUT:
• div – integer; divisor of Tamagawa number at p
• mul – integer; multiple of Tamagawa number at p
• mul_primes – tuple; in case mul==0, a list of all primes that can possibly divide the Tamagawa number at p

EXAMPLES:

```
sage: A = J0(63).new_subvariety()[1]; A
Simple abelian subvariety 63b(1,63) of dimension 2 of J0(63)
sage: A.tamagawa_number_bounds(7)
(3, 3, ())
```

sage: A.tamagawa_number_bounds(3)
(1, 0, (2, 3, 5))

class sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract (groups, base_field, is_simple=None, new_form_level=None, isogeny_number=None, number=None, check=True)

Bases: sage.modular.abvar.abvar.ModularAbelianVariety_abstract

decomposition (simple=True, bound=None)

Decompose this modular abelian variety as a product of abelian subvarieties, up to isogeny.

INPUT: simple- bool (default: True) if True, all factors are simple. If False, each factor returned is isogenous to a power of a simple and the simples in each factor are distinct.

• bound - int (default: None) if given, only use Hecke operators up to this bound when decomposing. This can give wrong answers, so use with caution!

EXAMPLES:

sage: J = J0(33)
sage: J.decomposition()
[Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33),
Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33),
Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)]
sage: J1(17).decomposition()
[Simple abelian subvariety 17aG1(1,17) of dimension 1 of J1(17),
Simple abelian subvariety 17bG1(1,17) of dimension 4 of J1(17)]

dimension ()

Return the dimension of this modular abelian variety.

EXAMPLES:

sage: J0(37)[0].dimension()
1
sage: J0(43)[1].dimension()
2
sage: J1(17)[1].dimension()
4

group ()

Return the congruence subgroup associated that this modular abelian variety is associated to.

EXAMPLES:

sage: J0(13).group()
Congruence Subgroup Gamma0(13)
sage: J1(997).group()
Congruence Subgroup Gamma1(997)
sage: JH(37, [3]).group()
Congruence Subgroup Gamma_H(37) with H generated by [3]
sage: J0(37)[1].groups()
(Congruence Subgroup Gamma0(37),)

**groups()**
Return the tuple of groups associated to the modular symbols abelian variety. This is always a 1-tuple.

OUTPUT: tuple

EXAMPLES:

```sage
sage: A = ModularSymbols(33).cuspidal_submodule().abelian_variety(); A
Abelian variety J0(33) of dimension 3
sage: A.groups()
(Congruence Subgroup Gamma0(33),)
```

**is_ambient()**
Return True if this abelian variety attached to a modular symbols space is attached to the cuspidal subspace of the ambient modular symbols space.

OUTPUT: bool

EXAMPLES:

```sage
sage: A = ModularSymbols(43).cuspidal_subspace().abelian_variety(); A
Abelian variety J0(43) of dimension 3
sage: A.is_ambient()
True
sage: type(A)
<class 'sage.modular.abvar.abvar.ModularAbelianVariety_modsym_with_category'>
```

**is_subvariety (other)**
Return True if self is a subvariety of other.

EXAMPLES:

```sage
sage: J = J0(37); J
Abelian variety J0(37) of dimension 2
sage: A = J[0]; A
Simple abelian subvariety 37a(1,37) of dimension 1 of J0(37)
sage: A.is_subvariety(J)
True
sage: A.is_subvariety(J0(11))
False
```

There may be a way to map $A$ into $J_0(74)$, but $A$ is not equipped with any special structure of an embedding.

```sage
sage: A.is_subvariety(J0(74))
False
```
Some ambient examples:

```
sage: J = J0(37)
sage: J.is_subvariety(J)
True
sage: J.is_subvariety(25)
False
```

More examples:

```
sage: A = J0(42); D = A.decomposition(); D
[Simple abelian subvariety 14a(1,42) of dimension 1 of J0(42),
  Simple abelian subvariety 14a(3,42) of dimension 1 of J0(42),
  Simple abelian subvariety 21a(1,42) of dimension 1 of J0(42),
  Simple abelian subvariety 21a(2,42) of dimension 1 of J0(42),
  Simple abelian subvariety 42a(1,42) of dimension 1 of J0(42)]
sage: D[0].is_subvariety(A)
True
sage: D[1].is_subvariety(D[0] + D[1])
True
sage: D[2].is_subvariety(D[0] + D[1])
False
```

```
lattice()

Return the lattice defining this modular abelian variety.

OUTPUT:

A free \( \mathbb{Z} \)-module embedded in an ambient \( \mathbb{Q} \)-vector space.

EXAMPLES:

```
sage: A = ModularSymbols(33).cuspidal_submodule()[0].abelian_variety(); A
Abelian subvariety of dimension 1 of J0(33)
sage: A.lattice()
Free module of degree 6 and rank 2 over \( \mathbb{Z} \)
User basis matrix:
[ 1 0 0 -1 0 0]
[ 0 0 1 0 1 -1]
sage: type(A)
<class 'sage.modular.abvar.abvar.ModularAbelianVariety_modsym_with_category'>
```

```
modular_symbols(sign=0)

Return space of modular symbols (with given sign) associated to this modular abelian variety, if it can be
found by cutting down using Hecke operators. Otherwise raise a RuntimeError exception.

EXAMPLES:

```
sage: A = J0(37)
sage: A.modular_symbols()
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension
\rightarrow 5 for Gamma_0(37) of weight 2 with sign 0 over \mathbb{Q}
sage: A.modular_symbols(1)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension
\rightarrow 3 for Gamma_0(37) of weight 2 with sign 1 over \mathbb{Q}
```

More examples:
```python
sage: J0(11).modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field
sage: J0(11).modular_symbols(sign=1)
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0(11) of weight 2 with sign 1 over Rational Field
sage: J0(11).modular_symbols(sign=0)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field
sage: J0(11).modular_symbols(sign=-1)
Modular Symbols subspace of dimension 1 for Gamma_0(11) of weight 2 with sign -1 over Rational Field
```

Even more examples:

```python
sage: A = J0(33)[1]; A
Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33)
sage: A.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field
```

It is not always possible to determine the sign subspaces:

```python
sage: A.modular_symbols(1)
Traceback (most recent call last):
  ... 
RuntimeError: unable to determine sign (=1) space of modular symbols
sage: A.modular_symbols(-1)
Traceback (most recent call last):
  ... 
RuntimeError: unable to determine sign (==-1) space of modular symbols
```

**new_subvariety** *(p=None)*

Return the new or \( p \)-new subvariety of self.

**INPUT:**

- **self** - a modular abelian variety
- **p** - prime number or None (default); if \( p \) is a prime, return the \( p \)-new subvariety. Otherwise return the full new subvariety.

**EXAMPLES:**

```python
sage: J0(34).new_subvariety()
Abelian subvariety of dimension 1 of J0(34)
sage: J0(100).new_subvariety()
Abelian subvariety of dimension 1 of J0(100)
sage: J1(13).new_subvariety()
Abelian variety J1(13) of dimension 2
```

**old_subvariety** *(p=None)*

Return the old or \( p \)-old abelian variety of self.

**INPUT:**

- **self** - a modular abelian variety
• \( p \) - prime number or None (default); if \( p \) is a prime, return the \( p \)-old subvariety. Otherwise return the full old subvariety.

**EXAMPLES:**

```
sage: J0(33).old_subvariety()
Abelian subvariety of dimension 2 of J0(33)
sage: J0(100).old_subvariety()
Abelian subvariety of dimension 6 of J0(100)
sage: J1(13).old_subvariety()
Abelian subvariety of dimension 0 of J1(13)
```

```python
sage.modular.abvar.abvar.factor_modsym_space_new_factors(M)
```

Return the factorizations of all the new subspaces of \( M \).

**INPUT:**

• \( M \) – ambient modular symbols space

**OUTPUT:** list of decompositions corresponding to each new space.

**EXAMPLES:**

```
sage: M = ModularSymbols(33)
sage: sage.modular.abvar.abvar.factor_modsym_space_new_factors(M)
[[
    Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3
    \rightarrow for Gamma_0(11) of weight 2 with sign 0 over Rational Field,
],
[    Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9
    \rightarrow for Gamma_0(33) of weight 2 with sign 0 over Rational Field
]]
```

```python
sage.modular.abvar.abvar.factor_new_space(M)
```

Given a new space \( M \) of modular symbols, return the decomposition into simple of \( M \) under the Hecke operators.

**INPUT:**

• \( M \) - modular symbols space

**OUTPUT:** list of factors

**EXAMPLES:**

```
sage: M = ModularSymbols(37).cuspidal_subspace()
sage: sage.modular.abvar.abvar.factor_new_space(M)
[    Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5
    \rightarrow for Gamma_0(37) of weight 2 with sign 0 over Rational Field,
    Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5
    \rightarrow for Gamma_0(37) of weight 2 with sign 0 over Rational Field
]
```

```python
sage.modular.abvar.abvar.is_ModularAbelianVariety(x)
```

Return True if \( x \) is a modular abelian variety.

**INPUT:**

• \( x \) - object

**EXAMPLES:**
```
sage: from sage.modular.abvar.abvar import is_ModularAbelianVariety
sage: is_ModularAbelianVariety(5)
False
sage: is_ModularAbelianVariety(J0(37))
True
```

Returning True is a statement about the data type not whether or not some abelian variety is modular:

```
sage: is_ModularAbelianVariety(EllipticCurve('37a'))
False
```

```sage.modular.abvar.abvar.modsym_lattices(M, factors)
```

Append lattice information to the output of `simple_factorization_of_modsym_space`.

**INPUT:**

- `M` - modular symbols spaces
- `factors` - Sequence (`simple_factorization_of_modsym_space`)

**OUTPUT:** sequence with more information for each factor (the lattice)

**EXAMPLES:**

```
sage: M = ModularSymbols(33)
sage: factors = sage.modular.abvar.abvar.simple_factorization_of_modsym_space(M, simple=False)
sage: sage.modular.abvar.abvar.modsym_lattices(M, factors)
[(11, 0, None, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field, Free module of degree 6 and rank 4 over Integer Ring
  Echelon basis matrix:
  [ 1 0 0 0 -1 2]
  [ 0 1 0 0 -1 1]
  [ 0 0 1 0 -2 2]
  [ 0 0 0 1 -1 -1]),
(33, 0, None, Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field, Free module of degree 6 and rank 2 over Integer Ring
  Echelon basis matrix:
  [ 1 0 0 -1 0 0]
  [ 0 0 1 0 1 -1])]
```

```sage.modular.abvar.abvar.random_hecke_operator(M, t=None, p=2)
```

Return a random Hecke operator acting on `M`, got by adding to `t` a random multiple of `T_p`.

**INPUT:**

- `M` - modular symbols space
- `t` - None or a Hecke operator
- `p` - a prime

**OUTPUT:** Hecke operator prime

**EXAMPLES:**
```
sage: M = ModularSymbols(11).cuspidal_subspace()
sage: t, p = sage.modular.abvar.abvar.random_hecke_operator(M)
sage: p
3
sage: t, p = sage.modular.abvar.abvar.random_hecke_operator(M, t, p)
sage: p
5
```

```
sage.modular.abvar.abvar.simple_factorization_of_modsym_space(M, simple=True)
```

Return the canonical factorization of $M$ into (simple) subspaces.

**INPUT:**

- $M$ -- ambient modular symbols space
- $\text{simple}$ -- boolean (default: True); if set to False, isogenous simple factors are grouped together

**OUTPUT:** sequence

**EXAMPLES:**

```
sage: M = ModularSymbols(33)
sage: sage.modular.abvar.abvar.simple_factorization_of_modsym_space(M)
[(11, 0, 1, Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field),
 (11, 0, 3, Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field),
 (33, 0, 1, Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field)]
```

```
sage: sage.modular.abvar.abvar.simple_factorization_of_modsym_space(M, simple=False)
[(11, 0, None, Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field),
 (33, 0, None, Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 9 for Gamma_0(33) of weight 2 with sign 0 over Rational Field)]
```

```
sage.modular.abvar.abvar.sqrt_poly(f)
```

Return the square root of the polynomial $f$.

**Note:** At some point something like this should be a member of the polynomial class. For now this is just used internally by some charpoly functions above.

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: f = (x-1)*(x+2)*(x^2 + 1/3*x + 5)
sage: f
x^4 + 4/3*x^3 + 10/3*x^2 + 13/3*x - 10
sage: sage.modular.abvar.abvar.sqrt_poly(f^2)
x^4 + 4/3*x^3 + 10/3*x^2 + 13/3*x - 10
sage: sage.modular.abvar.abvar.sqrt_poly(f)
Traceback (most recent call last):
...
ValueError: f must be a perfect square

```python
sage: sage.modular.abvar.abvar.sqrt_poly(2*f^2)
Traceback (most recent call last):
...
ValueError: f must be monic
```
sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian(group)

Return the ambient Jacobian attached to a given congruence subgroup.

The result is cached using a weakref. This function is called internally by modular abelian variety constructors.

INPUT:

• group - a congruence subgroup.

OUTPUT: a modular abelian variety attached

EXAMPLES:

sage: import sage.modular.abvar.abvar_ambient_jacobian as abvar_ambient_jacobian
sage: A = abvar_ambient_jacobian.ModAbVar_ambient_jacobian(Gamma0(11))
sage: A
Abelian variety J0(11) of dimension 1
sage: B = abvar_ambient_jacobian.ModAbVar_ambient_jacobian(Gamma0(11))
sage: A is B
True

You can get access to and/or clear the cache as follows:

sage: abvar_ambient_jacobian._cache = {}
sage: B = abvar_ambient_jacobian.ModAbVar_ambient_jacobian(Gamma0(11))
sage: A is B
False

class sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class(group)

Bases: sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract

An ambient Jacobian modular abelian variety attached to a congruence subgroup.

ambient_variety()

Return the ambient modular abelian variety that contains self. Since self is a Jacobian modular abelian variety, this is just self.

OUTPUT: abelian variety

EXAMPLES:

sage: A = J0(17)
sage: A.ambient_variety()
Abelian variety J0(17) of dimension 1
sage: A is A.ambient_variety()
True
decomposition \((\text{simple}=\text{True}, \text{bound}=\text{None})\)
Decompose this ambient Jacobian as a product of abelian subvarieties, up to isogeny.

**EXAMPLES:**

```python
sage: J0(33).decomposition(simple=False)
[Abelian subvariety of dimension 2 of J0(33),
 Abelian subvariety of dimension 1 of J0(33)]
sage: J0(33).decomposition(simple=False)[1].is_simple()
True
sage: J0(33).decomposition(simple=False)[0].is_simple()
False
sage: J0(33).decomposition(simple=False)
[Abelian subvariety of dimension 2 of J0(33),
 Simple abelian subvariety 33a(None,33) of dimension 1 of J0(33)]
sage: J0(33).decomposition(simple=True)
[Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33),
 Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33),
 Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)]
```

degeneracy_map \((\text{level}, t=1, \text{check}=\text{True})\)
Return the \(t\)-th degeneracy map from self to J(\(\text{level}\)). Here \(t\) must be a divisor of either \(\text{level/self.level()}\) or \(\text{self.level()/level}\).

**INPUT:**

- **level** - integer (multiple or divisor of level of self)
- **\(t\)** - divisor of quotient of level of self and level
- **\(\text{check}\)** - bool (default: True); if True do some checks on the input

**OUTPUT:** a morphism

**EXAMPLES:**

```python
sage: J0(11).degeneracy_map(33)
Degeneracy map from Abelian variety J0(11) of dimension 1 to Abelian variety...
  →J0(33) of dimension 3 defined by [1]
sage: J0(11).degeneracy_map(33).matrix()
[ 0 -3 2 1 -2 0]
[ 1 -2 0 1 0 -1]
sage: J0(11).degeneracy_map(33,3).matrix()
[-1 0 0 0 1 -2]
[-1 -1 1 -1 1 0]
sage: J0(33).degeneracy_map(11,1).matrix()
[ 0 1]
[ 0 -1]
[ 1 -1]
[ 0 1]
[-1 1]
[ 0 0]
sage: J0(11).degeneracy_map(33,1).matrix() * J0(33).degeneracy_map(11,1).
  →matrix()
(continues on next page)
dimension()
Return the dimension of this modular abelian variety.

EXAMPLES:

```
sage: J0(2007).dimension()
sage: J1(13).dimension()
sage: J1(997).dimension()
sage: J0(389).dimension()
sage: JH(389,[4]).dimension()
sage: J1(389).dimension()
```


group()
Return the group that this Jacobian modular abelian variety is attached to.

EXAMPLES:

```
sage: J1(37).group()
sage: J0(5077).group()
sage: J = GammaH(11,[3]).modular_abelian_variety(); J
```


groups()
Return the tuple of congruence subgroups attached to this ambient Jacobian. This is always a tuple of length 1.

OUTPUT: tuple

EXAMPLES:

```
sage: J0(37).groups()
```


newform_decomposition(names=None)
Return the newforms of the simple subvarieties in the decomposition of self as a product of simple subvarieties, up to isogeny.

OUTPUT:

- an array of newforms

EXAMPLES:

```
sage: J0(81).newform_decomposition('a')
```


sage: J1(19).newform_decomposition('a')
[q - 2*q^3 - 2*q^4 + 3*q^5 + O(q^6),
 q + a1*q^2 + (-1/9*a1^5 - 1/3*a1^4 - 1/3*a1^3 + 1/3*a1^2 - a1 - 1)*q^3 + (4/9*a1^5 + 2*a1^4 + 14/3*a1^3 + 17/3*a1^2 + 6*a1 + 2)*q^4 + (-2/3*a1^5 - 11/3*a1^4 - 10*a1^3 - 14*a1^2 - 15*a1 - 9)*q^5 + O(q^6)]
FINITE SUBGROUPS OF MODULAR ABELIAN VARIETIES

Sage can compute with fairly general finite subgroups of modular abelian varieties. Elements of finite order are represented by equivalence classes of elements in $H_1(A,\mathbb{Q})$ modulo $H_1(A,\mathbb{Z})$. A finite subgroup can be defined by giving generators and via various other constructions. Given a finite subgroup, one can compute generators, as well as the structure as an abstract group. Arithmetic on subgroups is also supported, including adding two subgroups together, checking inclusion, etc.

TODO: Intersection, action of Hecke operators.

AUTHORS:

- William Stein (2007-03)

EXAMPLES:

```sage
sage: J = J0(33)
sage: C = J.cuspidal_subgroup()
sage: C
Finite subgroup with invariants [10, 10] over QQ of Abelian variety J0(33) of dimension 3
sage: C.order()
100
sage: C.gens()
[(1/10, 0, 1/10, 1/10, 3/10), (0, 1/5, 1/10, 0, 1/10, 9/10), (0, 0, 1/2, 0, 1/2, 1/2)]
sage: C.0 + C.1
[(1/10, 1/5, 1/5, 1/10, 1/5, 6/5)]
sage: 10*(C.0 + C.1)
[(0, 0, 0, 0, 0, 0)]
sage: G = C.subgroup([C.0 + C.1]); G
Finite subgroup with invariants [10] over QQbar of Abelian variety J0(33) of dimension 3
sage: G.gens()
[(1/10, 1/5, 1/5, 1/10, 1/5, 1/5)]
sage: G.order()
10
sage: G <= C
True
sage: G >= C
False
```

We make a table of the order of the cuspidal subgroup for the first few levels:

```sage
sage: for N in range(11,40):
    ....:     print("{} 
11 5
```
class sage.modular.abvar.finite_subgroup.FiniteSubgroup(abvar, field_of_definition=Rational Field)

Bases: sage.modules.module.Module

A finite subgroup of a modular abelian variety.

INPUT:

- abvar – a modular abelian variety
- field_of_definition – a field over which this group is defined

EXAMPLES:

This is an abstract base class, so there are no instances of this class itself:

```sage
A = J0(37)
G = A.torsion_subgroup(3); G
Finite subgroup with invariants [3, 3, 3, 3] over QQ of Abelian variety J0(37) of dimension 2

sage: type(G)
<class 'sage.modular.abvar.finite_subgroup.FiniteSubgroup_lattice_with_category'>

sage: from sage.modular.abvar.finite_subgroup import FiniteSubgroup

sage: isinstance(G, FiniteSubgroup)
True
```

Element

alias of sage.modular.abvar.torsion_point.TorsionPoint
abelian_variety()  
Return the abelian variety that this is a finite subgroup of.

EXAMPLES:

```
sage: J = J0(42)
sage: G = J.rational_torsion_subgroup(); G
Torsion subgroup of Abelian variety J0(42) of dimension 5
sage: G.abelian_variety()
Abelian variety J0(42) of dimension 5
```

exponent()  
Return the exponent of this finite abelian group.

OUTPUT: Integer

EXAMPLES:

```
sage: t = J0(33).hecke_operator(7)
sage: G = t.kernel()[0]; G
Finite subgroup with invariants [2, 2, 2, 2, 4, 4] over QQ of Abelian variety J0(33) of dimension 3
sage: G.exponent()
4
```

field_of_definition()  
Return the field over which this finite modular abelian variety subgroup is defined. This is a field over which this subgroup is defined.

EXAMPLES:

```
sage: J = J0(42)
sage: G = J.rational_torsion_subgroup(); G
Torsion subgroup of Abelian variety J0(42) of dimension 5
sage: G.field_of_definition()
Rational Field
```

gen(n)  
Return \(n\)th generator of self.

EXAMPLES:

```
sage: J = J0(23)
sage: C = J.torsion_subgroup(3)
sage: C.gens()
[[(1/3, 0, 0, 0)], [(0, 1/3, 0, 0)], [(0, 0, 1/3, 0)], [(0, 0, 0, 1/3)]]
sage: C.gen(0)
[(1/3, 0, 0, 0)]
sage: C.gen(3)
[(0, 0, 0, 1/3)]
sage: C.gen(4)
Traceback (most recent call last):
...
IndexError: list index out of range
```

Negative indices wrap around:

```
sage: C.gen(-1)
[(0, 0, 0, 1/3)]
```
gens ()
Return generators for this finite subgroup.

EXAMPLES: We list generators for several cuspidal subgroups:

```python
sage: J0(11).cuspidal_subgroup().gens()
[[[0, 1/5]]]
sage: J0(37).cuspidal_subgroup().gens()
[[[0, 0, 0, 1/3]]]
sage: J0(43).cuspidal_subgroup().gens()
[[[0, 1/7, 0, 6/7, 0, 5/7]]]
sage: J1(13).cuspidal_subgroup().gens()
[[[1/19, 0, 9/19, 9/19]], [[0, 1/19, 0, 9/19]]]
sage: J0(22).torsion_subgroup(6).gens()
[[[1/6, 0, 0, 0]], [[0, 1/6, 0, 0]], [[0, 0, 1/6, 0]], [[0, 0, 0, 1/6]]]
```

intersection (other)
Return the intersection of the finite subgroups self and other.

INPUT:
• **other** - a finite group

OUTPUT: a finite group

EXAMPLES:

```python
sage: E11a0, E11a1, B = J0(33)
sage: G = E11a0.torsion_subgroup(6); H = E11a0.torsion_subgroup(9)
sage: G.intersection(H)
Finite subgroup with invariants [3, 3] over QQ of Simple abelian subvariety ˓→11a(1,33) of dimension 1 of J0(33)
sage: W = E11a1.torsion_subgroup(15)
sage: G.intersection(W)
Finite subgroup with invariants [] over QQ of Simple abelian subvariety 11a(1, ˓→33) of dimension 1 of J0(33)
sage: E11a0.intersection(E11a1)[0]
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety ˓→11a(1,33) of dimension 1 of J0(33)
```

We intersect subgroups of different abelian varieties.

```python
sage: E11a0, E11a1, B = J0(33)
sage: G = E11a0.torsion_subgroup(5); H = E11a1.torsion_subgroup(5)
sage: G.intersection(H)
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety ˓→11a(1,33) of dimension 1 of J0(33)
sage: E11a0.intersection(E11a1)[0]
Finite subgroup with invariants [5] over QQ of Simple abelian subvariety ˓→11a(1,33) of dimension 1 of J0(33)
```

We intersect abelian varieties with subgroups:

```python
sage: t = J0(33).hecke_operator(7)
sage: G = t.kernel()[0]; G
Finite subgroup with invariants [2, 2, 2, 4, 4] over QQ of Abelian variety ˓→J0(33) of dimension 3
sage: A = J0(33).old_subvariety()
sage: A.intersection(G)
```

(continues on next page)
Finite subgroup with invariants [2, 2, 2, 2] over QQ of Abelian subvariety of dimension 2 of J0(33)
sage: A.hecke_operator(7).kernel()[0]
Finite subgroup with invariants [2, 2, 2, 2] over QQ of Abelian subvariety of dimension 2 of J0(33)
sage: B = J0(33).new_subvariety()
sage: B.intersection(G)
Finite subgroup with invariants [4, 4] over QQ of Abelian subvariety of dimension 1 of J0(33)
sage: B.hecke_operator(7).kernel()[0]
Finite subgroup with invariants [4, 4] over QQ of Abelian subvariety of dimension 1 of J0(33)
sage: A.intersection(B)[0]
Finite subgroup with invariants [3, 3] over QQ of Abelian subvariety of dimension 2 of J0(33)

invariants()
Return elementary invariants of this abelian group, by which we mean a nondecreasing (immutable) sequence of integers $n_i$, $1 \leq i \leq k$, with $n_i$ dividing $n_{i+1}$, and such that this group is abstractly isomorphic to $\mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}$.

EXAMPLES:

sage: J = J0(38)
sage: C = J.cuspidal_subgroup(); C
Finite subgroup with invariants [3, 45] over QQ of Abelian variety J0(38) of dimension 4
sage: v = C.invariants(); v
[3, 45]
sage: v[0] = 5
Traceback (most recent call last):
... ValueError: object is immutable; please change a copy instead.
sage: type(v[0])
<type 'sage.rings.integer.Integer'>

sage: C + 3
Finite subgroup with invariants [15] over QQ of Abelian variety J0(38) of dimension 4

An example involving another cuspidal subgroup:

sage: C = J0(22).cuspidal_subgroup(); C
Finite subgroup with invariants [5, 5] over QQ of Abelian variety J0(22) of dimension 2
sage: C.lattice()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[1/5 1/5 4/5 0]
[ 0 1 0 0]
[ 0 0 1 0]
[ 0 0 0 1/5]
sage: C.invariants()
[5, 5]

is_subgroup(other)
Return True exactly if self is a subgroup of other, and both are defined as subgroups of the same ambient
abelian variety.

EXAMPLES:

```sage
c = J0(22).cuspidal_subgroup()
sage: H = C.subgroup([C.0])
sage: K = C.subgroup([C.1])
sage: H.is_subgroup(K)
False
sage: K.is_subgroup(H)
False
sage: K.is_subgroup(C)
True
sage: H.is_subgroup(C)
True
```

lattice()

Return the lattice corresponding to this subgroup in the rational homology of the modular Jacobian product. The elements of the subgroup are represented by vectors in the ambient vector space (the rational homology), and this returns the lattice they span. EXAMPLES:

```sage
J = J0(33); C = J[0].cuspidal_subgroup(); C
Finite subgroup with invariants \([5]\) over \(\mathbb{Q}\) of Simple abelian subvariety
\(\text{-->la}(1,33)\) of dimension 1 of \(J0(33)\)
sage: C.lattice()
Free module of degree 6 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1/5 13/5 -2 -4/5 2 -1/5]
[ 0 3 -2 -1 2 0]
```

order()

Return the order (number of elements) of this finite subgroup.

EXAMPLES:

```sage
J = J0(42)
sage: C = J.cuspidal_subgroup()
sage: C.order()
2304
```

subgroup(gens)

Return the subgroup of self spanned by the given generators, which must all be elements of self.

EXAMPLES:

```sage
J = J0(23)
sage: G = J.torsion_subgroup(11); G
Finite subgroup with invariants \([11, 11, 11, 11]\) over \(\mathbb{Q}\) of Abelian variety
\(\text{-->J0(23)}\) of dimension 2
```

We create the subgroup of the 11-torsion subgroup of \(J_0(23)\) generated by the first 11-torsion point:

```sage
H = G.subgroup([G.0]); H
Finite subgroup with invariants \([11]\) over \(\mathbb{Q}\bar{\mathbb{Q}}\) of Abelian variety \(J0(23)\) of
\(\text{-->dimension 2}\)
sage: H.invariants()
[11]
```
We can also create a subgroup from a list of objects that can be converted into the ambient rational homology:

```python
sage: H == G.subgroup([[1/11,0,0,0]])
True
```

```python
class sage.modular.abvar.finite_subgroup.FiniteSubgroup_lattice
    (abvar, lattice, field_of_definition=Algebraic Field, check=True)
```

Bases: `sage.modular.abvar.finite_subgroup.FiniteSubgroup`

A finite subgroup of a modular abelian variety that is defined by a given lattice.

**INPUT:**
- `abvar` - a modular abelian variety
- `lattice` - a lattice that contains the lattice of `abvar`
- `field_of_definition` - the field of definition of this finite group scheme
- `check` - bool (default: True) whether or not to check that `lattice` contains the `abvar` lattice.

**EXAMPLES:**

```python
sage: J = J0(11)
sage: G = J.finite_subgroup([[1/3,0], [0,1/5]]); G
Finite subgroup with invariants [15] over QQbar of Abelian variety J0(11) of dimension 1
```

```python
lattice()
    Return lattice that defines this finite subgroup.
```

**EXAMPLES:**

```python
sage: J = J0(11)
sage: G = J.finite_subgroup([[1/3,0], [0,1/5]]); G
Finite subgroup with invariants [15] over QQbar of Abelian variety J0(11) of dimension 1
sage: G.lattice()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1/3 0]
[ 0 1/5]
```
TORSION SUBGROUPS OF MODULAR ABELIAN VARIETIES

Sage can compute information about the structure of the torsion subgroup of a modular abelian variety. Sage computes a multiple of the order by computing the greatest common divisor of the orders of the torsion subgroup of the reduction of the abelian variety modulo p for various primes p. Sage computes a divisor of the order by computing the rational cuspidal subgroup. When these two bounds agree (which is often the case), we determine the exact structure of the torsion subgroup.

AUTHORS:

• William Stein (2007-03)

EXAMPLES: First we consider \( J_0(50) \) where everything works out nicely:

```python
sage: J = J0(50)
sage: T = J.rational_torsion_subgroup(); T
Torsion subgroup of Abelian variety J0(50) of dimension 2
sage: T.multiple_of_order()
15
sage: T.divisor_of_order()
15
sage: T.gens()
[[(1/15, 3/5, 2/5, 14/15)]]
sage: T.invariants()
[15]
sage: d = J.decomposition(); d
[Simple abelian subvariety 50a(1,50) of dimension 1 of J0(50),
 Simple abelian subvariety 50b(1,50) of dimension 1 of J0(50)]
sage: d[0].rational_torsion_subgroup().order()
3
sage: d[1].rational_torsion_subgroup().order()
5
```

Next we make a table of the upper and lower bounds for each new factor.

```python
sage: for N in range(1,38):
    ....:     for A in J0(N).new_subvariety().decomposition():
    ....:         T = A.rational_torsion_subgroup()
    ....:         print('%-5s%-5s%-5s%-5s'%(N, A.dimension(), T.divisor_of_order(), T.multiple_of_order()))
            11  1   5    5
            14  1   6    6
            15  1   8    8
            17  1   4    4
            19  1   3    3
```

(continues on next page)
class sage.modular.abvar.torsion_subgroup.QQbarTorsionSubgroup(abvar)
    Bases: sage.modules.module.Module

Group of all torsion points over the algebraic closure on an abelian variety.

INPUT:
    • abvar - an abelian variety

EXAMPLES:

    sage: A = J0(23)
sage: A.qbar_torsion_subgroup()
Group of all torsion points in QQbar on Abelian variety J0(23) of dimension 2

Element
    alias of sage.modular.abvar.torsion_point.TorsionPoint

abelian_variety()
    Return the abelian variety that this is the set of all torsion points on.

    OUTPUT: abelian variety

    EXAMPLES:

    sage: J0(23).qbar_torsion_subgroup().abelian_variety()
Abelian variety J0(23) of dimension 2

field_of_definition()
    Return the field of definition of this subgroup. Since this is the group of all torsion it is defined over the base field of this abelian variety.

    OUTPUT: a field

    EXAMPLES:

    sage: J0(23).qbar_torsion_subgroup().field_of_definition()
Rational Field

class sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup(abvar)
    Bases: sage.modular.abvar.finite_subgroup.FiniteSubgroup

56 Chapter 5. Torsion subgroups of modular abelian varieties
The torsion subgroup of a modular abelian variety.

**divisor_of_order()**

Return a divisor of the order of this torsion subgroup of a modular abelian variety.

**OUTPUT:**

A divisor of this torsion subgroup.

**EXAMPLES:**

```sage
t = J0(37)[1].rational_torsion_subgroup()
sage: t.divisor_of_order()
3

sage: J = J1(19)
sage: J.rational_torsion_subgroup().divisor_of_order()
4383

sage: J = J0(45)
sage: J.rational_cusp_subgroup().order()
32
sage: J.rational_cuspidal_subgroup().order()
64
sage: J.rational_torsion_subgroup().divisor_of_order()
64
```

**lattice()**

Return lattice that defines this torsion subgroup, if possible.

**Warning:** There is no known algorithm in general to compute the rational torsion subgroup. Use `rational_cusp_group` to obtain a subgroup of the rational torsion subgroup in general.

**EXAMPLES:**

```sage
J0(11).rational_torsion_subgroup().lattice()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1 0]
[ 0 1/5]
```

The following fails because in fact I know of no (reasonable) algorithm to provably compute the torsion subgroup in general.

```sage
T = J0(33).rational_torsion_subgroup()
sage: T.lattice()
Traceback (most recent call last):
...
NotImplementedError: unable to compute the rational torsion subgroup in this case (there is no known general algorithm yet)
```

The problem is that the multiple of the order obtained by counting points over finite fields is twice the divisor of the order got from the rational cuspidal subgroup.

```sage
T.multiple_of_order(30)
200
sage: J0(33).rational_cusp_subgroup().order()
100
```
**multiple_of_order** *(maxp=None, proof=True)*

Return a multiple of the order.

**INPUT:**

- **proof** – a boolean (default: True)

The computation of the rational torsion order of \( J1(p) \) is conjectural and will only be used if proof=False. See Section 6.2.3 of [CES2003].

**EXAMPLES:**

```
sage: J = J1(11); J
Abelian variety J1(11) of dimension 1
sage: J.rational_torsion_subgroup().multiple_of_order()
5
sage: J = J0(17)
sage: J.rational_torsion_subgroup().order()
4
```

This is an example where proof=False leads to a better bound and better performance.

```
sage: J = J1(23)
sage: J.rational_torsion_subgroup().multiple_of_order() # long time (2s)
9406793
sage: J.rational_torsion_subgroup().multiple_of_order(proof=False)
408991
```

**multiple_of_order_using_frobp** *(maxp=None)*

Return a multiple of the order of this torsion group.

In the \( \Gamma_0 \) case, the multiple is computed using characteristic polynomials of Hecke operators of odd index not dividing the level. In the \( \Gamma_1 \) case, the multiple is computed by expressing the Frobenius polynomial in terms of the characteristic polynomial of left multiplication by \( a_p \) for odd primes \( p \) not dividing the level.

**INPUT:**

- **maxp** - (default: None) If maxp is None (the default), return gcd of best bound computed so far with bound obtained by computing GCD’s of orders modulo \( p \) until this gcd stabilizes for 3 successive primes. If maxp is given, just use all primes up to and including maxp.

**EXAMPLES:**

```
sage: J = J0(11)
sage: G = J.rational_torsion_subgroup(); G
Torsion subgroup of Abelian variety J0(11) of dimension 32
sage: G.multiple_of_order_using_frobp(11)
5
```

Increasing maxp may yield a tighter bound. If maxp=None, then Sage will use more primes until the multiple stabilizes for 3 successive primes.

```
sage: J = J0(389)
sage: G = J.rational_torsion_subgroup(); G
Torsion subgroup of Abelian variety J0(389) of dimension 32
sage: G.multiple_of_order_using_frobp()
97
```

(continues on next page)
We can compute the multiple of order of the torsion subgroup for Gamma0 and Gamma1 varieties, and their products.

```python
sage: A = J0(11) * J0(33)
sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp()
1000

sage: A = J1(23)
sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp()
9406793

sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp(maxp=50)
408991

sage: A = J1(19) * J0(21)
sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp()
35064

The next example illustrates calling this function with a larger input and how the result may be cached when maxp is None:

```python
sage: T = J0(43)[1].rational_torsion_subgroup()
sage: T.multiple_of_order_using_frobp()
14

sage: T.multiple_of_order_using_frobp(50)
7

sage: T.multiple_of_order_using_frobp()
7
```

This function is not implemented for general congruence subgroups unless the dimension is zero.

```python
sage: A = JH(13, [2]); A
Abelian variety J0(13) of dimension 0

sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp()
1

sage: A = JH(15, [2]); A
Abelian variety JH(15,[2]) of dimension 1

sage: A.rational_torsion_subgroup().multiple_of_order_using_frobp()
Traceback (most recent call last):
...
NotImplementedError: torsion multiple only implemented for Gamma0 and Gamma1
```

`order (proof=True)`

Return the order of the torsion subgroup of this modular abelian variety.

This function may fail if the multiple obtained by counting points modulo $p$ exceeds the divisor obtained from the rational cuspidal subgroup.

The computation of the rational torsion order of $J1(p)$ is conjectural and will only be used if proof=False. See Section 6.2.3 of [CES2003].

INPUT:
• proof – a boolean (default: True)

OUTPUT:

The order of this torsion subgroup.

EXAMPLES:

```
sage: A = J0(11)
sage: A.rational_torsion_subgroup().order()
5
sage: A = J0(23)
sage: A.rational_torsion_subgroup().order()
11
sage: T = J0(37)[1].rational_torsion_subgroup()
sage: T.order()
3
sage: J = J1(13)
sage: J.rational_torsion_subgroup().order()
19
```

Sometimes the order can only be computed with proof=False.

```
sage: J = J1(23)
sage: J.rational_torsion_subgroup().order()
Traceback (most recent call last):
  ...  
RuntimeError: Unable to compute order of torsion subgroup (it is in [408991, 9406793])
sage: J.rational_torsion_subgroup().order(proof=False)
408991
```

possible_orders (proof=True)

Return the possible orders of this torsion subgroup. Outside of special cases, this is done by computing a divisor and multiple of the order.

INPUT:

• proof – a boolean (default: True)

OUTPUT:

• an array of positive integers

The computation of the rational torsion order of J1(p) is conjectural and will only be used if proof=False. See Section 6.2.3 of [CES2003].

EXAMPLES:

```
sage: J0(11).rational_torsion_subgroup().possible_orders()
[5]
sage: J0(33).rational_torsion_subgroup().possible_orders()
[100, 200]
sage: J1(13).rational_torsion_subgroup().possible_orders()
[19]
sage: J1(16).rational_torsion_subgroup().possible_orders()
[1, 2, 4, 5, 10, 20]
```

60 Chapter 5. Torsion subgroups of modular abelian varieties
CUSPIDAL SUBGROUPS OF MODULAR ABELIAN VARIETIES

AUTHORS:

- William Stein (2007-03, 2008-02)

EXAMPLES: We compute the cuspidal subgroup of $J_1(13)$:

```sage
sage: A = J1(13)
sage: C = A.cuspidal_subgroup(); C
Finite subgroup with invariants [19, 19] over QQ of Abelian variety J1(13) of dimension 2
sage: C.gens()
[(1/19, 0, 9/19, 9/19), (0, 1/19, 0, 9/19)]
sage: C.order()
361
sage: C.invariants()
[19, 19]
```

We compute the cuspidal subgroup of $J_0(54)$:

```sage
sage: A = J0(54)
sage: C = A.cuspidal_subgroup(); C
Finite subgroup with invariants [3, 3, 3, 3, 3, 9] over QQ of Abelian variety J0(54) of dimension 4
sage: C.gens()
[(1/3, 0, 0, 0, 1/3, 0, 0, 1/3, 2/3), (0, 1/3, 0, 0, 0, 1/3, 0, 2/3, 0, 1/3), (0, 0, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 2/9), (0, 0, 0, 1/3, 0, 1/3, 0, 1/3, 0, 1/3), (0, 0, 0, 0, 0, 1/3, 2/3)]
sage: C.order()
2187
sage: C.invariants()
[3, 3, 3, 3, 3, 9]
```

We compute the subgroup of the cuspidal subgroup generated by rational cusps.

```sage
sage: C = J0(54).rational_cusp_subgroup(); C
Finite subgroup with invariants [3, 3, 9] over QQ of Abelian variety J0(54) of dimension 4
sage: C.gens()
[(1/3, 0, 0, 1/3, 2/3, 1/3, 0, 1/3), (0, 0, 1/9, 1/9, 1/9, 1/9, 7/9, 7/9, 1/9, 8/9), (0, 0, 0, 0, 0, 0, 1/3, 2/3)]
sage: C.order()
81
sage: C.invariants()
[3, 3, 9]
```
This might not give us the exact rational torsion subgroup, since it might be bigger than order 81:

```
sage: J0(54).rational_torsion_subgroup().multiple_of_order()
sage: J0(54).rational_torsion_subgroup().multiple_of_order()
243
```

```
class sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup(abvar, field_of_definition=Rational Field)

Bases: sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup_generic

EXAMPLES:
```
sage: a = J0(65)[2]
sage: t = a.cuspidal_subgroup()
sage: t.order()

6
```

```
lattice()

Retuned cached tuple of vectors that define elements of the rational homology that generate this finite subgroup.

OUTPUT:

• tuple - cached

EXAMPLES:
```
sage: J = J0(27)
sage: G = J.cuspidal_subgroup()
sage: G.lattice()

Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1/3 0]
[0 1/3]

Test that the result is cached:
```
sage: G.lattice() is G.lattice()
True
```
```
class sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup_generic(abvar, field_of_definition=Rational Field)

Bases: sage.modular.abvar.finite_subgroup.FiniteSubgroup

class sage.modular.abvar.cuspidal_subgroup.RationalCuspSubgroup(abvar, field_of_definition=Rational Field)

Bases: sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup_generic

EXAMPLES:
```
sage: a = J0(65)[2]
sage: t = a.rational_cusp_subgroup()
sage: t.order()

6
```

```
lattice()

Return lattice that defines this group.

OUTPUT: lattice
EXAMPLES:

```
sage: G = J0(27).rational_cusp_subgroup()
sage: G.lattice()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1/3  0]
[ 0  1]
```

Test that the result is cached.

```
sage: G.lattice() is G.lattice()
True
```

class sage.modular.abvar.cuspidal_subgroup.RationalCuspidalSubgroup(abvar, field_of_definition=Rational Field)

Bases: sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup_generic

EXAMPLES:

```
sage: a = J0(65)[2]
sage: t = a.rational_cusp_subgroup()
sage: t.order()
6
```

```
 sage: G = J0(27).rational_cuspidal_subgroup()
sage: G.lattice()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[1/3  0]
[ 0  1]
```

Test that the result is cached.

```
sage: G.lattice() is G.lattice()
True
```

sage.modular.abvar.cuspidal_subgroup.is_rational_cusp_gamma0(c, N, data)

Return True if the rational number \( c \) is a rational cusp of level \( N \).

This uses remarks in Glenn Steven’s Ph.D. thesis.

INPUT:

- \( c \) - a cusp
- \( N \) - a positive integer
- \( \text{data} \) - the list \([n \text{ for } n \text{ in } \text{range}(2,N) \text{ if } \gcd(n,N) == 1]\), which is passed in as a parameter purely for efficiency reasons.

EXAMPLES:
```python
sage: from sage.modular.abvar.cuspidal_subgroup import is_rational_cusp_gamma0
sage: N = 27
sage: data = [n for n in range(2,N) if gcd(n,N) == 1]

sage: is_rational_cusp_gamma0(Cusp(1/3), N, data)
False
sage: is_rational_cusp_gamma0(Cusp(1), N, data)
True
sage: is_rational_cusp_gamma0(Cusp(oo), N, data)
True
sage: is_rational_cusp_gamma0(Cusp(2/9), N, data)
False
```
HOMOLOGY OF MODULAR ABELIAN VARIETIES

Sage can compute with homology groups associated to modular abelian varieties with coefficients in any commutative ring. Supported operations include computing matrices and characteristic polynomials of Hecke operators, rank, and rational decomposition as a direct sum of factors (obtained by cutting out kernels of Hecke operators).

AUTHORS:

- William Stein (2007-03)

EXAMPLES:

```python
sage: J = J0(43)
sage: H = J.integral_homology()
sage: H
Integral Homology of Abelian variety J0(43) of dimension 3
sage: H.hecke_matrix(19)
[ 0 0 -2 0 2 0]
[ 2 -4 -2 0 2 0]
[ 0 0 -2 -2 0 0]
[ 2 0 -2 -4 2 -2]
[ 0 2 0 -2 -2 0]
[ 0 2 0 -2 0 0]
sage: H.base_ring()
Integer Ring
sage: d = H.decomposition(); d
[Submodule of rank 2 of Integral Homology of Abelian variety J0(43) of dimension 3, Submodule of rank 4 of Integral Homology of Abelian variety J0(43) of dimension 3]
sage: a = d[0]
sage: a.hecke_matrix(5)
[-4 0]
[ 0 -4]
sage: a.T(7)
Hecke operator T_7 on Submodule of rank 2 of Integral Homology of Abelian variety J0(43) of dimension 3
```

```python
class sage.modular.abvar.homology.Homology(base_ring, level, weight, category=None):
    Bases: sage.modular.hecke.module.HeckeModule_free_module

    A homology group of an abelian variety, equipped with a Hecke action.

    hecke_polynomial(n, var='x')
    Return the n-th Hecke polynomial in the given variable.

    INPUT:

    - n - positive integer
```
• var - string (default: ‘x’) the variable name

OUTPUT: a polynomial over ZZ in the given variable

EXAMPLES:

```
sage: H = J0(43).integral_homology(); H
Integral Homology of Abelian variety J0(43) of dimension 3
sage: f = H.hecke_polynomial(3); f
x^6 + 4*x^5 - 16*x^3 - 12*x^2 + 16*x + 16
sage: parent(f)
Univariate Polynomial Ring in x over Integer Ring
sage: H.hecke_polynomial(3,'w')
w^6 + 4*w^5 - 16*w^3 - 12*w^2 + 16*w + 16
```

```python
class sage.modular.abvar.homology.Homology_abvar(abvar, base)

Bases: sage.modular.abvar.homology.Homology

The homology of a modular abelian variety.

abelian_variety()
Return the abelian variety that this is the homology of.

EXAMPLES:

```
sage: H = J0(48).homology()
sage: H.abelian_variety()
Abelian variety J0(48) of dimension 3
```

ambient_hecke_module()
Return the ambient Hecke module that this homology is contained in.

EXAMPLES:

```
sage: H = J0(48).homology(); H
Integral Homology of Abelian variety J0(48) of dimension 3
sage: H.ambient_hecke_module()
Integral Homology of Abelian variety J0(48) of dimension 3
```

free_module()
Return the underlying free module of this homology group.

EXAMPLES:

```
sage: H = J0(48).homology()
sage: H.free_module()
Ambient free module of rank 6 over the principal ideal domain Integer Ring
```

gen(n)
Return \( n^{th} \) generator of self.

This is not yet implemented!

EXAMPLES:

```
sage: H = J0(37).homology()
sage: H.gen(0)  # this will change
Traceback (most recent call last):
...
NotImplementedError: homology classes not yet implemented
```
**gens()**

Return generators of self.

This is not yet implemented!

**EXAMPLES:**

```python
sage: H = J0(37).homology()
sage: H.gens()  # this will change
Traceback (most recent call last):
...  
NotImplementedError: homology classes not yet implemented
```

**hecke_bound()**

Return bound on the number of Hecke operators needed to generate the Hecke algebra as a $\mathbb{Z}$-module acting on this space.

**EXAMPLES:**

```python
sage: J0(48).homology().hecke_bound()
16
sage: J1(15).homology().hecke_bound()
32
```

**hecke_matrix(n)**

Return the matrix of the $n$-th Hecke operator acting on this homology group.

**INPUT:**

- $n$ - a positive integer

**OUTPUT:** a matrix over the coefficient ring of this homology group

**EXAMPLES:**

```python
sage: H = J0(23).integral_homology()
sage: H.hecke_matrix(3)
[-1 -2 2 0]
[ 0 -3 2 -2]
[ 2 -4 3 -2]
[ 2 -2 0 1]
```

The matrix is over the coefficient ring:

```python
sage: J = J0(23)
sage: J.homology(QQ[I]).hecke_matrix(3).parent()
Full MatrixSpace of 4 by 4 dense matrices over Number Field in I with defining polynomial x^2 + 1 with I = 1*I
```

**rank()**

Return the rank as a module or vector space of this homology group.

**EXAMPLES:**

```python
sage: H = J0(5077).homology(); H
Integral Homology of Abelian variety J0(5077) of dimension 422
sage: H.rank()
844
```
```python
submodule(U, check=True)
```

Return the submodule of this homology group given by $U$, which should be a submodule of the free module associated to this homology group.

**INPUT:**

- $U$ - submodule of ambient free module (or something that defines one)
- check - currently ignored.

**Note:** We do not check that $U$ is invariant under all Hecke operators.

**EXAMPLES:**

```python
sage: H = J0(23).homology(); H
Integral Homology of Abelian variety J0(23) of dimension 2
sage: F = H.free_module()
sage: U = F.span([[1,2,3,4]])
sage: M = H.submodule(U); M
Submodule of rank 1 of Integral Homology of Abelian variety J0(23) of dimension 2
```

Note that the submodule command doesn’t actually check that the object defined is a homology group or is invariant under the Hecke operators. For example, the fairly random $M$ that we just defined is not invariant under the Hecke operators, so it is not a Hecke submodule - it is only a $\mathbb{Z}$-submodule.

```python
sage: M.hecke_matrix(3)
Traceback (most recent call last):
  ... ArithmeticError: subspace is not invariant under matrix
```

**class** `sage.modular.abvar.homology.Homology_over_base(abvar, base_ring)`

The homology over a modular abelian variety over an arbitrary base commutative ring (not $\mathbb{Z}$ or $\mathbb{Q}$).

**hecke_matrix(n)**

Return the matrix of the $n$-th Hecke operator acting on this homology group.

**EXAMPLES:**

```python
sage: t = J1(13).homology(GF(3)).hecke_matrix(3); t
[1 2 2 1]
[1 0 2 0]
[0 0 0 1]
[0 0 2 1]
sage: t.base_ring()
Finite Field of size 3
```

**class** `sage.modular.abvar.homology.Homology_submodule(ambient, submodule)`

A submodule of the homology of a modular abelian variety.

**ambient_hecke_module()**

Return the ambient Hecke module that this homology is contained in.

**EXAMPLES:**

```python
```
sage: H = J0(48).homology(); H
Integral Homology of Abelian variety J0(48) of dimension 3
sage: d = H.decomposition(); d
[
Submodule of rank 2 of Integral Homology of Abelian variety J0(48) of
→dimension 3,
Submodule of rank 4 of Integral Homology of Abelian variety J0(48) of
→dimension 3
]
sage: d[0].ambient_hecke_module()
Integral Homology of Abelian variety J0(48) of dimension 3

free_module()
Return the underlying free module of the homology group.

EXAMPLES:

sage: H = J0(48).homology()
sage: K = H.decomposition()[1]; K
Submodule of rank 4 of Integral Homology of Abelian variety J0(48) of
→dimension 3
sage: K.free_module()
Free module of degree 6 and rank 4 over Integer Ring
Echelon basis matrix:
[ 1 0 0 0 0 0]
[ 0 1 0 0 1 -1]
[ 0 0 1 0 -1 1]
[ 0 0 0 1 0 -1]

hecke_bound()
Return a bound on the number of Hecke operators needed to generate the Hecke algebra acting on this homology group.

EXAMPLES:

sage: d = J0(43).homology().decomposition(2); d
[
Submodule of rank 2 of Integral Homology of Abelian variety J0(43) of
→dimension 3,
Submodule of rank 4 of Integral Homology of Abelian variety J0(43) of
→dimension 3
]
Because the first factor has dimension 2 it corresponds to an elliptic curve, so we have a Hecke bound of 1.

sage: d[0].hecke_bound()
1
sage: d[1].hecke_bound()
8

hecke_matrix(n)
Return the matrix of the n-th Hecke operator acting on this homology group.

EXAMPLES:

sage: d = J0(125).homology(GF(17)).decomposition(2); d
[ (continues on next page)
Submodule of rank 4 of Homology with coefficients in Finite Field of size 17→ of Abelian variety \(J_0(125)\) of dimension 8,
Submodule of rank 4 of Homology with coefficients in Finite Field of size 17→ of Abelian variety \(J_0(125)\) of dimension 8,
Submodule of rank 8 of Homology with coefficients in Finite Field of size 17→ of Abelian variety \(J_0(125)\) of dimension 8

```python
sage: t = d[0].hecke_matrix(17); t
[16 15 15 0]
[ 0 5 0 2]
[ 2 0 5 15]
[ 0 15 0 16]
sage: t.base_ring()
Finite Field of size 17
tsage: t.fcp()
(x^2 + 13*x + 16)^2
```

**rank()**

Return the rank of this homology group.

**EXAMPLES:**

```python
sage: d = J0(43).homology().decomposition(2)
sage: [H.rank() for H in d]
[2, 4]
```

**class** `sage.modular.abvar.homology.IntegralHomology(abvar)`

Bases: `sage.modular.abvar.homology.Homology_abvar`

The integral homology \(H_1(A,\mathbb{Z})\) of a modular abelian variety.

**hecke_matrix\(n\)**

Return the matrix of the \(n\)-th Hecke operator acting on this homology group.

**EXAMPLES:**

```python
sage: J0(48).integral_homology().hecke_bound()
16
sage: t = J1(13).integral_homology().hecke_matrix(3); t
[-2 2 2 -2]
[-2 0 2 0]
[ 0 0 0 -2]
[ 0 0 2 -2]
sage: t.base_ring()
Integer Ring
```

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

Return the \(n\)-th Hecke polynomial on this integral homology group.

**EXAMPLES:**

```python
sage: f = J0(43).integral_homology().hecke_polynomial(2)
sage: f.base_ring()
Integer Ring
sage: factor(f)
(x + 2)^2 * (x^2 - 2)^2
```

**class** `sage.modular.abvar.homology.RationalHomology(abvar)`

Bases: `sage.modular.abvar.homology.Homology_abvar`

70 Chapter 7. Homology of modular abelian varieties
The rational homology \(H_1(A, \mathbb{Q})\) of a modular abelian variety.

**hecke_matrix** \((n)\)
Return the matrix of the \(n\)-th Hecke operator acting on this homology group.

**EXAMPLES:**

```python
sage: t = J1(13).homology(QQ).hecke_matrix(3); t
[-2  2  2 -2]
[-2  0  2  0]
[ 0  0  0 -2]
[ 0  0  2 -2]
sage: t.base_ring()
Rational Field
```

```python
sage: t = J1(13).homology(GF(3)).hecke_matrix(3); t
[1 2 2 1]
[1 0 2 0]
[0 0 1]
[0 0 2 1]
sage: t.base_ring()
Finite Field of size 3
```

**hecke_polynomial** \((n, \text{var}='x')\)
Return the \(n\)-th Hecke polynomial on this rational homology group.

**EXAMPLES:**

```python
sage: f = J0(43).rational_homology().hecke_polynomial(2)
sage: f.base_ring()
Rational Field
sage: factor(f)
(x + 2) * (x^2 - 2)
```
CHAPTER EIGHT

SPACES OF HOMOMORPHISMS BETWEEN MODULAR ABELIAN VARIETIES

EXAMPLES:

First, we consider $J_0(37)$. This Jacobian has two simple factors, corresponding to distinct newforms. These two intersect nontrivially in $J_0(37)$.

```
sage: J = J0(37)
sage: D = J.decomposition() ; D
[Simple abelian subvariety 37a(1,37) of dimension 1 of J0(37),
 Simple abelian subvariety 37b(1,37) of dimension 1 of J0(37)]
sage: D[0].intersection(D[1])
(Finite subgroup with invariants [2, 2] over QQ of Simple abelian subvariety 37a(1, ˓→37) of dimension 1 of J0(37),
 Simple abelian subvariety of dimension 0 of J0(37))
```

As an abstract product, since these newforms are distinct, the corresponding simple abelian varieties are not isogenous, and so there are no maps between them. The endomorphism ring of the corresponding product is thus isomorphic to the direct sum of the endomorphism rings for each factor. Since the factors correspond to abelian varieties of dimension 1, these endomorphism rings are each isomorphic to $\mathbb{Z}Z$.

```
sage: Hom(D[0],D[1]).gens()
()sage: A = D[0] * D[1] ; A
Abelian subvariety of dimension 2 of J0(37) x J0(37)
sage: A.endomorphism_ring().gens()
(Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(37) x J0(37),
 Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(37) x J0(37))
sage: [ x.matrix() for x in A.endomorphism_ring().gens() ]
[[1 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 1] ]
```

However, these two newforms have a congruence between them modulo 2, which gives rise to interesting endomorphisms of $J_0(37)$.

```
sage: E = J.endomorphism_ring()
sage: E.gens()
(Abelian variety endomorphism of Abelian variety J0(37) of dimension 2, (continues on next page)
Abelian variety endomorphism of Abelian variety J0(37) of dimension 2

```
sage: [ x.matrix() for x in E.gens() ]
```

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

```
sage: (-1*E.gens()[0] + E.gens()[1]).matrix()
```

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

Of course, these endomorphisms will be reflected in the Hecke algebra, which is in fact the full endomorphism ring of J0(37) in this case:

```
sage: J.hecke_operator(2).matrix()
```

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

```
sage: T = E.image_of_hecke_algebra()
sage: T.gens()
```

```
(Abelian variety endomorphism of Abelian variety J0(37) of dimension 2,
Abelian variety endomorphism of Abelian variety J0(37) of dimension 2)
```

```
sage: [ x.matrix() for x in T.gens() ]
```

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

```
sage: T.index_in(E)
```

```
1
```

Next, we consider J0(33). In this case, we have both oldforms and newforms. There are two copies of J0(11), one for each degeneracy map from J0(11) to J0(33). There is also one newform at level 33. The images of the two degeneracy maps are, of course, isogenous.

```
sage: J = J0(33)
sage: D = J.decomposition()
sage: D
```

```
[Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33),
Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33),
Simple abelian subvariety 33a(1,33) of dimension 1 of J0(33)]
```

```
sage: Hom(D[0],D[1]).gens()
```

```
(Abelian variety morphism:
  From: Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
  To:  Simple abelian subvariety 11a(3,33) of dimension 1 of J0(33),)
```

```
sage: Hom(D[0],D[1]).gens()[0].matrix()
```

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

Then this gives that the component corresponding to the sum of the oldforms will have a rank 4 endomorphism ring.
We also have a rank one endomorphism ring for the newform 33a (since it is again 1-dimensional), which gives a rank 5 endomorphism ring for $J_0(33)$.

```python
sage: DD = J.decomposition(simple=False) ; DD
[Abelian subvariety of dimension 2 of J0(33),
 Abelian subvariety of dimension 1 of J0(33)]
sage: A, B = DD
sage: A == D[0] + D[1]
True
sage: A.endomorphism_ring().gens()
(Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(33),
 Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(33),
 Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(33),
 Abelian variety endomorphism of Abelian subvariety of dimension 2 of J0(33))
sage: B.endomorphism_ring().gens()
(Abelian variety endomorphism of Abelian subvariety of dimension 1 of J0(33),)
```

In this case, the image of the Hecke algebra will only have rank 3, so that it is of infinite index in the full endomorphism ring. However, if we call this image $T$, we can still ask about the index of $T$ in its saturation, which is 1 in this case.

```python
sage: T = E.image_of_hecke_algebra()  # long time
sage: T.gens()  # long time
(Abelian variety endomorphism of Abelian variety J0(33) of dimension 3,
 Abelian variety endomorphism of Abelian variety J0(33) of dimension 3,
 Abelian variety endomorphism of Abelian variety J0(33) of dimension 3,
 Abelian variety endomorphism of Abelian variety J0(33) of dimension 3)
sage: T.index_in(E)  # long time
+Infinity
sage: T.index_in_saturation()  # long time
1
```

AUTHORS:

- William Stein (2007-03)
- Craig Citro, Robert Bradshaw (2008-03): Rewrote with modabvar overhaul

class sage.modular.abvar.homspace.EndomorphismSubring(A, gens=None, category=None):

    Bases: sage.modular.abvar.homspace.Homspace, sage.rings.ring.Ring

    A subring of the endomorphism ring.

    INPUT:

    - $A$ - an abelian variety
    - gens - (default: None); optional; if given should be a tuple of the generators as matrices

    EXAMPLES:

    ```python
    sage: J0(23).endomorphism_ring()
    Endomorphism ring of Abelian variety J0(23) of dimension 2
    sage: sage.modular.abvar.homspace.EndomorphismSubring(J0(25))
    ```

(continues on next page)
Endomorphism ring of Abelian variety $J_0(25)$ of dimension 0

```python
sage: E = J0(11).endomorphism_ring()
sage: type(E)
<class 'sage.modular.abvar.homspace.EndomorphismSubring_with_category'>
sage: E.homset_category()
Category of modular abelian varieties over Rational Field
sage: E.category()
Category of endsets of modular abelian varieties over Rational Field
sage: E
in Rings()
True
sage: TestSuite(E).run(skip="_test_prod")
```

**abelian_variety()**

Return the abelian variety that this endomorphism ring is attached to.

**EXAMPLES:**

```python
sage: J0(11).endomorphism_ring().abelian_variety()
Abelian variety $J_0(11)$ of dimension 1
```

**discriminant()**

Return the discriminant of this ring, which is the discriminant of the trace pairing.

**Note:** One knows that for modular abelian varieties, the endomorphism ring should be isomorphic to an order in a number field. However, the discriminant returned by this function will be $2^n$ ($n = \text{self.dimension()}$) times the discriminant of that order, since the elements are represented as $2d \times 2d$ matrices. Notice, for example, that the case of a one dimensional abelian variety, whose endomorphism ring must be $\mathbb{Z}$, has discriminant 2, as in the example below.

**EXAMPLES:**

```python
sage: J0(33).endomorphism_ring().discriminant()
-64800
sage: J0(46).endomorphism_ring().discriminant()  # long time (6s on sage.math, → 2011)
2420000000
sage: J0(11).endomorphism_ring().discriminant()
2
```

**image_of_hecke_algebra**(check_every=1)

Compute the image of the Hecke algebra inside this endomorphism subring.

We simply calculate Hecke operators up to the Sturm bound, and look at the submodule spanned by them. While computing, we can check to see if the submodule spanned so far is saturated and of maximal dimension, in which case we may be done. The optional argument check_every determines how many Hecke operators we add in before checking to see if this condition is met.

**INPUT:**

- check_every – integer (default: 1) If this integer is positive, this integer determines how many Hecke operators we add in before checking to see if the submodule spanned so far is maximal and saturated.

**OUTPUT:**

- The image of the Hecke algebra as an subring of self.
EXAMPLES:

```python
sage: E = J0(33).endomorphism_ring()
sage: E.image_of_hecke_algebra()
Subring of endomorphism ring of Abelian variety J0(33) of dimension 3
sage: E.image_of_hecke_algebra().gens()
(Abelian variety endomorphism of Abelian variety J0(33) of dimension 3,
 Abelian variety endomorphism of Abelian variety J0(33) of dimension 3,
 Abelian variety endomorphism of Abelian variety J0(33) of dimension 3)
sage: [ x.matrix() for x in E.image_of_hecke_algebra().gens() ]
[[1 0 0 0 0 0 0 0 1 -1 1 -1]
 [0 2 0 -1 1 -1]
 [0 1 0 0 0 -1 -2 2 -1 2 -1]
 [0 0 1 0 0 0 0 1 -1 3 -1]
 [0 0 0 0 0 0 0 1 -2 3 0 1 1 -1]
 [0 0 0 0 0 0 0 1 -2 0 1 1 1 -1]
 [0 0 0 0 0 0 1 -1 1 0 2 0 -3]
 [0 0 0 0 0 0 1 -1 1 1 2], [-1 0 0 1 0 -1]
 [1 -1 -1 -1 1 -1 1 -2], [-1 0 0 1 0 -1]
```

```python
sage: J0(33).hecke_operator(2).matrix()
[-1 0 1 -1 1 -1]
[0 -2 1 0 1 -1]
[0 0 0 0 2 -2]
[-2 0 1 0 1 -1]
[-1 0 1 1 -1 -1]
[-1 0 0 1 0 -2]
```

`index_in(other, check=True)`

Return the index of self in other.

**INPUT:**

- `other` - another endomorphism subring of the same abelian variety
- `check` - bool (default: True); whether to do some type and other consistency checks

**EXAMPLES:**

```python
sage: R = J0(33).endomorphism_ring()
sage: R.index_in(R)
1
sage: J = J0(37) ; E = J.endomorphism_ring() ; T = E.image_of_hecke_algebra()
sage: T.index_in(E)
1
sage: J = J0(22) ; E = J.endomorphism_ring() ; T = E.image_of_hecke_algebra()
sage: T.index_in(E)
+Infinity
```

`index_in_saturation()`

Given a Hecke algebra T, compute its index in its saturation.

**EXAMPLES:**

```python
sage: End(J0(23)).image_of_hecke_algebra().index_in_saturation()
1
sage: End(J0(44)).image_of_hecke_algebra().index_in_saturation()
2
```

**class** `sage.modular.abvar.homspace.Homspace(domain, codomain, cat)`

**Bases:** `sage.categories.homset.HomsetWithBase`

A space of homomorphisms between two modular abelian varieties.
Element
alias of sage.modular.abvar.morphism.Morphism
calculate_generators()
If generators haven’t already been computed, calculate generators for this homspace. If they have been computed, do nothing.
EXAMPLES:
```
sage: E = End(J0(11))
sage: E.calculate_generators()
```
free_module()
Return this endomorphism ring as a free submodule of a big $\mathbb{Z}^{mn}$, where $n$ is the dimension of the domain abelian variety and $m$ the dimension of the codomain.
OUTPUT: free module
EXAMPLES:
```
sage: E = Hom(J0(11), J0(22))
sage: E.free_module()
Free module of degree 8 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1 0 -3 1 1 1 -1 -1]
[ 0 1 -3 1 1 1 -1 0]
```
gen(i=0)
Return i-th generator of self.
INPUT:
• i - an integer
OUTPUT: a morphism
EXAMPLES:
```
sage: E = End(J0(22))
sage: E.gen(0).matrix()
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```
gens()
Return tuple of generators for this endomorphism ring.
EXAMPLES:
```
sage: E = End(J0(22))
sage: E.gens()
(Abelian variety endomorphism of Abelian variety J0(22) of dimension 2,
Abelian variety endomorphism of Abelian variety J0(22) of dimension 2,
Abelian variety endomorphism of Abelian variety J0(22) of dimension 2,
Abelian variety endomorphism of Abelian variety J0(22) of dimension 2)
```
identity()
Return the identity endomorphism.
EXAMPLES:
sage: E = End(J0(11))
sage: E.identity()
Abelian variety endomorphism of Abelian variety J0(11) of dimension 1
sage: E.one()
Abelian variety endomorphism of Abelian variety J0(11) of dimension 1

sage: H = Hom(J0(11), J0(22))
sage: H.identity()
Traceback (most recent call last):
  ...
TypeError: the identity map is only defined for endomorphisms

matrix_space()
Return the underlying matrix space that we view this endomorphism ring as being embedded into.

EXAMPLES:

sage: E = End(J0(22))
sage: E.matrix_space()
Full MatrixSpace of 4 by 4 dense matrices over Integer Ring

ngens()
Return number of generators of self.

OUTPUT: integer

EXAMPLES:

sage: E = End(J0(22))
sage: E.ngens()
4
MORPHISMS BETWEEN MODULAR ABELIAN VARIETIES, INCLUDING HECKE OPERATORS ACTING ON MODULAR ABELIAN VARIETIES

Sage can compute with Hecke operators on modular abelian varieties. A Hecke operator is defined by given a modular abelian variety and an index. Given a Hecke operator, Sage can compute the characteristic polynomial, and the action of the Hecke operator on various homology groups.

AUTHORS:
• William Stein (2007-03)
• Craig Citro (2008-03)

EXAMPLES:

```python
sage: A = J0(54)
sage: t5 = A.hecke_operator(5); t5
Hecke operator T_5 on Abelian variety J0(54) of dimension 4
sage: t5.charpoly().factor()
(x - 3) * (x + 3) * x^2
sage: B = A.new_subvariety(); B
Abelian subvariety of dimension 2 of J0(54)
sage: t5 = B.hecke_operator(5); t5
Hecke operator T_5 on Abelian subvariety of dimension 2 of J0(54)
sage: t5.charpoly().factor()
(x - 3) * (x + 3)
sage: t5.action_on_homology().matrix()
[ 0 3 3 -3]
[-3 3 3 0]
[ 3 3 0 -3]
[-3 6 3 -3]
```

class sage.modular.abvar.morphism.DegeneracyMap (parent, A, t)

Bases: sage.modular.abvar.morphism.Morphism

Create the degeneracy map of index t in parent defined by the matrix A.

INPUT:
• `parent` - a space of homomorphisms of abelian varieties
• `A` - a matrix defining self
• `t` - a list of indices defining the degeneracy map

EXAMPLES:
```python
sage: J0(44).degeneracy_map(11,2)
Degeneracy map from Abelian variety J0(44) of dimension 4 to Abelian variety
J0(11) of dimension 1 defined by [2]
sage: J0(44)[0].degeneracy_map(88,2)
Degeneracy map from Simple abelian subvariety 11a(1,44) of dimension 1 of J0(44)
to Abelian variety J0(88) of dimension 9 defined by [2]
```

```python
t()
Return the list of indices defining self.

EXAMPLES:
```
```python
sage: J0(22).degeneracy_map(44).t()
[1]
sage: J = J0(22) + J0(11)
```sage
J.degeneracy_map([44,44], [2,1])
Degeneracy map from Abelian variety J0(22) x J0(11) of dimension 3 to Abelian
variety J0(44) x J0(44) of dimension 8 defined by [2, 1]
sage: J.degeneracy_map([44,44], [2,1]).t()
[2, 1]
```

```python
class sage.modular.abvar.morphism.HeckeOperator(abvar, n)
Bases: sage.modular.abvar.morphism.Morphism
```
A Hecke operator acting on a modular abelian variety.

```python
action_on_homology(R=Integer Ring)
Return the action of this Hecke operator on the homology $H_1(A; R)$ of this abelian variety with coefficients in $R$.

EXAMPLES:
```
```python
sage: A = J0(43)
sage: t2 = A.hecke_operator(2); t2
Hecke operator $T_2$ on Abelian variety J0(43) of dimension 3
```sage
sage: h2 = t2.action_on_homology(); h2
Hecke operator $T_2$ on Integral Homology of Abelian variety J0(43) of dimension 3
```sage
sage: h2.matrix()
[-2 1 0 0 0 0]
[-1 1 1 0 -1 0]
[-1 0 -1 2 -1 1]
[-1 0 1 1 -1 1]
[ 0 -2 0 2 -2 1]
[ 0 -1 0 1 0 -1]
sage: h2 = t2.action_on_homology(GF(2)); h2
Hecke operator $T_2$ on Homology with coefficients in Finite Field of size 2 of Abelian variety J0(43) of dimension 3
```sage
sage: h2.matrix()
[0 1 0 0 0 0]
[1 1 1 0 1 0]
[1 0 1 0 1 1]
[1 0 1 1 1 1]
[0 0 0 0 0 1]
[0 1 0 1 0 1]
```

```python
characteristic_polynomial(var='x')
Return the characteristic polynomial of this Hecke operator in the given variable.
```
INPUT:

- \texttt{var} - a string (default: ‘x’)

OUTPUT: a polynomial in \texttt{var} over the rational numbers.

EXAMPLES:

```python
sage: A = J0(43)[1]; A
Simple abelian subvariety 43b1 of dimension 2 of J0(43)

sage: t2 = A.hecke_operator(2); t2
Hecke operator T_2 on Simple abelian subvariety 43b1 of dimension 2 of J0(43)

sage: f = t2.characteristic_polynomial(); f
x^2 - 2

sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring

sage: f.factor()
x^2 - 2

sage: t2.characteristic_polynomial('y')
y^2 - 2
```

\texttt{charpoly}(\texttt{var}='x')

Synonym for \texttt{self.characteristic_polynomial(var)}.

INPUT:

- \texttt{var} - string (default: ‘x’)

EXAMPLES:

```python
sage: A = J1(13)

sage: t2 = A.hecke_operator(2); t2
Hecke operator T_2 on Abelian variety J1(13) of dimension 2

sage: f = t2.charpoly(); f
x^2 + 3*x + 3

sage: f.factor()
x^2 + 3*x + 3

sage: t2.charpoly('y')
y^2 + 3*y + 3
```

\texttt{index}()

Return the index of this Hecke operator. (For example, if this is the operator \(T_n\), then the index is the integer \(n\).)

OUTPUT:

- \texttt{n} - a (Sage) Integer

EXAMPLES:

```python
sage: J = J0(15)

sage: t = J.hecke_operator(53)
Hecke operator T_53 on Abelian variety J0(15) of dimension 1

sage: t.index()
53

sage: t = J.hecke_operator(54)

sage: t
Hecke operator T_54 on Abelian variety J0(15) of dimension 1
```

(continues on next page)
matrix()

Return the matrix of self acting on the homology \( H_1(A, \mathbb{Z}) \) of this abelian variety with coefficients in \( \mathbb{Z} \).

**EXAMPLES:**

```sage
sage: J0(47).hecke_operator(3).matrix()
[ 0 0 1 -2 1 0 -1 0]
[ 0 0 1 0 -1 0 0 0]
[-1 2 0 0 2 -2 1 -1]
[-2 1 1 -1 3 -1 -1 0]
[-1 -1 1 0 1 0 -1 1]
[-1 0 0 -1 2 0 -1 0]
[-1 -1 2 -2 2 0 -1 0]
[ 0 -1 0 0 1 0 -1 1]
```

```sage
sage: J0(11).hecke_operator(7).matrix()
[-2 0]
[ 0 -2]
```

```sage
sage: (J0(11) * J0(33)).hecke_operator(7).matrix()
[-2 0 0 0 0 0 0 0]
[ 0 -2 0 0 0 0 0 0]
[ 0 0 0 0 2 -2 2 -2]
[ 0 0 0 -2 2 0 -2 -2]
[ 0 0 0 0 2 0 4 -4]
[ 0 0 -4 0 2 2 2 -2]
[ 0 0 -2 0 2 2 0 -2]
[ 0 0 -2 0 0 2 0 -2]
```

```sage
sage: J0(23).hecke_operator(2).matrix()
[ 0 1 -1 0]
[ 0 1 -1 1]
[-1 2 -2 1]
[-1 1 0 -1]
```

n()

Alias for self.index().

**EXAMPLES:**

```sage
sage: J = J0(17)
sage: J.hecke_operator(5).n() 5
```

class sage.modular.abvar.morphism.Morphism

**Bases:**

sage.modular.abvar.morphism.Morphism_abstract, sage.modules.matrix_morphism.MatrixMorphism

84 Chapter 9. Morphisms between modular abelian varieties, including Hecke operators acting on modular abelian varieties
**restrict_domain** (*sub*)

Restrict self to the subvariety sub of self.domain().

**EXAMPLES:**

```python
sage: J = J0(37) ; A, B = J.decomposition()
sage: A.lattice().matrix()
[ 1 -1 1 0]
[ 0 0 2 -1]
sage: B.lattice().matrix()
[1 1 0]
[0 0 1]
sage: T = J.hecke_operator(2) ; T.matrix()
[-1 1 1 -1]
[1 -1 1 0]
[0 0 -2 1]
[0 0 0 0]
sage: T.restrict_domain(A)
Abelian variety morphism:
    From: Simple abelian subvariety 37a(1,37) of dimension 1 of J0(37)
    To: Abelian variety J0(37) of dimension 2
sage: T.restrict_domain(A).matrix()
[-2 2 -2 0]
[0 0 -4 2]
sage: T.restrict_domain(B)
Abelian variety morphism:
    From: Simple abelian subvariety 37b(1,37) of dimension 1 of J0(37)
    To: Abelian variety J0(37) of dimension 2
sage: T.restrict_domain(B).matrix()
[0 0 0 0]
[0 0 0 0]
```

class **sage.modular.abvar.morphism.Morphism_abstract** (*parent*)

Bases: *sage.modules.matrix_morphism.MatrixMorphism_abstract*

A morphism between modular abelian varieties.

**EXAMPLES:**

```python
sage: t = J0(11).hecke_operator(2)
sage: from sage.modular.abvar.morphism import Morphism
tsage: isinstance(t, Morphism)
True
```

c**okernel** ()

Return the cokernel of self.

**OUTPUT:**

- *A* - an abelian variety (the cokernel)
- *phi* - a quotient map from self.codomain() to the cokernel of self

**EXAMPLES:**

```python
sage: t = J0(33).hecke_operator(2)
sage: (t-1).cokernel()
(Abelian subvariety of dimension 1 of J0(33),
Abelian variety morphism:
    From: Abelian variety J0(33) of dimension 3
    To:    Abelian subvariety of dimension 1 of J0(33))
```
Projection will always have cokernel zero.

```
sage: J0(37).projection(J0(37)[0]).cokernel()
(Simple abelian subvariety of dimension 0 of J0(37),
Abelian variety morphism:
  From: Simple abelian subvariety 37a(1,37) of dimension 1 of J0(37)
  To: Simple abelian subvariety of dimension 0 of J0(37))
```

Here we have a nontrivial cokernel of a Hecke operator, as the \(T_2\)-eigenvalue for the newform 37b is 0.

```
sage: J0(37).hecke_operator(2).cokernel()
(Abelian subvariety of dimension 1 of J0(37),
Abelian variety morphism:
  From: Abelian variety J0(37) of dimension 2
  To: Abelian subvariety of dimension 1 of J0(37))
sage: AbelianVariety('37b').newform().q_expansion(5)
q + q^3 - 2*q^4 + O(q^5)
```

**complementary_isogeny()**

Returns the complementary isogeny of self.

EXAMPLES:

```
sage: J = J0(43)
sage: A = J[1]
sage: T5 = A.hecke_operator(5)
sage: T5.is_isogeny()
True
sage: T5.complementary_isogeny()
Abelian variety endomorphism of Simple abelian subvariety 43b(1,43) of dimension 2 of J0(43)
sage: (T5.complementary_isogeny() * T5).matrix()
[2 0 0 0]
[0 2 0 0]
[0 0 2 0]
[0 0 0 2]
```

**factor_out_component_group**

View self as a morphism \(f : A \rightarrow B\). Then \(\ker(f)\) is an extension of an abelian variety \(C\) by a finite component group \(G\). This function constructs a morphism \(g\) with domain \(A\) and codomain \(Q\) isogenous to \(C\) such that \(\ker(g)\) is equal to \(C\).

OUTPUT: a morphism

EXAMPLES:

```
sage: A,B,C = J0(33)
sage: pi = J0(33).projection(A)
sage: pi.kernel()
(Finite subgroup with invariants [5] over QQbar of Abelian variety J0(33) of dimension 3,
Abelian subvariety of dimension 2 of J0(33))
sage: psi = pi.factor_out_component_group()
sage: psi.kernel()
(Finite subgroup with invariants [] over QQbar of Abelian variety J0(33) of dimension 3,
Abelian subvariety of dimension 2 of J0(33))
```

ALGORITHM: We compute a subgroup \(G\) of \(B\) so that the composition \(h : A \rightarrow B \rightarrow B/G\) has kernel
that contains $A[n]$ and component group isomorphic to $(\mathbb{Z}/n\mathbb{Z})^{2d}$, where $d$ is the dimension of $A$. Then $h$ factors through multiplication by $n$, so there is a morphism $g : A \to B/G$ such that $g \circ [n] = h$. Then $g$ is the desired morphism. We give more details below about how to transform this into linear algebra.

**image()**
Return the image of this morphism.

**OUTPUT:** an abelian variety

**EXAMPLES:** We compute the image of projection onto a factor of $J_0(33)$:

```python
sage: A, B, C = J0(33)
sage: A
Simple abelian subvariety 11a(1,33) of dimension 1 of J0(33)
sage: f = J0(33).projection(A)
sage: f.image()
Abelian subvariety of dimension 1 of J0(33)
sage: f.image() == A
True
```

We compute the image of a Hecke operator:

```python
sage: t2 = J0(33).hecke_operator(2); t2.fcp()
(x - 1) * (x + 2)^2
sage: phi = t2 + 2
sage: phi.image()
Abelian subvariety of dimension 1 of J0(33)
```

The sum of the image and the kernel is the whole space:

```python
sage: phi.kernel()[1] + phi.image() == J0(33)
True
```

**is_isogeny()**
Return True if this morphism is an isogeny of abelian varieties.

**EXAMPLES:**

```python
sage: J = J0(39)
sage: Id = J.hecke_operator(1)
sage: Id.is_isogeny()
True
sage: J.hecke_operator(19).is_isogeny()
False
```

**kernel()**
Return the kernel of this morphism.

**OUTPUT:**
- **G** - a finite group
- **A** - an abelian variety (identity component of the kernel)

**EXAMPLES:** We compute the kernel of a projection map. Notice that the kernel has a nontrivial abelian variety part.

```python
sage: A, B, C = J0(33)
sage: pi = J0(33).projection(B)
sage: pi.kernel()
```
We compute the kernels of some Hecke operators:

```
sage: t2 = J0(33).hecke_operator(2)
sage: t2
Hecke operator T_2 on Abelian variety J0(33) of dimension 3
sage: t2.kernel()
(Finite subgroup with invariants [2, 2, 2, 2] over QQ of Abelian variety J0(33) of dimension 3,
Abelian subvariety of dimension 0 of J0(33))
sage: t3 = J0(33).hecke_operator(3)
sage: t3.kernel()
(Finite subgroup with invariants [3, 3] over QQ of Abelian variety J0(33) of dimension 3,
Abelian subvariety of dimension 0 of J0(33))
```
ABELIAN VARIETIES ATTACHED TO NEWFORMS

class sage.modular.abvar.abvar_newform.ModularAbelianVariety_newform(f, \text{internal\_name}='False')

Bases: sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract

A modular abelian variety attached to a specific newform.

\text{endomorphism\_ring}()

Return the endomorphism ring of this newform abelian variety.

\text{EXAMPLES:}

\begin{verbatim}
sage: A = AbelianVariety('23a')
sage: E = A.endomorphism_ring(); E
Endomorphism ring of Newform abelian subvariety 23a of dimension 2 of J0(23)
\end{verbatim}

We display the matrices of these two basis matrices:

\begin{verbatim}
sage: E.0.matrix()
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
sage: E.1.matrix()
[ 0 1 -1 0]
[ 0 1 -1 1]
[-1 2 -2 1]
[-1 1 0 -1]
\end{verbatim}

The result is cached:

\begin{verbatim}
sage: E is A.endomorphism_ring()
True
\end{verbatim}

\text{factor\_number}()

Return factor number.

\text{OUTPUT:}

\begin{verbatim}
int
\end{verbatim}

\text{EXAMPLES:}

\begin{verbatim}
sage: A = AbelianVariety('43b')
sage: \text{A.factor\_number}()
1
\end{verbatim}
**label()**
Return canonical label that defines this newform modular abelian variety.

**OUTPUT:**
string

**EXAMPLES:**
```
sage: A = AbelianVariety('43b')
sage: A.label()
'43b'
```

**newform(names=None)**
Return the newform that this modular abelian variety is attached to.

**EXAMPLES:**
```
sage: f = Newform('37a')
sage: A = f.abelian_variety()
sage: A.newform()
q - 2*q^2 - 3*q^3 + 2*q^4 - 2*q^5 + O(q^6)
sage: A.newform() is f
True
```

If the a variable name has not been specified, we must specify one:
```
sage: A = AbelianVariety('67b')
sage: A.newform()
Traceback (most recent call last):
  ...
TypeError: You must specify the name of the generator.
sage: A.newform('alpha')
q + alpha*q^2 + (-alpha - 3)*q^3 + (-3*alpha - 3)*q^4 - 3*q^5 + O(q^6)
```

If the eigenform is actually over $\mathbb{Q}$ then we don’t have to specify the name:
```
sage: A = AbelianVariety('67a')
sage: A.newform()
q + 2*q^2 - 2*q^3 + 2*q^4 + 2*q^5 + O(q^6)
```
L-SERIES OF MODULAR ABELIAN VARIETIES

AUTHOR:

- William Stein (2007-03)

```python
class sage.modular.abvar.lseries.Lseries(abvar)
    Bases: sage.structure.sage_object.SageObject

    Base class for $L$-series attached to modular abelian varieties.
    This is a common base class for complex and $p$-adic $L$-series of modular abelian varieties.

    abelian_variety()
        Return the abelian variety that this $L$-series is attached to.
        OUTPUT:
        a modular abelian variety
        EXAMPLES:

        sage: J0(11).padic_lseries(7).abelian_variety()
        Abelian variety J0(11) of dimension 1
```

```python
class sage.modular.abvar.lseries.Lseries_complex(abvar)
    Bases: sage.modular.abvar.lseries.Lseries

    A complex $L$-series attached to a modular abelian variety.

    EXAMPLES:

    sage: A = J0(37)
sage: A.lseries()
    Complex L-series attached to Abelian variety J0(37) of dimension 2
```

```python
lratio()
    Return the rational part of this $L$-function at the central critical value 1.
    OUTPUT:
    a rational number
    EXAMPLES:

    sage: A, B = J0(43).decomposition()
sage: A.lseries().rational_part()
    0
    sage: B.lseries().rational_part()
    2/7
```
**rational_part()**

Return the rational part of this \(L\)-function at the central critical value 1.

**OUTPUT:**

a rational number

**EXAMPLES:**

```python
sage: A, B = J0(43).decomposition()
sage: A.lseries().rational_part() 0
sage: B.lseries().rational_part() 2/7
```

**vanishes_at_1()**

Return True if \(L(1) = 0\) and return False otherwise.

**OUTPUT:**

a boolean

**EXAMPLES:**

Numerically, the \(L\)-series for \(J_0(389)\) appears to vanish at 1. This is confirmed by this algebraic computation:

```python
sage: L = J0(389)[0].lseries(); L
Complex L-series attached to Simple abelian subvariety 389a(1,389) of dimension 1 of J0(389)
sage: L(1)  # long time (2s) abstol 1e-10 -1.33139759782370e-19
sage: L.vanishes_at_1() True
```

Numerically, one might guess that the \(L\)-series for \(J_1(23)\) and \(J_1(31)\) vanish at 1. This algebraic computation shows otherwise:

```python
sage: L = J1(23).lseries(); L
Complex L-series attached to Abelian variety J1(23) of dimension 12
sage: abs(L(1, prec=100) - 0.00012951986142702571478817757148) < 1e-32  # long time (about 3 s)
True
sage: L.vanishes_at_1() False
```

```python
sage: L = J1(31).lseries(); L
Complex L-series attached to Abelian variety J1(31) of dimension 26
sage: abs(L(1) - 3.45014267547611e-7) < 1e-15  # long time (about 8 s)
True
sage: L.vanishes_at_1()  # long time (about 6 s)
False
```

**class** `sage.modular.abvar.lseries.Lseries_padic(abvar, p)`

**Bases:** `sage.modular.abvar.lseries.Lseries`

A \(p\)-adic \(L\)-series attached to a modular abelian variety.
**power_series** ($n=2, prec=5$)

Return the $n$-th approximation to this $p$-adic $L$-series as a power series in $T$.

Each coefficient is a $p$-adic number whose precision is provably correct.

NOTE: This is not yet implemented.

**EXAMPLES:**

```python
sage: L = J0(37)[0].padic_lseries(5)
sage: L.power_series()
Traceback (most recent call last):
  ...  
NotImplementedError
sage: L.power_series(3, 7)
Traceback (most recent call last):
  ...  
NotImplementedError
```

**prime**

Return the prime $p$ of this $p$-adic $L$-series.

**EXAMPLES:**

```python
sage: J0(11).padic_lseries(7).prime()
7
```
Chapter 11. $L$-series of modular abelian varieties
CHAPTER
TWELVE

INDICES AND TABLES

• Index
• Module Index
• Search Page
m
sage.modular.abvar.abvar, 3
sage.modular.abvar.abvar_ambient_jacobian, 43
sage.modular.abvar.abvar_newform, 89
sage.modular.abvar.constructor, 1
sage.modular.abvar.cuspidal_subgroup, 61
sage.modular.abvar.finite_subgroup, 47
sage.modular.abvar.homology, 65
sage.modular.abvar.homspace, 73
sage.modular.abvar.lseries, 91
sage.modular.abvar.morphism, 81
sage.modular.abvar.torsion_subgroup, 55
INDEX

A

abelian_variety() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 48
abelian_variety() (sage.modular.abvar.homology.Homology_abvar method), 66
abelian_variety() (sage.modular.abvar.homspace.EndomorphismSubring method), 76
abelian_variety() (sage.modular.abvar.series.Lseries method), 91
abelian_variety() (sage.modular.abvar.torsion_subgroup.QQbarTorsionSubgroup method), 56
AbelianVariety() (in module sage.modular.abvar.constructor), 1
action_on_homology() (sage.modular.abvar.morphism.HeckeOperator method), 82
ambient_hecke_module() (sage.modular.abvar.homology.Homology_abvar method), 66
ambient_hecke_module() (sage.modular.abvar.homology.Homology submodule method), 68
ambient_morphism() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 4
ambient_variety() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 5
ambient_variety() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 43

B

base_extend() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 5
base_field() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 6
brandt_module() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym method), 32

calculate_generators() (sage.modular.abvar.homspace.Homspace method), 78
change_ring() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 6
characteristic_polynomial() (sage.modular.abvar.morphism.HeckeOperator method), 82
charpoly() (sage.modular.abvar.morphism.HeckeOperator method), 83
cokernel() (sage.modular.abvar.morphism.Morphism abstract method), 85
complement() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 6
complementary_isogeny() (sage.modular.abvar.morphism.Morphism abstract method), 86
component_group_order() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym method), 32
conductor() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 6
cuspidal_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 7
CuspidalSubgroup (class in sage.modular.abvar.cuspidal_subgroup), 62
CuspidalSubgroup_generic (class in sage.modular.abvar.cuspidal_subgroup), 62

D
decomposition() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 7
decomposition() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 34
decomposition() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 43
degen_t() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 8
degeneracy_map() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 9
degeneracy_map() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 44
DegeneracyMap (class in sage.modular.abvar.morphism), 81
degree() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 9
dimension() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 10
dimension() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 45
direct_product() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 10
discriminant() (sage.modular.abvar.homspace.EndomorphismSubring method), 76
divisor_of_order() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 57
dual() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 10
E
Element (sage.modular.abvar.finite_subgroup.FiniteSubgroup attribute), 48
Element (sage.modular.abvar.homspace.Homspace attribute), 77
Element (sage.modular.abvar.torsion_subgroup.QQbarTorsionSubgroup attribute), 56
elliptic_curve() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 11
endomorphism_ring() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 12
endomorphism_ring() (sage.modular.abvar.abvar_newform.ModularAbelianVariety_newform method), 89
EndomorphismSubring (class in sage.modular.abvar.homspace), 75
exponent() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 49
F
factor_mmodsym_space_new_factors() (in module sage.modular.abvar.abvar), 38
factor_new_space() (in module sage.modular.abvar.abvar), 38
factor_number() (sage.modular.abvar.abvar_newform.ModularAbelianVariety_newform method), 89
factor_out_component_group() (sage.modular.abvar.morphism.Morphism_abstract method), 86
field_of_definition() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 49
field_of_definition() (sage.modular.abvar.torsion_subgroup.QQbarTorsionSubgroup method), 56
finite_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 12
FiniteSubgroup (class in sage.modular.abvar.finite_subgroup), 48
FiniteSubgroup_lattice (class in sage.modular.abvar.finite_subgroup), 53
free_module() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 13
free_module() (sage.modular.abvar.homology.Homology_abvar method), 66
free_module() (sage.modular.abvar.homology.Homology_submodule method), 69
free_module() (sage.modular.abvar.homspace.Homspace method), 78
frobenius_polynomial() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 13
G
gen() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 49
gen() (sage.modular.abvar.homology.Homology_abvar method), 66
gen() (sage.modular.abvar.homspace.Homspace method), 78
gens() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 49
gens() (sage.modular.abvar.homology.Homology_abvar method), 66
gens() (sage.modular.abvar.homspace.Homspace method), 78
group() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 34
group() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 45
groups() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 14
groups() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 35
groups() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 45

H
hecke_bound() (sage.modular.abvar.homology.Homology_abvar method), 67
hecke_bound() (sage.modular.abvar.homology.Homology submodule method), 69
hecke_matrix() (sage.modular.abvar.homology.Homology_abvar method), 67
hecke_matrix() (sage.modular.abvar.homology.Homology_over_base method), 68
hecke_matrix() (sage.modular.abvar.homology.Homology submodule method), 69
hecke_matrix() (sage.modular.abvar.homology.IntegralHomology method), 70
hecke_matrix() (sage.modular.abvar.homology.RationalHomology method), 71
hecke_operator() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 14
hecke_polynomial() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 15
hecke_polynomial() (sage.modular.abvar.homology.Homology method), 65
hecke_polynomial() (sage.modular.abvar.homology.IntegralHomology method), 70
hecke_polynomial() (sage.modular.abvar.homology.RationalHomology method), 71
HeckeOperator (class in sage.modular.abvar.morphism), 82
Homology (class in sage.modular.abvar.homology), 65
homology() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 15
Homology_abvar (class in sage.modular.abvar.homology), 66
Homology_over_base (class in sage.modular.abvar.homology), 68
Homology submodule (class in sage.modular.abvar.homology), 68
Homspace (class in sage.modular.abvar.homspace), 77

I
identity() (sage.modular.abvar.homspace.Homspace method), 78
image() (sage.modular.abvar.morphism.Morphism_abstract method), 87
image_of_hecke_algebra() (sage.modular.abvar.homology.EndomorphismSubring method), 76
in_same_ambient_variety() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 16
index() (sage.modular.abvar.morphism.HeckeOperator method), 83
index_in() (sage.modular.abvar.homspace.EndomorphismSubring method), 77
index_in_saturation() (sage.modular.abvar.homspace.EndomorphismSubring method), 77
integral_homology() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 16
IntegralHomology (class in sage.modular.abvar.homology), 70
intersection() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 16
intersection() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 50
invariants() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 51
is_ambient() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 18
is_ambient() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 35
is_hecke_stable() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 19
is_isogeny() (sage.modular.abvar.morphism.Morphism_abstract method), 87
is_J0() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 18
is_J1() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 18
is_ModularAbelianVariety() (in module sage.modular.abvar.abvar), 38
is_rational_cusp_gamma0() (in module sage.modular.abvar.cuspidal_subgroup), 63
is_simple() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 19
is_subgroup() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 51
is_subvariety() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 19
is_subvariety() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 35
is_subvariety_of_ambient_jacobian() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 19
isogeny_number() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 20

J
J0() (in module sage.modular.abvar.constructor), 1
J1() (in module sage.modular.abvar.constructor), 1
JH() (in module sage.modular.abvar.constructor), 2

K
kernel() (sage.modular.abvar.morphism.Morphism_abstract method), 87

L
label() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 20
label() (sage.modular.abvar.abvar_newform.ModularAbelianVariety_newform method), 89
lattice() (sage.modular.abvar.abvar.ModularAbelianVariety method), 3
lattice() (sage.modular.abvar.abvar_modsym.ModularAbelianVariety_abstract method), 21
lattice() (sage.modular.abvar.cuspidal_subgroup.CuspidalSubgroup method), 62
lattice() (sage.modular.abvar.cuspidal_subgroup.RationalCuspidalSubgroup method), 63
lattice() (sage.modular.abvar.cuspidal_subgroup.RationalCuspSubgroup method), 62
lattice() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 52
lattice() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 57
level() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 21
lratio() (sage.modular.abvar.lseries.Lseries_complex method), 91
Lseries (class in sage.modular.abvar.lseries), 91
lseries() (sage.modular.abvar.ModularAbelianVariety_abstract method), 22
Lseries_complex (class in sage.modular.abvar.lseries), 91
Lseries_padic (class in sage.modular.abvar.lseries), 92

M
matrix() (sage.modular.abvar.morphism.HeckeOperator method), 84
matrix_space() (sage.modular.abvar.homspace.Homspace method), 79
ModAbVar_ambient_jacobian() (in module sage.modular.abvar.var_ambient_jacobian), 43
ModAbVar_ambient_jacobian_class (class in sage.modular.abvar.var_ambient_jacobian), 43
modsym_lattices() (in module sage.modular.abvar.var), 39
modular_degree() (sage.modular.abvar.ModularAbelianVariety_abstract method), 22
modular.kernel() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 22
modular_symbols() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 36
ModularAbelianVariety (class in sage.modular.abvar.abvar), 3
ModularAbelianVariety_abstract (class in sage.modular.abvar.abvar), 3
ModularAbelianVariety_modsym (class in sage.modular.abvar.abvar), 31
ModularAbelianVariety_modsym_abstract (class in sage.modular.abvar.abvar), 34
ModularAbelianVariety_newform (class in sage.modular.abvar.var_newform), 89
module
sage.modular.abvar.abvar, 3
sage.modular.abvar.abvar_ambient_jacobian, 43
sage.modular.abvar.abvar_newform, 89
sage.modular.abvar.constructor, 1
sage.modular.abvar.cuspidal_subgroup, 61
sage.modular.abvar.finite_subgroup, 47
sage.modular.abvar.homology, 65
sage.modular.abvar.homspace, 73
sage.modular.abvar.lseries, 91
sage.modular.abvar.morphism, 81
sage.modular.abvar.torsion_subgroup, 55

Morphism (class in sage.modular.abvar.morphism), 84
Morphism_abstract (class in sage.modular.abvar.morphism), 85

multiple_of_order() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 57
multiple_of_order_using_frobp() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 58

N
n() (sage.modular.abvar.morphism.HeckeOperator method), 84
new_subvariety() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 37
newform() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 22
newform() (sage.modular.abvar.abvar_newform.ModularAbelianVariety_abstract method), 90
newform_decomposition() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 23
newform_decomposition() (sage.modular.abvar.abvar_ambient_jacobian.ModAbVar_ambient_jacobian_class method), 45
newform_label() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 23
newform_level() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 23
ngens() (sage.modular.abvar.homspace.Homspace method), 79
number_of_rational_points() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 24

O
old_subvariety() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym_abstract method), 37
order() (sage.modular.abvar.finite_subgroup.FiniteSubgroup method), 52
order() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 59

P
padic_lseries() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 24
possible_orders() (sage.modular.abvar.torsion_subgroup.RationalTorsionSubgroup method), 60
power_series() (sage.modular.abvar.lseries.Lseries_padic method), 92
prime() (sage.modular.abvar.lseries.Lseries_padic method), 93
project_to_factor() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 24
projection() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 25

Q
qbar_torsion_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 26
QQbarTorsionSubgroup (class in sage.modular.abvar.torsion_subgroup), 56
quotient() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 26

R
random_hecke_operator() (in module sage.modular.abvar.abvar), 39
rank() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 27
rank() (sage.modular.abvar.homology.Homology_abvar method), 67
rank() (sage.modular.abvar.homology.Homology submodule method), 70
rational_cusp_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 27
rational_cuspidal_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 28
rational_homology() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 28
rational_part() (sage.modular.abvar.lseries.Lseries_complex method), 91
rational_torsion_order() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 29
rational_torsion_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 29
RationalCuspidalSubgroup (class in sage.modular.abvar.cuspidal_subgroup), 63
RationalCuspSubgroup (class in sage.modular.abvar.cuspidal_subgroup), 62
RationalHomology (class in sage.modular.abvar.homology), 70
RationalTorsionSubgroup (class in sage.modular.abvar.torsion_subgroup), 56
restrict_domain() (sage.modular.abvar.morphism.Morphism method), 84

S
sage.modular.abvar.abvar
    module, 3
sage.modular.abvar.abvar_ambient_jacobian
    module, 43
sage.modular.abvar.abvar_newform
    module, 89
sage.modular.abvar.constructor
    module, 1
sage.modular.abvar.cuspidal_subgroup
    module, 61
sage.modular.abvar.finite_subgroup
    module, 47
sage.modular.abvar.homology
    module, 65
sage.modular.abvar.homspace
    module, 73
sage.modular.abvar.lseries
    module, 91
sage.modular.abvar.morphism
    module, 81
sage.modular.abvar.torsion_subgroup
    module, 55
shimura_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 29
simple_factorization_of_modsym_space() (in module sage.modular.abvar.abvar), 40
sqrt_poly() (in module sage.modular.abvar.abvar), 40
sturm_bound() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 30
subgroup() (sage.modular.abvar.abvar.finite_subgroup.FiniteSubgroup method), 52
submodule() (sage.modular.abvar.homology.Homology_abvar method), 67

T
t() (sage.modular.abvar.morphism.DegeneracyMap method), 82
tamagawa_number() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym method), 33
tamagawa_number_bounds() (sage.modular.abvar.abvar.ModularAbelianVariety_modsym method), 33
torsion_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 30

V
vanishes_at_1() (sage.modular.abvar.lseries.Lseries_complex method), 92
vector_space() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 31

Z

zero_subgroup() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 31
zero_subvariety() (sage.modular.abvar.abvar.ModularAbelianVariety_abstract method), 31