Tensors on free modules

A tutorial

This notebook provides some introduction to **tensors on free modules of finite rank** in SageMath. This is a pure algebraic subpart of the SageManifolds project, which does not depend on other parts of SageManifolds and which has been integrated in SageMath 6.6.

In [1]:
```
version()
```

Out[1]: 'SageMath version 10.3.beta6, Release Date: 2024-01-21'

First we set up the notebook to display mathematical objects using LaTeX rendering:

In [2]:
```
%display latex
```

Constructing a free module of finite rank

Let R be a commutative ring and M a *free module of finite rank over* R, i.e. a module over R that admits a *finite basis* (finite family of linearly independent generators). Since R is commutative, it has the invariant basis number property: all bases of M have the same cardinality, which is called the *rank of* M. In this tutorial, we consider a free module of rank 3 over the integer ring $\mathbb{Z}$:

In [3]:
```
M = FiniteRankFreeModule(ZZ, 3, name='M', start_index=1)
```

The first two arguments are the ring and the rank; the third one is a string to denote the module and the last one defines the range of indices to be used for tensor components on the module: setting it to 1 means that indices will range in $\{1, 2, 3\}$. The default value is `start_index=0`.

The function `print` returns a short description of the just constructed module:

In [4]:
```
print(M)
```

```
Rank-3 free module M over the Integer Ring
```

If we ask just for M, the module's LaTeX symbol is returned; by default, this is the same as the argument `name` in the constructor (this can be changed by providing the optional argument `latex_name`:

In [5]:
```
M
```

Out[5]: M

In [6]:
```
M1 = FiniteRankFreeModule(ZZ, 3, name='M', latex_name=r'\mathcal{M}', sta
```

```
M1
```

Out[6]: $\mathcal{M}$

The indices of basis elements or tensor components on the module are generated by the method `irange()`, to be used in loops:

In [7]:
```
for i in M.irange():
    print(i)
```

```
1
2
3
```

If the parameter `start_index` had not been specified, the default range of the indices would have been $\{0, 1, 2\}$ instead:

In [8]:
```
M0 = FiniteRankFreeModule(ZZ, 3, name='M')
for i in M0.irange():
    print(i)
```

```
0
1
2
```

M is the category of finite dimensional modules over $\mathbb{Z}$:

In [9]:
```
print(M.category())
```

```
Category of finite dimensional modules over Integer Ring
```

Self-inquiry commands are

In [10]:
```
M.base_ring()
```

Out[10]: $\mathbf{Z}$

In [11]:
```
M.rank()
```

Out[11]: 3

Defining bases on the free module

At construction, the free module M has no pre-defined basis:

In [12]:
```
M.print_bases()
```

```
No basis has been defined on the Rank-3 free module M over the Integer Rin
g
```

In [13]:
```
M.bases()
```

Out[13]: []

For this reason, the class `FiniteRankFreeModule` does not inherit from Sage class `CombinatorialFreeModule`

In [14]:
```
isinstance(M, CombinatorialFreeModule)
```

Out[14]: False

and M does not belong to the category of modules with a distinguished basis:

In [15]:
```
M in ModulesWithBasis(ZZ)
```

Out[15]: False

It simply belongs to the category of modules over $\mathbb{Z}$:

In [16]:
```
M in Modules(ZZ)
```

Out[16]: True

More precisely, it belongs to the subcategory of finite dimensional modules over $\mathbb{Z}$:

In [17]:
```
M in Modules(ZZ).FiniteDimensional()
```

Out[17]: True

We define a first basis on M as follows:

In [18]:
```
e = M.basis('e')
e
```

Out[18]: (e_1, e_2, e_3)

In [19]:
```
M.print_bases()
```

```
Bases defined on the Rank-3 free module M over the Integer Ring:
 - (e_1,e_2,e_3) (default basis)
```

The elements of the basis are accessed via their indices:

In [20]:
```
e[1]
```

Out[20]: e_1

In [21]:
```
print(e[1])
```

```
Element e_1 of the Rank-3 free module M over the Integer Ring
```

In [22]:
```
e[1] in M
```

Out[22]: True

In [23]:
```
e[1].parent()
```

Out[23]: M

Let us introduce a second basis on the free module M from a family of 3 linearly independent module elements:

In [24]:
```
f = M.basis('f', from_family=(-e[1]+2*e[2]-4*e[3],
```

```
                                               e[2]+2*e[3],
                                               e[2]+3*e[3]))
print(f)
f
```

```
Basis (f_1,f_2,f_3) on the Rank-3 free module M over the Integer Ring
```

Out[24]: (f_1, f_2, f_3)

We may ask to view each element of basis f in terms of its expansion onto basis e, via the method `display()` :

In [25]:
```
f[1].display(e)
```

Out[25]: $f_1 = -e_1 + 2e_2 - 4e_3$

In [26]:
```
f[2].display(e)
```

Out[26]: $f_2 = e_2 + 2e_3$

In [27]:
```
f[3].display(e)
```

Out[27]: $f_3 = e_2 + 3e_3$

Conversely, the expression of basis e is terms of basis f is

In [28]:
```
e[1].display(f)
```

Out[28]: $e_1 = -f_1 + 10f_2 - 8f_3$

In [29]:
```
e[2].display(f)
```

Out[29]: $e_2 = 3f_2 - 2f_3$

In [30]:
```
e[3].display(f)
```

Out[30]: $e_3 = -f_2 + f_3$

The module automorphism a relating the two bases is obtained as

In [31]:
```
a = M.change_of_basis(e,f)
a
```

Out[31]: Automorphism of the Rank-3 free module M over the Integer Ring

It belongs to the general linear group of the free module M:

In [32]:
```
a.parent()
```

Out[32]: $\mathrm{GL}(M)$

and its matrix w.r.t. basis e is

In [33]:
```
a.matrix(e)
```

Out[33]:
$$\begin{pmatrix} -1 & 0 & 0 \\ 2 & 1 & 1 \\ -4 & 2 & 3 \end{pmatrix}$$

Let us check that the elements of basis f are images of the elements of basis e via a:

In [34]:
```
all([f[i] == a(e[i]) for i in M.irange()])
```

Out[34]: True

The reverse change of basis is of course the inverse automorphism:

In [35]:
```
M.change_of_basis(f,e) == a^(-1)
```

Out[35]: True

In [36]:
```
(a^(-1)).matrix(f)
```

Out[36]:
$$\begin{pmatrix} -1 & 0 & 0 \\ 10 & 3 & -1 \\ -8 & -2 & 1 \end{pmatrix}$$

At this stage, two bases have been defined on M:

In [37]:
```
M.print_bases()
```

```
Bases defined on the Rank-3 free module M over the Integer Ring:
 - (e_1,e_2,e_3) (default basis)
 - (f_1,f_2,f_3)
```

The first defined basis, e, is considered as the *default basis*, which means that it can be skipped in any method argument requiring a basis. For instance, let us consider the method `display()` :

In [38]:
```
f[1].display(e)
```

Out[38]: $f_1 = -e_1 + 2e_2 - 4e_3$

Since e is the default basis, the above command is fully equivalent to

In [39]:
```
f[1].display()
```

Out[39]: $f_1 = -e_1 + 2e_2 - 4e_3$

Of course, the names of non-default bases have to be specified:

In [40]:
```
f[1].display(f)
```

Out[40]: $f_1 = f_1$

In [41]:
```
e[1].display(f)
```

Out[41]: $e_1 = -f_1 + 10f_2 - 8f_3$

Note that the concept of ***default basis*** is different from that of ***distinguished**

basis* which is implemented in other free module constructions in Sage (e.g. `CombinatorialFreeModule`): the default basis is intended only for shorthand notations in user commands, avoiding to repeat the basis name many times; it is by no means a privileged basis on the module. For user convenience, the default basis can be changed at any moment by means of the method `set_default_basis()`:

In [42]:
```
M.set_default_basis(f)
e[1].display()
```

Out[42]: $e_1 = -f_1 + 10f_2 - 8f_3$

Let us revert to e as the default basis:

In [43]:
```
M.set_default_basis(e)
```

Module elements

Elements of the free module M are constructed by providing their components with respect to a given basis to the operator `()` acting on the module:

In [44]:
```
v = M([3,-4,1], basis=e, name='v')
print(v)
```

```
Element v of the Rank-3 free module M over the Integer Ring
```

Since e is the default basis, its mention can be skipped:

In [45]:
```
v = M([3,-4,1], name='v')
print(v)
```

```
Element v of the Rank-3 free module M over the Integer Ring
```

In [46]:
```
v.display()
```

Out[46]: $v = 3e_1 - 4e_2 + e_3$

While v has been defined from the basis e, its expression in terms of the basis f can be evaluated, thanks to the known relation between the two bases:

In [47]:
```
v.display(f)
```

Out[47]: $v = -3f_1 + 17f_2 - 15f_3$

According to Sage terminology, the parent of v is of course M:

In [48]:
```
v.parent()
```

Out[48]: M

We have also

In [49]:
```
v in M
```

Out[49]: True

Let us define a second module element, from the basis f this time:

In [50]:
```
u = M([-1,3,5], basis=f, name='u')
u.display(f)
```

Out[50]: $u = -f_1 + 3f_2 + 5f_3$

Another way to define module elements is of course via linear combinations:

In [51]:
```
w = 2*e[1] - e[2] - 3*e[3]
print(w)
```

```
Element of the Rank-3 free module M over the Integer Ring
```

In [52]:
```
w.display()
```

Out[52]: $2e_1 - e_2 - 3e_3$

As the result of a linear combination, w has no name; it can be given one by the method `set_name()` and the LaTeX symbol can be specified if different from the name:

In [53]:
```
w.set_name('w', latex_name=r'\omega')
w.display()
```

Out[53]: $\omega = 2e_1 - e_2 - 3e_3$

Module operations are implemented, independently of the bases:

In [54]:
```
s = u + 3*v
print(s)
```

```
Element of the Rank-3 free module M over the Integer Ring
```

In [55]:
```
s.display()
```

Out[55]: $10e_1 - 6e_2 + 28e_3$

In [56]:
```
s.display(f)
```

Out[56]: $-10f_1 + 54f_2 - 40f_3$

In [57]:
```
s = u - v
print(s)
```

```
Element u-v of the Rank-3 free module M over the Integer Ring
```

In [58]:
```
s.display()
```

Out[58]: $u - v = -2e_1 + 10e_2 + 24e_3$

In [59]:
```
s.display(f)
```

Out[59]: $u - v = 2f_1 - 14f_2 + 20f_3$

The components of a module element with respect to a given basis are given by the method `components()`:

In [60]:
```
v.components(f)
```

Out[60]: `1-index components w.r.t. Basis (f_1,f_2,f_3) on the Rank-3 free mo`

A shortcut is `comp()`:

In [61]:
```
v.comp(f) is v.components(f)
```

Out[61]: True

In [62]:
```
for i in M.irange():
    print(v.comp(f)[i])
```

```
-3
17
-15
```

In [63]:
```
v.comp(f)[:]
```

Out[63]: $[-3, 17, -15]$

The function `display_comp()` provides a list of components w.r.t. to a given basis:

In [64]:
```
v.display_comp(f)
```

Out[64]:
$$\begin{array}{lcl} v^1 & = & -3 \\ v^2 & = & 17 \\ v^3 & = & -15 \end{array}$$

As a shortcut, instead of calling the method `comp()`, the basis can be provided as the first argument of the square bracket operator:

In [65]:
```
v[f,2]
```

Out[65]: 17

In [66]:
```
v[f,:]
```

Out[66]: $[-3, 17, -15]$

For the default basis, the basis can be omitted:

In [67]:
```
v[:]
```

Out[67]: $[3, -4, 1]$

In [68]:
```
v[2]
```

Out[68]: -4

A specific module element is the zero one:

In [69]:
```
print(M.zero())
```

```
Element zero of the Rank-3 free module M over the Integer Ring
```

In [70]:
```
M.zero()[:]
```

Out[70]: $[0, 0, 0]$

In [71]:
```
M.zero()[f,:]
```

Out[71]: $[0, 0, 0]$

In [72]:
```
v + M.zero() == v
```

Out[72]: True

Linear forms

Let us introduce some linear form on the free module M:

In [73]:
```
a = M.linear_form('a')
print(a)
```

```
Linear form a on the Rank-3 free module M over the Integer Ring
```

a is specified by its components with respect to the basis dual to e:

In [74]:
```
a[:] = [2,-1,3]
a.display()
```

Out[74]: $a = 2e^1 - e^2 + 3e^3$

The notation e^i stands for the elements of the basis dual to e, i.e. the basis of the dual module M^* such that

$$e^i(e_j) = \delta^i{}_j$$

Indeed

In [75]:
```
ed = e.dual_basis()
ed
```

Out[75]: $\left(e^1, e^2, e^3\right)$

In [76]:
```
print(ed[1])
```

```
Linear form e^1 on the Rank-3 free module M over the Integer Ring
```

In [77]:
```
ed[1](e[1]), ed[1](e[2]), ed[1](e[3])
```

Out[77]: $(1, 0, 0)$

In [78]:
```
ed[2](e[1]), ed[2](e[2]), ed[2](e[3])
```

Out[78]: $(0, 1, 0)$

In [79]:
```
ed[3](e[1]), ed[3](e[2]), ed[3](e[3])
```

Out[79]: $(0, 0, 1)$

The linear form a can also be defined by its components with respect to the basis dual to f:

In [80]:
```
a[f,:] = [2,-1,3]
a.display(f)
```

Out[80]: $a = 2f^1 - f^2 + 3f^3$

For consistency, the previously defined components with respect to the basis dual to e are automatically deleted and new ones are computed from the change-of-basis formula:

In [81]:
```
a.display()
```

Out[81]: $a = -36e^1 - 9e^2 + 4e^3$

By definition, linear forms belong to the dual module:

In [82]:
```
a.parent()
```

Out[82]: M^*

In [83]:
```
print(a.parent())
```

```
Dual of the Rank-3 free module M over the Integer Ring
```

In [84]:
```
a.parent() is M.dual()
```

Out[84]: True

The dual module is itself a free module of the same rank as M:

In [85]:
```
isinstance(M.dual(), FiniteRankFreeModule)
```

Out[85]: False

In [86]:
```
M.dual().rank()
```

Out[86]: 3

Linear forms map module elements to ring elements:

In [87]:
```
a(v)
```

Out[87]: -68

In [88]:
```
a(u)
```

Out[88]: 10

in a linear way:

In [89]:
```
a(u+2*v) == a(u) + 2*a(v)
```

Out[89]: True

Alternating forms

Let us introduce a second linear form, b, on the free module M:

In [90]:
```
b = M.linear_form('b')
b[:] = [-4,2,5]
```

and take its exterior product with the linear form a:

In [91]:
```
c = a.wedge(b)
print(c)
c
```

```
Alternating form a∧b of degree 2 on the Rank-3 free module M over the Inte
ger Ring
```

Out[91]: $a \wedge b$

In [92]:
```
c.display()
```

Out[92]: $a \wedge b = -108e^1 \wedge e^2 - 164e^1 \wedge e^3 - 53e^2 \wedge e^3$

In [93]:
```
c.display(f)
```

Out[93]: $a \wedge b = 12f^1 \wedge f^2 + 70f^1 \wedge f^3 - 53f^2 \wedge f^3$

In [94]:
```
c(u,v)
```

Out[94]: 8894

c is antisymmetric:

In [95]:
```
c(v,u)
```

Out[95]: -8894

and is multilinear:

In [96]:
```
c(u+4*w,v) == c(u,v) + 4*c(w,v)
```

Out[96]: True

We may check the standard formula for the exterior product of two linear forms:

In [97]:
```
c(u,v) == a(u)*b(v) - b(u)*a(v)
```

Out[97]: True

In terms of tensor product (denoted here by *), it reads

In [98]:
```
c == a*b - b*a
```

Out[98]: True

The parent of the alternating form c is the second external power of the dual module M^*, which is denoted by $\Lambda^2(M^*)$:

In [99]:
```
c.parent()
```

Out[99]: $\Lambda^2(M^*)$

In [100…
```
print(c.parent())
```

```
2nd exterior power of the dual of the Rank-3 free module M over the Intege
r Ring
```

c is a tensor of type $(0, 2)$:

In [101…
```
c.tensor_type()
```

Out[101… $(0, 2)$

whose components with respect to any basis are antisymmetric:

In [102…
```
c[:]  # components with respect to the default basis (e)
```

Out[102…
$$\begin{pmatrix} 0 & -108 & -164 \\ 108 & 0 & -53 \\ 164 & 53 & 0 \end{pmatrix}$$

In [103…
```
c[f,:] # components with respect to basis f
```

Out[103…
$$\begin{pmatrix} 0 & 12 & 70 \\ -12 & 0 & -53 \\ -70 & 53 & 0 \end{pmatrix}$$

In [104…
```
c.comp(f)
```

Out[104… **Fully antisymmetric 2-indices components w.r.t. Basis (f_1,f_2,f_3)**

An alternating form can be constructed from scratch:

In [105…
```
c1 = M.alternating_form(2)  # 2 stands for the degree
```

Only the non-zero and non-redundant components are to be defined (the others are deduced by antisymmetry); for the components with respect to the default basis, we write:

In [106…
```
c1[1,2] = -108
c1[1,3] = -164
c1[2,3] = -53
```

Then

In [107…
```
c1[:]
```

Out[107…
$$\begin{pmatrix} 0 & -108 & -164 \\ 108 & 0 & -53 \\ 164 & 53 & 0 \end{pmatrix}$$

In [108…
```
c1 == c
```

Out[108… True

Internally, only non-redundant components are stored, in a dictionary whose keys are the indices:

In [109…
```
c.comp(e)._comp
```

Out[109… $\{(1,2) : -108, (1,3) : -164, (2,3) : -53\}$

In [110…
```
c.comp(f)._comp
```

Out[110… $\{(1,2) : 12, (1,3) : 70, (2,3) : -53\}$

The other components are deduced by antisymmetry.

The exterior product of a linear form with an alternating form of degree 2 leads to an alternating form of degree 3:

In [111…
```
d = M.linear_form('d')
d[:] = [-1,-2,4]
s = d.wedge(c)
print(s)
```

```
Alternating form d∧a∧b of degree 3 on the Rank-3 free module M over the In
teger Ring
```

In [112…
```
s.display()
```

Out[112… $d \wedge a \wedge b = -707e^1 \wedge e^2 \wedge e^3$

In [113…
```
s.display(f)
```

Out[113… $d \wedge a \wedge b = 707f^1 \wedge f^2 \wedge f^3$

In [114…
```
s(e[1], e[2], e[3])
```

Out[114… -707

In [115…
```
s(f[1], f[2], f[3])
```

Out[115… 707

s is antisymmetric:

In [116…
```
s(u,v,w), s(u,w,v), s(v,w,u), s(v,u,w), s(w,u,v), s(w,v,u)
```

Out[116… $(-144228, 144228, -144228, 144228, -144228, 144228)$

Tensors

k and l being non negative integers, a tensor of type (k,l) on the free module M is a multilinear map

$$t: \underbrace{M^* \times \cdots \times M^*}_{k \text{ times}} \times \underbrace{M \times \cdots \times M}_{l \text{ times}} \longrightarrow R$$

In the present case the ring R is $\mathbb{Z}$.

For free modules of finite rank, we have the canonical isomorphism $M^{**} \simeq M$, so that the set of all tensors of type (k,l) can be identified with the tensor product

$$T^{(k,l)}(M) = \underbrace{M \otimes \cdots \otimes M}_{k \text{ times}} \otimes \underbrace{M^* \otimes \cdots \otimes M^*}_{l \text{ times}}$$

In particular, tensors of type $(1,0)$ are identified with elements of M:

In [117…
```
M.tensor_module(1,0) is M
```
Out[117… True

In [118…
```
v.tensor_type()
```
Out[118… $(1,0)$

According to the above definition, linear forms are tensors of type (0,1):

In [119…
```
a in M.tensor_module(0,1)
```
Out[119… True

We have the identification of $T^{(0,1)}(M)$ with M^*:

In [120…
```
M.tensor_module(0,1) is M.dual()
```
Out[120… True

Arbitrary tensors are constructed via the module method `tensor()` , by providing the tensor type (k,l) and possibly the symbol to denote the tensor:

In [121…
```
t = M.tensor((1,1), name='t')
print(t)
```
```
Type-(1,1) tensor t on the Rank-3 free module M over the Integer Ring
```

Let us set some component of t in the basis e, for instance the component $t^1{}_2$:

In [122…
```
t[e,1,2] = -3
```

Since e is the default basis, a shortcut for the above is

In [123…
```
t[1,2] = -3
```

The unset components are zero:

In [124…
```
t[:]
```

Out[124…
$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Components can be set at any time:

In [125…
```
t[2,3] = 4
t[:]
```

Out[125…
$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

The components with respect to the basis f are evaluated by the change-of-basis formula $e \to f$:

In [126…
```
t[f,:]
```

Out[126…
$$\begin{pmatrix} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{pmatrix}$$

Another view of t, which reflects the fact that $T^{(1,1)}(M) = M \otimes M^*$, is

In [127…
```
t.display()
```

Out[127… $t = -3e_1 \otimes e^2 + 4e_2 \otimes e^3$

Recall that (e^i) is the basis of M^* that is dual to the basis (e_i) of M.

In term of the basis (f_i) and its dual basis (f^i), we have

In [128…
```
t.display(f)
```

Out[128… $t = 6f_1 \otimes f^1 + 3f_1 \otimes f^2 + 3f_1 \otimes f^3 - 108f_2 \otimes f^1 - 6f_2 \otimes f^2 + 6f_2 \otimes f^3 + 80f_3 \otimes j$

As a tensor of type (1,1), t maps pairs (linear form, module element) to ring elements:

In [129…
```
t(a,v)
```

Out[129… -468

In [130…
```
t(a,v).parent()
```

Out[130… $\mathbf{Z}$

Tensors of type (1,1) can be considered as endomorphisms, thanks to the isomorphism

$$
\begin{array}{ccccccc}
\mathrm{End}(M) & \longrightarrow & T^{(1,1)}(M) & & & & \\
\tilde{t} & \longmapsto & t: & & M^* \times M & \longrightarrow & R \\
 & & & & (a, v) & \longmapsto & a(\tilde{t}(v))
\end{array}
$$

In [131…
```
tt = End(M)(t)
print(tt)
```

```
Generic endomorphism of Rank-3 free module M over the Integer Ring
```

In [132…
```
tt.parent()
```

Out[132… $\mathrm{Hom}\left(M, M\right)$

In a given basis, the matrix $\tilde{t}^i{}_j$ of the endomorphism $\tilde{t}$ is identical to the matrix of the tensor t:

In [133…
```
tt.matrix(e)
```

Out[133… $\left(\begin{array}{rrr} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{array}\right)$

In [134…
```
t[e,:]
```

Out[134… $\left(\begin{array}{rrr} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{array}\right)$

In [135…
```
tt.matrix(e) == t[e,:]
```

Out[135… True

In [136…
```
tt.matrix(f)
```

Out[136… $\left(\begin{array}{rrr} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{array}\right)$

In [137…
```
t[f,:]
```

Out[137… $\left(\begin{array}{rrr} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{array}\right)$

In [138…
```
tt.matrix(f) == t[f,:]
```

Out[138… True

As an endomorphism, t maps module elements to module elements:

In [139…
```
tt(v)
```

Out[139… $t\left(v\right)$

In [140…
```
tt(v).parent()
```
Out[140… M

In [141…
```
tt(v).display()
```
Out[141… $t(v) = 12e_1 + 4e_2$

t belongs to the module $T^{(1,1)}(M)$:

In [142…
```
t in M.tensor_module(1,1)
```
Out[142… True

or, in Sage terminology,

In [143…
```
t.parent() is M.tensor_module(1,1)
```
Out[143… True

In [144…
```
M.tensor_module(1,1).base_ring()
```
Out[144… $\mathbb{Z}$

In [145…
```
M.tensor_module(1,1).rank()
```
Out[145… 9

Tensor calculus

In addition to the arithmetic operations inherent to the module structure of $T^{(k,l)}(M)$, the following operations are implemented:

- tensor product
- symmetrization and antisymmetrization
- tensor contraction

Tensor product

The tensor product is formed with the * operator. For instance the tensor product $t \otimes a$ is

In [146…
```
ta = t*a
print(ta)
```
```
Type-(1,2) tensor t⊗a on the Rank-3 free module M over the Integer Ring
```

In [147…
```
ta
```
Out[147… $t \otimes a$

In [148…
```
ta.display()
```

Out[148... $t \otimes a = 108e_1 \otimes e^2 \otimes e^1 + 27e_1 \otimes e^2 \otimes e^2 - 12e_1 \otimes e^2 \otimes e^3 - 144e_2 \otimes e^3 \otimes e^1 - 36e_2$

In [149...
```
ta.display(f)
```

Out[149... $t \otimes a = 12f_1 \otimes f^1 \otimes f^1 - 6f_1 \otimes f^1 \otimes f^2 + 18f_1 \otimes f^1 \otimes f^3 + 6f_1 \otimes f^2 \otimes f^1 - 3f_1 \otimes$
$\otimes f^1 \otimes f^2 - 324f_2 \otimes f^1 \otimes f^3 - 12f_2 \otimes f^2 \otimes f^1 + 6f_2 \otimes f^2 \otimes f^2 - 18f_2 \otimes f^2 \otimes f^3 -$
$\otimes f^3 + 16f_3 \otimes f^2 \otimes f^1 - 8f_3 \otimes f^2 \otimes f^2 + 24f_3 \otimes f^2 \otimes f^3$

The components w.r.t. a given basis can also be displayed as an array:

In [150...
```
ta[:] # components w.r.t. the default basis (e)
```

Out[150... $[[[0,0,0],[108,27,-12],[0,0,0]],[[0,0,0],[0,0,0],[-144,-36,16]],[[0,0,0],[0,0,($

In [151...
```
ta[f,:] # components w.r.t. basis f
```

Out[151... $[[[12,-6,18],[6,-3,9],[6,-3,9]],[[-216,108,-324],[-12,6,-18],[12,-6,18]],[$

Each component ca be accessed individually:

In [152...
```
ta[1,2,3] # access to a component w.r.t. the default basis
```

Out[152... -12

In [153...
```
ta[f,1,2,3]
```

Out[153... 9

In [154...
```
ta.parent()
```

Out[154... $T^{(1,2)}(M)$

In [155...
```
ta in M.tensor_module(1,2)
```

Out[155... True

The tensor product is not commutative:

In [156...
```
print(a*t)
```

```
Type-(1,2) tensor a⊗t on the Rank-3 free module M over the Integer Ring
```

In [157...
```
a*t == t*a
```

Out[157... False

Forming a tensor of rank 4:

In [158...
```
tav = ta*v
print(tav)
```

```
Type-(2,2) tensor t⊗a⊗v on the Rank-3 free module M over the Integer Ring
```

In [159...
```
tav.display()
```

Out[159…
$t \otimes a \otimes v = 324 e_1 \otimes e_1 \otimes e^2 \otimes e^1 + 81 e_1 \otimes e_1 \otimes e^2 \otimes e^2 - 36 e_1 \otimes e_1 \otimes e^2 \otimes e^3 - 432 \otimes e^2 - 12 e_1 \otimes e_3 \otimes e^2 \otimes e^3 - 432 e_2 \otimes e_1 \otimes e^3 \otimes e^1 - 108 e_2 \otimes e_1 \otimes e^3 \otimes e^2 + 48 e_2 \otimes - 36 e_2 \otimes e_3 \otimes e^3 \otimes e^2 + 16 e_2 \otimes e_3 \otimes e^3 \otimes e^3$

Symmetrization / antisymmetrization

The (anti)symmetrization of a tensor t over n arguments involve the division by $n!$, which does not always make sense in the base ring R. In the present case, $R = \mathbb{Z}$ and to (anti)symmetrize over 2 arguments, we restrict to tensors with even components:

In [160…
```
g = M.tensor((0,2), name='g')
g[1,2], g[2,1], g[2,2], g[3,2], g[3,3] = 2, -4, 8, 2, -6
g[:]
```

Out[160…
$$\begin{pmatrix} 0 & 2 & 0 \\ -4 & 8 & 0 \\ 0 & 2 & -6 \end{pmatrix}$$

In [161…
```
s = g.symmetrize() ; s
```

Out[161… Symmetric bilinear form on the Rank-3 free module M over the Integer Ring

In [162…
```
s.symmetries()
```

```
symmetry: (0, 1); no antisymmetry
```

In [163…
```
s[:]
```

Out[163…
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 8 & 1 \\ 0 & 1 & -6 \end{pmatrix}$$

Symmetrization can be performed on an arbitray number of arguments, by providing their positions (first position = 0). In the present case

In [164…
```
s == g.symmetrize(0,1)
```

Out[164… True

One may use index notation to specify the symmetry:

In [165…
```
s == g['_(ab)']
```

Out[165… True

In [166…
```
s == g['_{(ab)}'] # LaTeX type notation
```

Out[166… True

Of course, since s is already symmetric:

In [167…
```
s.symmetrize() == s
```

Out[167… True

The antisymmetrization proceeds accordingly:

In [168…
```
s = g.antisymmetrize() ; s
```

Out[168… Alternating form of degree 2 on the Rank-3 free module M over the Integer Ring

In [169…
```
s.symmetries()
```

```
no symmetry; antisymmetry: (0, 1)
```

In [170…
```
s[:]
```

Out[170…
$$\begin{pmatrix} 0 & 3 & 0 \\ -3 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

In [171…
```
s == g.antisymmetrize(0,1)
```

Out[171… True

As for symmetries, index notation can be used, instead of `antisymmetrize()` :

In [172…
```
s == g['_[ab]']
```

Out[172… True

In [173…
```
s == g['_{[ab]}'] # LaTeX type notation
```

Out[173… True

Of course, since s is already antisymmetric:

In [174…
```
s == s.antisymmetrize()
```

Out[174… True

Tensor contractions

Contracting the type-(1,1) tensor t with the module element v results in another module element:

In [175…
```
t.contract(v)
```

Out[175… Element of the Rank-3 free module M over the Integer Ring

The components (w.r.t. a given basis) of the contraction are of course $t^i_j v^j$:

In [176…
```
t.contract(v)[i] == sum(t[i,j]*v[j] for j in M.irange())
```

Out[176… True

This contraction coincides with the action of t as an endomorphism:

In [177…
```
t.contract(v) == tt(v)
```

Out[177… True

Instead of `contract()` , index notations can be used to denote the contraction:

In [178…
```
t['^i_j']*v['j'] == t.contract(v)
```

Out[178… True

Contracting the linear form a with the module element v results in a ring element:

In [179…
```
a.contract(v)
```

Out[179… -68

It is of course the result of the linear form acting on the module element:

In [180…
```
a.contract(v) == a(v)
```

Out[180… True

By default, the contraction is performed on the last index of the first tensor and the first index of the second one. To perform contraction on other indices, one should specify the indices positions (with the convention position=0 for the first index): for instance to get the contraction $z^i{}_j = T^i{}_{kj} v^k$ (with $T = t \otimes a$):

In [181…
```
z = ta.contract(1,v)  # 1 -> second index of ta
print(z)
```

```
Type-(1,1) tensor on the Rank-3 free module M over the Integer Ring
```

To get $z^i{}_{jk} = t^l{}_j T^i{}_{lk}$:

In [182…
```
z = t.contract(0, ta, 1)  # 0 -> first index of t, 1 -> second index of t
print(z)
```

```
Type-(1,2) tensor on the Rank-3 free module M over the Integer Ring
```

or, in terms of index notation:

In [183…
```
z1 = t['^l_j']*ta['^i_lk']
z1 == z
```

Out[183… True

As for any function, inline documentation is obtained via the question mark:

In [184…
```
t.contract?
```

```
Signature:      t.contract(*args)
Docstring:
   Contraction on one or more indices with another tensor.

   INPUT:

   * "pos1" -- positions of the indices in "self" involved in the
     contraction; "pos1" must be a sequence of integers, with 0
     standing for the first index position, 1 for the second one, etc;
     if "pos1" is not provided, a single contraction on the last index
     position of "self" is assumed

   * "other" -- the tensor to contract with

   * "pos2" -- positions of the indices in "other" involved in the
     contraction, with the same conventions as for "pos1"; if "pos2"
     is not provided, a single contraction on the first index position
     of "other" is assumed

   OUTPUT:

   * tensor resulting from the contraction at the positions "pos1" and
     "pos2" of "self" with "other"

   EXAMPLES:

   Contraction of a tensor of type (0,1) with a tensor of type (1,0):

      sage: M = FiniteRankFreeModule(ZZ, 3, name='M')
      sage: e = M.basis('e')
      sage: a = M.linear_form()  # tensor of type (0,1) is a linear form
      sage: a[:] = [-3,2,1]
      sage: b = M([2,5,-2])  # tensor of type (1,0) is a module element
      sage: s = a.contract(b) ; s
      2
      sage: s in M.base_ring()
      True
      sage: s == a[0]*b[0] + a[1]*b[1] + a[2]*b[2]  # check of the computa
tion
      True

   The positions of the contraction indices can be set explicitly:

      sage: s == a.contract(0, b, 0)
      True
      sage: s == a.contract(0, b)
      True
      sage: s == a.contract(b, 0)
      True

   Instead of the explicit call to the method "contract()", the index
   notation can be used to specify the contraction, via Einstein
   convention (summation on repeated indices); it suffices to pass the
   indices as a string inside square brackets:

      sage: s1 = a['_i']*b['^i'] ; s1
      2
      sage: s1 == s
      True
```

In the present case, performing the contraction is identical to applying the linear form to the module element:

```
sage: a.contract(b) == a(b)
True
```

or to applying the module element, considered as a tensor of type (1,0), to the linear form:

```
sage: a.contract(b) == b(a)
True
```

We have also:

```
sage: a.contract(b) == b.contract(a)
True
```

Contraction of a tensor of type (1,1) with a tensor of type (1,0):

```
sage: a = M.tensor((1,1))
sage: a[:] = [[-1,2,3],[4,-5,6],[7,8,9]]
sage: s = a.contract(b) ; s
Element of the Rank-3 free module M over the Integer Ring
sage: s.display()
2 e_0 - 29 e_1 + 36 e_2
```

Since the index positions have not been specified, the contraction takes place on the last position of a (i.e. no. 1) and the first position of "b" (i.e. no. 0):

```
sage: a.contract(b) == a.contract(1, b, 0)
True
sage: a.contract(b) == b.contract(0, a, 1)
True
sage: a.contract(b) == b.contract(a, 1)
True
```

Using the index notation with Einstein convention:

```
sage: a['^i_j']*b['^j'] == a.contract(b)
True
```

The index "i" can be replaced by a dot:

```
sage: a['^._j']*b['^j'] == a.contract(b)
True
```

and the symbol "^" may be omitted, the distinction between contravariant and covariant indices being the position with respect to the symbol "_":

```
sage: a['._j']*b['j'] == a.contract(b)
True
```

Contraction is possible only between a contravariant index and a covariant one:

```
sage: a.contract(0, b)
Traceback (most recent call last):
...
```

```
TypeError: contraction on two contravariant indices not permitted
```

Contraction of a tensor of type (2,1) with a tensor of type (0,2):

```
sage: a = a*b ; a
Type-(2,1) tensor on the Rank-3 free module M over the Integer Ring
sage: b = M.tensor((0,2))
sage: b[:] = [[-2,3,1], [0,-2,3], [4,-7,6]]
sage: s = a.contract(1, b, 1) ; s
Type-(1,2) tensor on the Rank-3 free module M over the Integer Ring
sage: s[:]
[[[-9, 16, 39], [18, -32, -78], [27, -48, -117]],
 [[36, -64, -156], [-45, 80, 195], [54, -96, -234]],
 [[63, -112, -273], [72, -128, -312], [81, -144, -351]]]
```

Check of the computation:

```
sage: all(s[i,j,k] == a[i,0,j]*b[k,0]+a[i,1,j]*b[k,1]+a[i,2,j]*b[k,
2]
....:     for i in range(3) for j in range(3) for k in range(3))
True
```

Using index notation:

```
sage: a['il_j']*b['_kl'] == a.contract(1, b, 1)
True
```

LaTeX notation are allowed:

```
sage: a['^{il}_j']*b['_{kl}'] == a.contract(1, b, 1)
True
```

Indices not involved in the contraction may be replaced by dots:

```
sage: a['.l_.']*b['_.l'] == a.contract(1, b, 1)
True
```

The two tensors do not have to be defined on the same basis for the contraction to take place, reflecting the fact that the contraction is basis-independent:

```
sage: A = M.automorphism()
sage: A[:] =  [[0,0,1], [1,0,0], [0,-1,0]]
sage: h = e.new_basis(A, 'h')
sage: b.comp(h)[:]  # forces the computation of b's components w.r.t
. basis h
[-2 -3  0]
[ 7  6 -4]
[ 3 -1 -2]
sage: b.del_other_comp(h)  # deletes components w.r.t. basis e
sage: list(b._components)  # indeed:
[Basis (h_0,h_1,h_2) on the Rank-3 free module M over the Integer Ri
ng]
sage: list(a._components)  # while a is known only in basis e:
[Basis (e_0,e_1,e_2) on the Rank-3 free module M over the Integer Ri
ng]
sage: s1 = a.contract(1, b, 1) ; s1  # yet the computation is possib
le
Type-(1,2) tensor on the Rank-3 free module M over the Integer Ring
sage: s1 == s  # ... and yields the same result as previously:
```

```
        True

    The contraction can be performed on more than a single index; for
    instance a 2-indices contraction of a type-(2,1) tensor with a
    type-(1,2) one is:

        sage: a  # a is a tensor of type-(2,1)
        Type-(2,1) tensor on the Rank-3 free module M over the Integer Ring
        sage: b = M([1,-1,2])*b ; b # a tensor of type (1,2)
        Type-(1,2) tensor on the Rank-3 free module M over the Integer Ring
        sage: s = a.contract(1,2,b,1,0) ; s # the double contraction
        Type-(1,1) tensor on the Rank-3 free module M over the Integer Ring
        sage: s[:]
        [ -36   30   15]
        [-252  210  105]
        [-204  170   85]
        sage: s == a['^.k_l']*b['^l_k.']  # the same thing in index notation
        True
Init docstring: Initialize self.  See help(type(self)) for accurate signat
ure.
File:           ~/sage/10.3/src/sage/tensor/modules/free_module_tensor.py
Type:           method
```