Drinfeld modular forms

Release 10.10

The Sage Development Team

Oct 28, 2026

CONTENTS

1 Congruence subgroups 1

2 Drinfeld modular forms 7

3 Indices and Tables 29

Python Module Index 31

Index 33

CHAPTER

ONE

CONGRUENCE SUBGROUPS

1.1 Congruence subgroups $\Gamma_0(N)$ of $\mathrm{GL}_2(\mathbb{F}_q[T])$.

By definition, it is the subgroup of matrices whose bottom left coefficient is divislble by N.

AUTHORS:

- Cécile Armana, Xavier Caruso (2026-02): initial version

class sage.modular.drinfeld_modform.congroup_gamma0.**Gamma0Element**(*parent*, *elt*)

Bases: MultiplicativeGroupElement

An element in a congruence subgroup.

is_one()

Return whether this matrix is the identity matrix.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: G = Gamma0(T^4 + 2*T + 3)
sage: g = G.an_element()
sage: g
[4*T^4 + 3*T + 3 4*T^4 + 3*T + 2]
[  T^4 + 2*T + 3   T^4 + 2*T + 4]
sage: g.is_one()
False
sage: h = g * g^(-1)
sage: h.is_one()
True
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> G = Gamma0(T**Integer(4) + Integer(2)*T + Integer(3))
>>> g = G.an_element()
>>> g
[4*T^4 + 3*T + 3 4*T^4 + 3*T + 2]
[  T^4 + 2*T + 3   T^4 + 2*T + 4]
>>> g.is_one()
False
>>> h = g * g**(-Integer(1))
>>> h.is_one()
True
```

level()

Return the level of the congruence subgroup in which this element lives.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: G = Gamma0(T^4 + 2*T + 3)
sage: g = G.an_element()
sage: g.level()
T^4 + 2*T + 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> G = Gamma0(T**Integer(4) + Integer(2)*T + Integer(3))
>>> g = G.an_element()
>>> g.level()
T^4 + 2*T + 3
```

class sage.modular.drinfeld_modform.congroup_gamma0.**Gamma0_drinfeld**(*base*, *level*)

Bases: Group, UniqueRepresentation

A congruence subgroup.

Element

alias of *Gamma0Element*

base_ring()

Return the base ring of this congruence subgroup.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T^4 + 2*T + 3).base_ring()
Univariate Polynomial Ring in T over Finite Field of size 5
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> Gamma0(T**Integer(4) + Integer(2)*T + Integer(3)).base_ring()
Univariate Polynomial Ring in T over Finite Field of size 5
```

genus()

" Return the genus of this congruence subgroup, i.e. the genus of the attached Drinfeld modular curve.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T - 1).genus()
0
sage: Gamma0(T^3 + 1).genus()
5
sage: Gamma0(T^5 - 3*T^4 + 1).genus()
155
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```
>>> Gamma0(T - Integer(1)).genus()
0
>>> Gamma0(T**Integer(3) + Integer(1)).genus()
5
>>> Gamma0(T**Integer(5) - Integer(3)*T**Integer(4) + Integer(1)).genus()
155
```

If the level N is irreducible, the formula simplifies as follows:

```
sage: def genus_irr(N):
....:     q = N.base_ring().cardinality()
....:     d = N.degree()
....:     if is_even(d):
....:         g = (q^d - q^2) / (q^2 - 1)
....:     else:
....:         g = (q^d - q) / (q^2 - 1)
....:     return g

sage: N = T^6 + T^4 + 4*T^3 + T^2 + 2
sage: N.is_irreducible()
True
sage: Gamma0(N).genus() == genus_irr(N)
True

sage: N = T^7 + 3*T + 3
sage: N.is_irreducible()
True
sage: Gamma0(N).genus() == genus_irr(N)
True
```

```
>>> from sage.all import *
>>> def genus_irr(N):
...     q = N.base_ring().cardinality()
...     d = N.degree()
...     if is_even(d):
...         g = (q**d - q**Integer(2)) / (q**Integer(2) - Integer(1))
...     else:
...         g = (q**d - q) / (q**Integer(2) - Integer(1))
...     return g

>>> N = T**Integer(6) + T**Integer(4) + Integer(4)*T**Integer(3) +␣
↪T**Integer(2) + Integer(2)
>>> N.is_irreducible()
True
>>> Gamma0(N).genus() == genus_irr(N)
True

>>> N = T**Integer(7) + Integer(3)*T + Integer(3)
>>> N.is_irreducible()
True
>>> Gamma0(N).genus() == genus_irr(N)
True
```

REFERENCE:

[Gek2001], Theorem 8.1

index()

Return the index of this congruence subgroup as a subgroup of $\mathrm{GL}_2(A)$ (where A is the base ring).

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T^4 - 3*T^2 + 1).index()
900
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> Gamma0(T**Integer(4) - Integer(3)*T**Integer(2) + Integer(1)).index()
900
```

If the level N is irreducible, the index is $1 + q^{\deg(N)}$. We check it on an example below:

```
sage: N = T^4 + 4*T^2 + 4*T + 2
sage: N.is_irreducible()
True
sage: q = A.base_ring().cardinality()
sage: Gamma0(N).index() == 1 + q^4
True
```

```
>>> from sage.all import *
>>> N = T**Integer(4) + Integer(4)*T**Integer(2) + Integer(4)*T + Integer(2)
>>> N.is_irreducible()
True
>>> q = A.base_ring().cardinality()
>>> Gamma0(N).index() == Integer(1) + q**Integer(4)
True
```

level()

Return the level of this congruence subgroup.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T^4 + 2*T + 3).level()
T^4 + 2*T + 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> Gamma0(T**Integer(4) + Integer(2)*T + Integer(3)).level()
T^4 + 2*T + 3
```

matrix_space()

Return the matrix space of this congruence subgroup.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T^4 + 2*T + 3).matrix_space()
Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial
Ring in T over Finite Field of size 5
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> Gamma0(T**Integer(4) + Integer(2)*T + Integer(3)).matrix_space()
Full MatrixSpace of 2 by 2 dense matrices over Univariate Polynomial
Ring in T over Finite Field of size 5
```

ncusps()

Return the number of cusps of this congruence subgroup, i.e. the number of cusps of the attached Drinfeld modular curve.

EXAMPLES:

```
sage: A.<T> = GF(5)[]
sage: Gamma0(T - 1).ncusps()
2
sage: Gamma0(T^3 + 1).ncusps()
4
sage: Gamma0(T^7 - 3*T^4 + 1).ncusps()
8
```

```
>>> from sage.all import *
>>> A = GF(Integer(5))['T']; (T,) = A._first_ngens(1)
>>> Gamma0(T - Integer(1)).ncusps()
2
>>> Gamma0(T**Integer(3) + Integer(1)).ncusps()
4
>>> Gamma0(T**Integer(7) - Integer(3)*T**Integer(4) + Integer(1)).ncusps()
8
```

If the level N is irreducible, the number of cusps is 2. We check it below:

```
sage: N = T^6 + T^4 + 4*T^3 + T^2 + 2
sage: N.is_irreducible()
True
sage: Gamma0(N).ncusps() == 2
True
```

```
>>> from sage.all import *
>>> N = T**Integer(6) + T**Integer(4) + Integer(4)*T**Integer(3) +␣
↪T**Integer(2) + Integer(2)
>>> N.is_irreducible()
True
>>> Gamma0(N).ncusps() == Integer(2)
True
```

```
sage: N = T^7 + 3*T + 3
sage: N.is_irreducible()
```

(continues on next page)

(continued from previous page)

```
True
sage: Gamma0(N).ncusps() == 2
True
```

```
>>> from sage.all import *
>>> N = T**Integer(7) + Integer(3)*T + Integer(3)
>>> N.is_irreducible()
True
>>> Gamma0(N).ncusps() == Integer(2)
True
```

REFERENCE:

[Gek2001], Proposition 6.7

class sage.modular.drinfeld_modform.congroup_gamma0.**InclusionIntoMatrixSpace**(*parent*)

Bases: Map

Inclusion of a congruence subgroup into the corresponding matrix space.

CHAPTER

TWO

DRINFELD MODULAR FORMS

2.1 Introduction to Drinfeld modular forms

This tutorial outlines the definitions, the notations, and the implementation of Drinfeld modular forms in SageMath. We assume that the reader has basic knowledge of classical modular forms, as we will often make analogies to this setting. We also assume little knowledge of Drinfeld modules; for this topic, the interested reader can consult the SageMath reference manual Drinfeld modules.

Preliminary notations

Let q be a prime power and let A be the ring of functions of $\mathbb{P}^1/\mathbb{F}_q$ which are regular outside a closed point ∞. This ring is the polynomial ring $\mathbb{F}_q[T]$. We denote by $K := \mathbb{F}_q(T)$ rational function field. We endow K with the $1/T$-adic valuation and let $K_\infty := \mathbb{F}_q((1/T))$ be the completion of K. Next, we define $\mathbb{C}_\infty$ to be the completion of an algebraic closure of K_∞. Lastly, we denote by $\tau : x \mapsto x^q$ the q-Frobenius.

Note

The above construction of $\mathbb{C}_\infty$ is the same as the construction of $\mathbb{C}_p$ in the case of p-adic numbers (see Wikipedia article P-adic_number#Algebraic_closure).

In SageMath, we create the rational function field by first creating a univariate polynomial ring over $\mathbb{F}_q$ and, following this, by constructing its field of fractions:

```
sage: A = GF(3)['T']
sage: K.<T> = Frac(A)
sage: K
Fraction Field of Univariate Polynomial Ring in T over Finite Field of size 3
sage: K.base()  # returns A
Univariate Polynomial Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> K
Fraction Field of Univariate Polynomial Ring in T over Finite Field of size 3
>>> K.base()  # returns A
Univariate Polynomial Ring in T over Finite Field of size 3
```

Drinfeld period domain and action of $\mathrm{GL}_r(K_\infty)$

In the classical setting, the domain of any modular form is the complex upper half plane $\mathcal{H} := \{w \in \mathbb{C} : \mathrm{im}(w) > 0\}$. The analogue of this plane in the function field setting is the *Drinfeld period domain of rank* $r > 1$ and it is defined by

$$\Omega^r(\mathbb{C}_\infty) := \mathbb{P}^{r-1}(\mathbb{C}_\infty) \setminus \{K_\infty\text{-rational hyperplanes}\}.$$

This space is a rigid analytic space and, after fixing an arbitrary nonzero constant ξ in $\mathbb{C}_\infty$, we identify its elements with the set of column vectors $(w_1, \ldots, w_{r-1}, w_r)^{\mathrm{T}}$ in $\mathbb{C}_\infty^r$ such that the w_i are K_∞-linearly independant and $w_r = \xi$. Note that ξ is unspecified, but the reader can assume that $\xi = 1$ without any loss of significant information. Its value can be interesting simply for normalization purposes.

We define a left action of $\mathrm{GL}_r(K_\infty)$ on $\Omega^r(\mathbb{C}_\infty)$ by setting

$$\gamma(w) := j(\gamma, w)^{-1}\gamma w$$

where $j(\gamma, w) := \xi^{-1} \cdot$ (last entry of γw).

Universal Drinfeld module over $\Omega^r(\mathbb{C}_\infty)$

For any $w = (w_1, \ldots, w_{r-1}, \xi)$ in $\Omega^r(\mathbb{C}_\infty)$ we have a corresponding discrete A-module Λ^w which is free of rank r:

$$\Lambda^w := Aw_1 \oplus \cdots \oplus Aw_{r-1} \oplus A\xi.$$

An important result is that we have analytic uniformization which is the analogue of complex uniformization for elliptic curves. In our setting, elliptic curves are replaced by Drinfeld modules. In short, there exists a corresponding Drinfeld module

$$\phi^w : T \mapsto T + g_1(w)\tau + \cdots + g_{r-1}(w)\tau^{r-1} + g_r(w)\tau^r.$$

such that the exponential of ϕ^w induces an isomorphism (of abelian group) between the additive group $\mathbb{C}_\infty$ and the quotient $\mathbb{C}_\infty/\Lambda^w$. Background material on Drinfeld modules and their analytic uniformization can be found in section 4.3 and 4.6 of [Gos1998].

The Drinfeld module $\phi^w : A \to \mathbb{C}_\infty\{\tau\}$ is called the *universal Drinfeld* $\mathbb{F}_q[T]$*-module over* $\Omega^r(\mathbb{C}_\infty)$ and its coefficients $g_i : \Omega^r(\mathbb{C}_\infty) \to \mathbb{C}_\infty$ are rigid analytic functions satisfying the *invariance property*:

$$g_i(\gamma(w)) = j(\gamma, w)^{1-q^i} g_i(w),\ \forall \gamma \in \mathrm{GL}_r(A)$$

where $g_r(w)$ never vanishes. Moreover, these coefficients g_i admits an expansion at infinity, analogous to q-expansion principle for classical modular forms. The functions g_i are known as the *coefficients forms at* T. More generally, the coefficients of the image ϕ_a^w for any $a \in A$ are called the *coefficient forms at* a and they are an algebraic combination of the coefficient forms at T.

In the rank two case, the expansion at infinity is of the form

$$g_i(w) = \sum_{i=0}^{\infty} a_n(g_i)u(w)^i$$

where $u(w) := e(w)^{-1}$ and e is the exponential function of the Carlitz module $\rho : T \mapsto T + \tau$. The analytic parameter u is called the *parameter at infinity*.

A *Drinfeld modular form* of rank r, weight k, type m for $\mathrm{GL}_r(A)$ is a rigid analytic function

$$f : \Omega^r(\mathbb{C}_\infty) \to \mathbb{C}_\infty$$

such that

- $f(\gamma(w)) = \det(\gamma)^m j(\gamma, w)^k f(w)$ for all γ in $\mathrm{GL}_r(A)$ and $w \in \Omega^r(\mathbb{C}_\infty)$;
- f is holomorphic at infinity.

Without diving into the details, we mention that the second condition is similar to the classical case. More specifically, in the rank two situation, the expansion of f is given by a power series in u $f = \sum_{n\geq 0} a_n(f)u^n$ where $a_n(f) \in \mathbb{C}_\infty$.

Lastly, we also mention that the integer m only depends on its class modulo $q-1$.

Note that all the above theory is covered in much greater details in part I of [BRP2018].

Ring of Drinfeld modular forms

Letting $M_k^{r,m}(\mathrm{GL}_r(A))$ denote the space of rank r, weight $k \in (q-1)\mathbb{Z}$ and type m Drinfeld modular forms, we define

$$M^{r,0}(\mathrm{GL}_r(A)) := \bigoplus_{k \in ZZ} M_k^{r,0}(\mathrm{GL}_r(A))$$

to be the graded ring of all Drinfeld modular forms of type 0. The graduation is given by the weight of a modular form. Similarly, we let $M^r(\mathrm{GL}_r(A)) \supset M^{r,0}(\mathrm{GL}_r(A))$ be the ring of all Drinfeld modular forms of rank r and arbitrary type. By theorem 17.5 in part III of [BRP2018], we have

$$M^{r,0}(\mathrm{GL}_r(A)) = \mathbb{C}_\infty[g_1, \ldots, g_{r-1}, g_r].$$

and

$$M^r(\mathrm{GL}_r(A)) = \mathbb{C}_\infty[g_1, \ldots, g_{r-1}, h_r].$$

where h_r is a weight $(q^r-1)/(q-1)$ modular forms of type 1 which is a $(q-1)$-root of g_r sometimes known as *Gekeler's h function*, see theorem 3.8 of [Gek2017] for the precise definition of this function.

SageMath implementation

In SageMath, we model the ring of type 0 Drinfeld modular forms over K as a finitely generated ring in the coefficients forms g_i:

$$K[g_1, \ldots, g_{r-1}, g_r].$$

Hence, any ring element is seen as a formal algebraic combination of the coefficient forms g_i over K. Likewise, the ring of arbitrary type forms is generated by $g_1 \ldots, g_{r-1}, h_r$.

To create the ring of type zero and rank r Drinfeld modular forms, one uses the class *DrinfeldModularForms*:

```
sage: A = GF(3)['T']
sage: K.<T> = Frac(A)
sage: M = DrinfeldModularForms(K, 3)  # rank 3
sage: M
Ring of Drinfeld modular forms of rank 3 over Fraction Field of Univariate Polynomial␣
↪Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> M = DrinfeldModularForms(K, Integer(3))  # rank 3
>>> M
Ring of Drinfeld modular forms of rank 3 over Fraction Field of Univariate Polynomial␣
↪Ring in T over Finite Field of size 3
```

To create the ring of arbitrary types modular forms, one passes the keyword argument `has_type=True`:

```
sage: M = DrinfeldModularForms(K, 4, has_type=True)
sage: M.gens()
(g1, g2, g3, h4)
sage: h4 = M.3
sage: h4.weight()
40
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, Integer(4), has_type=True)
>>> M.gens()
(g1, g2, g3, h4)
>>> h4 = M.gen(3)
>>> h4.weight()
40
```

For more information about the functionalities of the implementation, one should consult the documentation of the main classes:

- Parent class: *DrinfeldModularForms*
- Element class: *DrinfeldModularFormsElement*

References

A good introduction to Drinfeld modular forms of rank 2, see Gekeler's paper [Gek1988]. See also [BRP2018] for a detailed exposition of the arbitrary rank theory.

2.2 Graded rings of Drinfeld modular forms

This module defines a class named *DrinfeldModularForms*. Currently, the implementation only supports the full modular group $\mathrm{GL}_r(A)$ where $A = \mathbb{F}_q[T]$.

The implementation is based on the following identification:

$$M^r(\mathrm{GL}_r(A)) = \mathbb{C}_\infty[g_1, \ldots, g_{r-1}, g_r].$$

where g_i is the i-th coefficient form of weight $q^i - 1$.

AUTHORS:

- David Ayotte (2022): initial version

class sage.modular.drinfeld_modform.ring.**DrinfeldModularForms**(*base_ring*, *rank*, *group*, *has_type*, *names*)

Bases: `Parent`, `UniqueRepresentation`

Base class for the graded ring of Drinfeld modular forms.

If $K = \mathrm{Frac}(A)$ where $A = \mathbb{F}_q[T]$, then the ring of Drinfeld modular forms over K of rank r and type zero for $\mathrm{GL}_r(A)$ is

$$M^{r,0}(\mathrm{GL}_r(A)) = K[g_1, \ldots, g_{r-1}, g_r].$$

where g_i the i-th coefficient form of weight $q^i - 1$ at T.

Similarly, the ring of Drinfeld modular forms over K of rank r and arbitrary type is

$$M^r(\mathrm{GL}_r(A)) = K[g_1, \ldots, g_{r-1}, h_r].$$

where h_r is a form of weight $(q^r - 1)/(q - 1)$ and type 1.

We will see the elements of this ring as formal objects given by algebraic combination of the generator of the ring. See the class *DrinfeldModularFormsElement* for more details about their implementation.

INPUT:

- `base_ring` – the fraction field of a univariate polynomial ring over $\mathbb{F}_q$
- `rank` integer (default: `None`); the rank of the ring. If the rank is `None`, then the names of the generators must be specified.
- `group` – (not implemented, default: `None`) the group of the ring. The current implementation only supports the full modular group $\mathrm{GL}_r(A)$.
- `has_type` – boolean (default: `False`); if set to `True`, returns the graded ring of arbitrary type
- `names` – string, tuple or list (default: `None`); a single character, a tuple or list of character, or comma separated string of character representing the names of the generators. If this parameter is set to `None` and the rank is specified, then the default names for the generators will be:
  - `g1, g2, ..., gr` for the type zero forms
  - `g1, g2, ..., hr` for the arbitrary type forms.

  If this parameter is a single character, for example `f`, and a rank is specified, then the names will be of the form `f1, f2, ..., fr`. Finally, if this parameter is a list, a tuple or a string of comma separated characters, then each character will corresponds to a generator. Note that in this case, it not necessary to specify the rank.

EXAMPLES:

```
sage: q = 3
sage: A = GF(q)['T']
sage: K.<T> = Frac(A)
sage: M = DrinfeldModularForms(K, 3)
sage: M
Ring of Drinfeld modular forms of rank 3 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> q = Integer(3)
>>> A = GF(q)['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> M = DrinfeldModularForms(K, Integer(3))
>>> M
Ring of Drinfeld modular forms of rank 3 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

Use the *gens()* method to obtain the generators of the ring:

```
sage: M.gens()
(g1, g2, g3)
sage: M.inject_variables()  # assign the variable g1, g2, g3
Defining g1, g2, g3
sage: T*g1*g2 + g3
g3 + T*g1*g2
```

```
>>> from sage.all import *
>>> M.gens()
(g1, g2, g3)
>>> M.inject_variables()  # assign the variable g1, g2, g3
Defining g1, g2, g3
>>> T*g1*g2 + g3
g3 + T*g1*g2
```

When creating the ring, one can name the generators in various ways:

```
sage: M.<F, G, H> = DrinfeldModularForms(K)
sage: M.gens()
(F, G, H)
sage: M = DrinfeldModularForms(K, 5, names='f')  # must specify the rank
sage: M.gens()
(f1, f2, f3, f4, f5)
sage: M = DrinfeldModularForms(K, names='u, v, w, x')
sage: M.gens()
(u, v, w, x)
sage: M = DrinfeldModularForms(K, names=['F', 'G', 'H'])
sage: M.gens()
(F, G, H)
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, names=('F', 'G', 'H',)); (F, G, H,) = M._first_
↪ngens(3)
>>> M.gens()
(F, G, H)
>>> M = DrinfeldModularForms(K, Integer(5), names='f')  # must specify the rank
>>> M.gens()
(f1, f2, f3, f4, f5)
>>> M = DrinfeldModularForms(K, names='u, v, w, x')
>>> M.gens()
(u, v, w, x)
>>> M = DrinfeldModularForms(K, names=['F', 'G', 'H'])
>>> M.gens()
(F, G, H)
```

Set the keyword parameter `has_type` to `True` in order to create the ring of Drinfeld modular forms of arbitrary type:

```
sage: M = DrinfeldModularForms(K, 4, has_type=True)
sage: M.gens()
(g1, g2, g3, h4)
sage: h4 = M.3
sage: h4.type()
1
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, Integer(4), has_type=True)
>>> M.gens()
(g1, g2, g3, h4)
>>> h4 = M.gen(3)
```

(continues on next page)

(continued from previous page)

```
>>> h4.type()
1
```

To obtain a generating set of the subspace of forms of a fixed weight, use the method *basis_of_weight()*:

```
sage: M = DrinfeldModularForms(K, 2)
sage: M.basis_of_weight(q^3 - 1)
[g1*g2^3, g1^5*g2^2, g1^9*g2, g1^13]
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.basis_of_weight(q**Integer(3) - Integer(1))
[g1*g2^3, g1^5*g2^2, g1^9*g2, g1^13]
```

In order to compute the coefficient forms, use the methods *coefficient_form()* and *coefficient_forms()*:

```
sage: M = DrinfeldModularForms(K, 3)
sage: M.coefficient_form(1)
g1
sage: M.coefficient_form(2)
g2
sage: M.coefficient_form(3)
g3
sage: M.coefficient_forms(T)
[g1, g2, g3]
sage: M.coefficient_forms(T^2)
[(T^3 + T)*g1,
 g1^4 + (T^9 + T)*g2,
 g1^9*g2 + g1*g2^3 + (T^27 + T)*g3,
 g1^27*g3 + g1*g3^3 + g2^10,
 g2^27*g3 + g2*g3^9,
 g3^28]
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, Integer(3))
>>> M.coefficient_form(Integer(1))
g1
>>> M.coefficient_form(Integer(2))
g2
>>> M.coefficient_form(Integer(3))
g3
>>> M.coefficient_forms(T)
[g1, g2, g3]
>>> M.coefficient_forms(T**Integer(2))
[(T^3 + T)*g1,
 g1^4 + (T^9 + T)*g2,
 g1^9*g2 + g1*g2^3 + (T^27 + T)*g3,
 g1^27*g3 + g1*g3^3 + g2^10,
 g2^27*g3 + g2*g3^9,
 g3^28]
```

REFERENCE:

For a quick introduction to Drinfeld modular forms, see the *tutorial*. For more extensive references, see [Gek1988] and [BRP2018].

Element

alias of *DrinfeldModularFormsElement*

basis(k)

alias of *basis_of_weight()*.

basis_of_weight(k)

Return a list of Drinfeld modular forms which forms a basis for the subspace of weight k.

Note that if $k \not\equiv 0$ modulo $q-1$, then the subspace is 0.

An alias of this method is `basis`.

INPUT:

- `k` – integer

EXAMPLES:

```
sage: q = 3; A = GF(q)['T']; K = Frac(A);
sage: M = DrinfeldModularForms(K, 2)
sage: M.basis_of_weight(q - 1)
[g1]
sage: M.basis_of_weight(q^2 - 1)
[g2, g1^4]
sage: M.basis_of_weight(q^3 - 1)
[g1*g2^3, g1^5*g2^2, g1^9*g2, g1^13]
sage: M.basis_of_weight(19*(q-1))
[g1^3*g2^4, g1^7*g2^3, g1^11*g2^2, g1^15*g2, g1^19]
```

```
>>> from sage.all import *
>>> q = Integer(3); A = GF(q)['T']; K = Frac(A);
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.basis_of_weight(q - Integer(1))
[g1]
>>> M.basis_of_weight(q**Integer(2) - Integer(1))
[g2, g1^4]
>>> M.basis_of_weight(q**Integer(3) - Integer(1))
[g1*g2^3, g1^5*g2^2, g1^9*g2, g1^13]
>>> M.basis_of_weight(Integer(19)*(q-Integer(1)))
[g1^3*g2^4, g1^7*g2^3, g1^11*g2^2, g1^15*g2, g1^19]
```

coefficient_form(i, *a=None*)

Return the i-th coefficient form of the universal Drinfeld module over $\Omega^r(\mathbb{C}_\infty)$:

$$\phi_{w,a} = a + g_{1,a}\tau + \cdots + g_{rd_a,a}\tau^{rd_a}$$

where $d_a := \deg(a)$.

INPUT:

- `i` – integer between 1 and rd_a
- `a` – (default: `None`) an element in the ring of regular functions. If a is `None`, then the method returns the i-th coefficient form of $\phi_{w,T}$.

EXAMPLES:

```
sage: q = 3
sage: A = GF(q)['T']
sage: K.<T> = Frac(A)
sage: M = DrinfeldModularForms(K, 3)
sage: M.coefficient_form(1)
g1
sage: M.coefficient_form(2)
g2
sage: M.coefficient_form(3)
g3
sage: M.coefficient_form(3, T^2)
g1^9*g2 + g1*g2^3 + (T^27 + T)*g3
```

```
>>> from sage.all import *
>>> q = Integer(3)
>>> A = GF(q)['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> M = DrinfeldModularForms(K, Integer(3))
>>> M.coefficient_form(Integer(1))
g1
>>> M.coefficient_form(Integer(2))
g2
>>> M.coefficient_form(Integer(3))
g3
>>> M.coefficient_form(Integer(3), T**Integer(2))
g1^9*g2 + g1*g2^3 + (T^27 + T)*g3
```

```
sage: M = DrinfeldModularForms(K, 2, has_type=True)
sage: M.coefficient_form(1)
g1
sage: M.coefficient_form(2)
h2^2
sage: M.coefficient_form(2, T^3 + T^2 + T)
(T^9 + T^3 + T + 1)*g1^4 + (T^18 + T^10 + T^9 + T^2 + T + 1)*h2^2
```

```
>>> from sage.all import *
>>> M = DrinfeldModularForms(K, Integer(2), has_type=True)
>>> M.coefficient_form(Integer(1))
g1
>>> M.coefficient_form(Integer(2))
h2^2
>>> M.coefficient_form(Integer(2), T**Integer(3) + T**Integer(2) + T)
(T^9 + T^3 + T + 1)*g1^4 + (T^18 + T^10 + T^9 + T^2 + T + 1)*h2^2
```

coefficient_forms(*a=None*)

Return the list of all coefficients of the universal Drinfeld module at a.

See also *coefficient_form()* for definitions.

INPUT:

- a – (default: None) an element in the ring of regular functions. If a is None, then the method returns the coefficients forms at $a = T$.

OUTPUT: list of Drinfeld modular forms. The i-th element of that list corresponds to the $(i+1)$-th coefficient form at a.

EXAMPLES:

```
sage: q = 3
sage: A = GF(q)['T']
sage: K.<T> = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.coefficient_forms()
[g1, g2]
sage: M.coefficient_forms(T^2)
[(T^3 + T)*g1, g1^4 + (T^9 + T)*g2, g1^9*g2 + g1*g2^3, g2^10]
sage: M.coefficient_forms(T^3)
[(T^6 + T^4 + T^2)*g1,
 (T^9 + T^3 + T)*g1^4 + (T^18 + T^10 + T^2)*g2,
 g1^13 + (T^27 + T^9 + T)*g1^9*g2 + (T^27 + T^3 + T)*g1*g2^3,
 g1^36*g2 + g1^28*g2^3 + g1^4*g2^9 + (T^81 + T^9 + T)*g2^10,
 g1^81*g2^10 + g1^9*g2^28 + g1*g2^30,
 g2^91]
```

```
>>> from sage.all import *
>>> q = Integer(3)
>>> A = GF(q)['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.coefficient_forms()
[g1, g2]
>>> M.coefficient_forms(T**Integer(2))
[(T^3 + T)*g1, g1^4 + (T^9 + T)*g2, g1^9*g2 + g1*g2^3, g2^10]
>>> M.coefficient_forms(T**Integer(3))
[(T^6 + T^4 + T^2)*g1,
 (T^9 + T^3 + T)*g1^4 + (T^18 + T^10 + T^2)*g2,
 g1^13 + (T^27 + T^9 + T)*g1^9*g2 + (T^27 + T^3 + T)*g1*g2^3,
 g1^36*g2 + g1^28*g2^3 + g1^4*g2^9 + (T^81 + T^9 + T)*g2^10,
 g1^81*g2^10 + g1^9*g2^28 + g1*g2^30,
 g2^91]
```

gen(n)

Return the n-th generator of this ring.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A); T = K.gen()
sage: M = DrinfeldModularForms(K, 2)
sage: M.gen(0)
g1
sage: M.1  # equivalent to M.gen(1)
g2
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A); T = K.gen()
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.gen(Integer(0))
```

(continues on next page)

(continued from previous page)

```
g1
>>> M.gen(1)  # equivalent to M.gen(1)
g2
```

Note

Recall that the ring of Drinfeld modular forms is generated by the r coefficient forms of the universal Drinfeld module at T, $g_1, g_2, \ldots, g_r$, see *coefficient_forms()*. We highlight however that we make a shift in the indexing so that the i-th generator corresponds to the $i+1$-th coefficient form for $0 \leq i \leq r-1$.

gens()

Return a tuple of generators of this ring.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A); T = K.gen()
sage: M = DrinfeldModularForms(K, 5)
sage: M.gens()
(g1, g2, g3, g4, g5)
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A); T = K.gen()
>>> M = DrinfeldModularForms(K, Integer(5))
>>> M.gens()
(g1, g2, g3, g4, g5)
```

ngens()

Return the number of generators of this ring.

Note that the number of generators is equal to the rank.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A); T = K.gen()
sage: M = DrinfeldModularForms(K, 5)
sage: M.ngens()
5
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A); T = K.gen()
>>> M = DrinfeldModularForms(K, Integer(5))
>>> M.ngens()
5
```

one()

Return the multiplicative unit.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A); T = K.gen()
sage: M = DrinfeldModularForms(K, 2)
```

(continues on next page)

(continued from previous page)

```
sage: M.one()
1
sage: M.one() * M.0
g1
sage: M.one().is_one()
True
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A); T = K.gen()
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.one()
1
>>> M.one() * M.gen(0)
g1
>>> M.one().is_one()
True
```

polynomial_ring()

Return the multivariate polynomial ring over the base ring where each variable corresponds to a generator of this Drinfeld modular forms ring.

EXAMPLES:

```
sage: q = 3; A = GF(q)['T']; K = Frac(A);
sage: M = DrinfeldModularForms(K, 2)
sage: P = M.polynomial_ring()
sage: P
Multivariate Polynomial Ring in g1, g2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> q = Integer(3); A = GF(q)['T']; K = Frac(A);
>>> M = DrinfeldModularForms(K, Integer(2))
>>> P = M.polynomial_ring()
>>> P
Multivariate Polynomial Ring in g1, g2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

The degree of the variables corresponds to the weight of the associated generator:

```
sage: P.inject_variables()
Defining g1, g2
sage: g1.degree()
2
sage: g2.degree()
8
```

```
>>> from sage.all import *
>>> P.inject_variables()
Defining g1, g2
>>> g1.degree()
2
```

(continues on next page)

(continued from previous page)

```
>>> g2.degree()
8
```

rank()

Return the rank of this ring of Drinfeld modular forms.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A);
sage: DrinfeldModularForms(K, 2).rank()
2
sage: DrinfeldModularForms(K, 3).rank()
3
sage: DrinfeldModularForms(K, 4).rank()
4
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A);
>>> DrinfeldModularForms(K, Integer(2)).rank()
2
>>> DrinfeldModularForms(K, Integer(3)).rank()
3
>>> DrinfeldModularForms(K, Integer(4)).rank()
4
```

zero()

Return the additive identity.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A); T = K.gen()
sage: M = DrinfeldModularForms(K, 2)
sage: M.zero()
0
sage: M.zero() + M.1
g2
sage: M.zero() * M.1
0
sage: M.zero().is_zero()
True
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A); T = K.gen()
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.zero()
0
>>> M.zero() + M.gen(1)
g2
>>> M.zero() * M.gen(1)
0
>>> M.zero().is_zero()
True
```

2.3 Elements of Drinfeld modular forms rings

This module defines the elements of the class *DrinfeldModularForms*.

AUTHORS:

- David Ayotte (2022): initial version

class sage.modular.drinfeld_modform.element.**DrinfeldModularFormsElement**(*parent*, *polynomial*)

Bases: ModuleElement

Element class of rings of Drinfeld modular forms.

Recall that a *graded Drinfeld form* is a sum of Drinfeld modular forms having potentially different weights:

$$F = f_{k_1} + f_{k_2} + \cdots + f_{k_n}$$

where f_{k_i} is a Drinfeld modular form of weight k_i. We also say that f_{k_i} is an *homogeneous component of weight* k_i. If $n = 1$, then we say that F is *homogeneous of weight* k_1.

EXAMPLES: use the `inject_variable` method of the parent to quickly assign variables names to the generators:

```
sage: A = GF(3)['T']
sage: K.<T> = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.inject_variables()
Defining g1, g2
sage: g1 in M
True
sage: g2.parent()
Ring of Drinfeld modular forms of rank 2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']
>>> K = Frac(A, names=('T',)); (T,) = K._first_ngens(1)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.inject_variables()
Defining g1, g2
>>> g1 in M
True
>>> g2.parent()
Ring of Drinfeld modular forms of rank 2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

Next, via algebraic combination of the generator, we may create any element of the ring:

```
sage: F = g1*g2 + g2
sage: F
g1*g2 + g2
sage: F.is_homogeneous()
False
sage: F.homogeneous_components()
{8: g2, 10: g1*g2}
```

```
>>> from sage.all import *
>>> F = g1*g2 + g2
>>> F
g1*g2 + g2
>>> F.is_homogeneous()
False
>>> F.homogeneous_components()
{8: g2, 10: g1*g2}
```

If the created form is homogeneous, we can ask for its weight in which case it will be a Drinfeld modular form:

```
sage: H = g1^4*g2^9 + T*g1^8*g2^8 + (T^2 - 1)*g1^28*g2^3
sage: H.is_homogeneous()
True
sage: H.weight()
80
```

```
>>> from sage.all import *
>>> H = g1**Integer(4)*g2**Integer(9) + T*g1**Integer(8)*g2**Integer(8) +␣
↪(T**Integer(2) - Integer(1))*g1**Integer(28)*g2**Integer(3)
>>> H.is_homogeneous()
True
>>> H.weight()
80
```

You can also construct an element by simply passing a multivariate polynomial to the parent:

```
sage: f1, f2 = polygens(K, 2, 'f1, f2')
sage: M(f1)
g1
sage: M(f2)
g2
sage: M(T*f1 + f2^3 + T^2 + 1)
g2^3 + T*g1 + (T^2 + 1)
```

```
>>> from sage.all import *
>>> f1, f2 = polygens(K, Integer(2), 'f1, f2')
>>> M(f1)
g1
>>> M(f2)
g2
>>> M(T*f1 + f2**Integer(3) + T**Integer(2) + Integer(1))
g2^3 + T*g1 + (T^2 + 1)
```

Note

This class should not be directly instantiated, instead create an instance of the parent *DrinfeldModularForms* and access its elements using the relevant methods.

homogeneous_components()

Return the homogeneous components of this graded Drinfeld form.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.inject_variables()
Defining g1, g2
sage: F = g1 + g1^2 + g1*g2^2 + g2^4
sage: D = F.homogeneous_components(); D
{2: g1, 4: g1^2, 18: g1*g2^2, 32: g2^4}
sage: D[32]
g2^4
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.inject_variables()
Defining g1, g2
>>> F = g1 + g1**Integer(2) + g1*g2**Integer(2) + g2**Integer(4)
>>> D = F.homogeneous_components(); D
{2: g1, 4: g1^2, 18: g1*g2^2, 32: g2^4}
>>> D[Integer(32)]
g2^4
```

is_homogeneous()

Return whether the graded form is homogeneous in the weight.

We recall that elements of Drinfeld modular forms ring are not necessarily modular forms as they may have mixed weight components.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.inject_variables()
Defining g1, g2
sage: f = g1^5*g2^2  # homogeneous polynomial
sage: f.is_homogeneous()
True
sage: g = g1 + g2  # mixed weight components
sage: g.is_homogeneous()
False
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.inject_variables()
Defining g1, g2
>>> f = g1**Integer(5)*g2**Integer(2)  # homogeneous polynomial
>>> f.is_homogeneous()
True
>>> g = g1 + g2  # mixed weight components
>>> g.is_homogeneous()
False
```

is_one()

Return `True` whether this graded Drinfeld form is the multiplicative identity.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: u = M.one()
sage: u.is_one()
True
sage: (M.0).is_one()
False
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> u = M.one()
>>> u.is_one()
True
>>> (M.gen(0)).is_one()
False
```

is_zero()

Return `True` whether this graded Drinfeld form is the additive identity.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: z = M.zero()
sage: z.is_zero()
True
sage: f = M.0
sage: f.is_zero()
False
sage: (f - f).is_zero()
True
sage: (0 * M.0).is_zero()
True
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> z = M.zero()
>>> z.is_zero()
True
>>> f = M.gen(0)
>>> f.is_zero()
False
>>> (f - f).is_zero()
True
>>> (Integer(0) * M.gen(0)).is_zero()
True
```

polynomial()

Return this graded Drinfeld forms as a multivariate polynomial over the generators of the ring.

OUTPUT: a multivariate polynomial over the base ring

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.inject_variables()
Defining g1, g2
sage: P1 = g1.polynomial();
sage: P2 = g2.polynomial();
sage: P2^2 + P1^2 + P1
g2^2 + g1^2 + g1
sage: P1.parent()
Multivariate Polynomial Ring in g1, g2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.inject_variables()
Defining g1, g2
>>> P1 = g1.polynomial();
>>> P2 = g2.polynomial();
>>> P2**Integer(2) + P1**Integer(2) + P1
g2^2 + g1^2 + g1
>>> P1.parent()
Multivariate Polynomial Ring in g1, g2 over Fraction Field of Univariate␣
↪Polynomial Ring in T over Finite Field of size 3
```

The degree of each variables corresponds to the weight of the generator:

```
sage: P1.degree()
2
sage: P2.degree()
8
```

```
>>> from sage.all import *
>>> P1.degree()
2
>>> P2.degree()
8
```

rank()

Return the rank of this graded Drinfeld form.

Note that the rank is independent of the chosen form and depends only on the parent.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M2 = DrinfeldModularForms(K, 2)
sage: (M2.0).rank()
```

(continues on next page)

(continued from previous page)

```
2
sage: M5 = DrinfeldModularForms(K, 5)
sage: (M5.0 + M5.3).rank()
5
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M2 = DrinfeldModularForms(K, Integer(2))
>>> (M2.gen(0)).rank()
2
>>> M5 = DrinfeldModularForms(K, Integer(5))
>>> (M5.gen(0) + M5.gen(3)).rank()
5
```

type()

Return the type of this graded Drinfeld form.

Recall that the *type* is the integer $0 \leq m \leq q-1$ such that

$$f(\gamma(w)) = \det(\gamma)^m j(\gamma, w)^k f(w).$$

EXAMPLES:

```
sage: A = GF(11)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2, has_type=True)
sage: M.inject_variables()
Defining g1, h2
sage: F = g1*h2^9
sage: F.type()
9
sage: (h2^11).type()
1
sage: g1.type()
0
```

```
>>> from sage.all import *
>>> A = GF(Integer(11))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2), has_type=True)
>>> M.inject_variables()
Defining g1, h2
>>> F = g1*h2**Integer(9)
>>> F.type()
9
>>> (h2**Integer(11)).type()
1
>>> g1.type()
0
```

The type only makes sense when the form is homogeneous:

```
sage: F = g1^4 + h2
sage: F.type()
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: the graded form is not homogeneous
```

```
>>> from sage.all import *
>>> F = g1**Integer(4) + h2
>>> F.type()
Traceback (most recent call last):
...
ValueError: the graded form is not homogeneous
```

weight()

Return the weight of this graded Drinfeld modular form.

EXAMPLES:

```
sage: A = GF(3)['T']; K = Frac(A)
sage: M = DrinfeldModularForms(K, 2)
sage: M.inject_variables()
Defining g1, g2
sage: g1.weight()
2
sage: g2.weight()
8
sage: f = g1^5*g2^2
sage: f.weight()
26
```

```
>>> from sage.all import *
>>> A = GF(Integer(3))['T']; K = Frac(A)
>>> M = DrinfeldModularForms(K, Integer(2))
>>> M.inject_variables()
Defining g1, g2
>>> g1.weight()
2
>>> g2.weight()
8
>>> f = g1**Integer(5)*g2**Integer(2)
>>> f.weight()
26
```

If the form is not homogeneous, then the method returns an error:

```
sage: f = g1 + g2
sage: f.weight()
Traceback (most recent call last):
...
ValueError: the graded form is not homogeneous
```

```
>>> from sage.all import *
>>> f = g1 + g2
>>> f.weight()
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: the graded form is not homogeneous
```

CHAPTER

THREE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

m

sage.modular.drinfeld_modform.congroup_gamma0, 1
sage.modular.drinfeld_modform.element, 20
sage.modular.drinfeld_modform.ring, 10
sage.modular.drinfeld_modform.tutorial, 7

INDEX

B

base_ring() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 2
basis() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 14
basis_of_weight() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 14

C

coefficient_form() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 14
coefficient_forms() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 15

D

DrinfeldModularForms (*class in sage.modular.drinfeld_modform.ring*), 10
DrinfeldModularFormsElement (*class in sage.modular.drinfeld_modform.element*), 20

E

Element (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld attribute*), 2
Element (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms attribute*), 14

G

Gamma0_drinfeld (*class in sage.modular.drinfeld_modform.congroup_gamma0*), 2
Gamma0Element (*class in sage.modular.drinfeld_modform.congroup_gamma0*), 1
gen() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 16
gens() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 17
genus() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 2

H

homogeneous_components() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 21

I

InclusionIntoMatrixSpace (*class in sage.modular.drinfeld_modform.congroup_gamma0*), 6
index() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 4
is_homogeneous() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 22
is_one() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0Element method*), 1
is_one() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 22
is_zero() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 23

L

level() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 4
level() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0Element method*), 1

M

matrix_space() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 4
module
    sage.modular.drinfeld_modform.congroup_gamma0, 1

sage.modular.drinfeld_modform.element, 20
sage.modular.drinfeld_modform.ring, 10
sage.modular.drinfeld_modform.tutorial, 7

N

ncusps() (*sage.modular.drinfeld_modform.congroup_gamma0.Gamma0_drinfeld method*), 5
ngens() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 17

O

one() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 17

P

polynomial() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 23
polynomial_ring() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 18

R

rank() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 24
rank() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 19

S

sage.modular.drinfeld_modform.congroup_gamma0
module, 1
sage.modular.drinfeld_modform.element
module, 20
sage.modular.drinfeld_modform.ring
module, 10
sage.modular.drinfeld_modform.tutorial
module, 7

T

type() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 25

W

weight() (*sage.modular.drinfeld_modform.element.DrinfeldModularFormsElement method*), 26

Z

zero() (*sage.modular.drinfeld_modform.ring.DrinfeldModularForms method*), 19