Tensors on free modules

A tutorial

This worksheet provides some introduction to **tensors on free modules of finite rank**. This is a pure algebraic subpart of SageManifolds (version 1.0), which does not depend on other parts of SageManifolds and which has been integrated in SageMath 6.6.

Click here to download the worksheet file (ipynb format). To run it, you must start SageMath with the Jupyter notebook, via the command `sage -n jupyter`

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

In [1]:
```
%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 [2]:
```
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 [3]:
```
print(M)
```
```
Rank-3 free module M over the Integer Ring
```

If we ask just for M, the module's LaTeX symbol is returned (provided that the worksheet's Typeset box has been selected); 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 [4]:
```
M
```
Out[4]: M

In [5]:
```
M1 = FiniteRankFreeModule(ZZ, 3, name='M', latex_name=r'\mathcal{M}', s
tart_index=1)
M1
```
Out[5]: $\mathcal{M}$

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

In [6]:
```
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 [7]:
```
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 [8]:
```
print(M.category())
```

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

Self-inquiry commands are

In [9]:
```
M.base_ring()
```

Out[9]: $\mathbf{Z}$

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

Out[10]: 3

Defining bases on the free module

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

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

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

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

Out[12]: []

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

In [13]:
```
isinstance(M, CombinatorialFreeModule)
```

Out[13]: False

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

In [14]:
```
M in ModulesWithBasis(ZZ)
```
Out[14]: False

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

In [15]:
```
M in Modules(ZZ)
```
Out[15]: True

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

In [16]:
```
M in Modules(ZZ).FiniteDimensional()
```
Out[16]: True

We define a first basis on M as follows:

In [17]:
```
e = M.basis('e') ; e
```
Out[17]: (e_1, e_2, e_3)

In [18]:
```
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 [19]:
```
e[1]
```
Out[19]: e_1

In [20]:
```
print(e[1])
```
```
Element e_1 of the Rank-3 free module M over the Integer Ring
```

In [21]:
```
e[1] in M
```
Out[21]: True

In [22]:
```
e[1].parent()
```
Out[22]: M

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

In [23]:
```
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[23]: (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()`, abridged as `disp()`:

In [24]:
```
f[1].disp(e)
```
Out[24]: $f_1 = -e_1 + 2e_2 - 4e_3$

In [25]:
```
f[2].disp(e)
```
Out[25]: $f_2 = e_2 + 2e_3$

In [26]:
```
f[3].disp(e)
```
Out[26]: $f_3 = e_2 + 3e_3$

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

In [27]:
```
e[1].disp(f)
```
Out[27]: $e_1 = -f_1 + 10f_2 - 8f_3$

In [28]:
```
e[2].disp(f)
```
Out[28]: $e_2 = 3f_2 - 2f_3$

In [29]:
```
e[3].disp(f)
```
Out[29]: $e_3 = -f_2 + f_3$

The module automorphism a relating the two bases is obtained as

In [30]:
```
a = M.change_of_basis(e,f) ; a
```
Out[30]: 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 [31]:
```
a.parent()
```
Out[31]: $\mathrm{GL}(M)$

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

In [32]:
```
a.matrix(e)
```
Out[32]:
$$\begin{pmatrix} -1 & 0 & 0 \\ 2 & 1 & 1 \\ -4 & 2 & 3 \end{pmatrix}$$

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

In [33]:
```
f[1] == a(e[1]), f[2] == a(e[2]), f[3] == a(e[3])
```
Out[33]: (True, True, True)

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

In [34]:
```
M.change_of_basis(f,e) == a^(-1)
```
Out[34]: True

In [35]:
```
(a^(-1)).matrix(f)
```

Out[35]:
$$\begin{pmatrix} -1 & 0 & 0 \\ 10 & 3 & -1 \\ -8 & -2 & 1 \end{pmatrix}$$

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

In [36]:
```
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 [37]:
```
f[1].display(e)
```

Out[37]: $f_1 = -e_1 + 2e_2 - 4e_3$

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

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

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

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

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

Out[39]: $f_1 = f_1$

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

Out[40]: $e_1 = -f_1 + 10f_2 - 8f_3$

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

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

Let us revert to e as the default basis:

In [42]:
```
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 [43]:
```
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 [44]:
```
v = M([3,-4,1], name='v')
print(v)
```

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

In [45]:
```
v.disp()
```

Out[45]: $v = 3e_1 - 4e_2 + e_3$

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 [46]:
```
v.disp(f)
```

Out[46]: $v = -3f_1 + 17f_2 - 15f_3$

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

In [47]:
```
v.parent()
```

Out[47]: M

We have also

In [48]:
```
v in M
```

Out[48]: True

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

In [49]:
```
u = M([-1,3,5], basis=f, name='u')
u.disp(f)
```

Out[49]: $u = -f_1 + 3f_2 + 5f_3$

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

In [50]:
```
w = 2*e[1] - e[2] - 3*e[3]
print(w)
```

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

In [51]:
```
w.disp()
```

Out[51]: $2e_1 - e_2 - 3e_3$

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 [52]:
```
w.set_name('w', latex_name=r'\omega')
w.disp()
```

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

Module operations are implemented, independently of the bases:

In [53]:
```
s = u + 3*v
print(s)
```

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

In [54]:
```
s.disp()
```

Out[54]: $10e_1 - 6e_2 + 28e_3$

In [55]:
```
s.disp(f)
```

Out[55]: $-10f_1 + 54f_2 - 40f_3$

In [56]:
```
s = u - v
print(s)
```

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

In [57]:
```
s.disp()
```

Out[57]: $u - v = -2e_1 + 10e_2 + 24e_3$

In [58]:
```
s.disp(f)
```

Out[58]: $u - v = 2f_1 - 14f_2 + 20f_3$

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

In [59]:
```
v.components(f)
```

Out[59]:
```
1-index components w.r.t. Basis (f_1,f_2,f_3) on the Rank-3 free modul
```

A shortcut is `comp()`:

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

Out[60]: True

In [61]:
```
for i in M.irange():
    print(v.comp(f)[i])
```

```
-3
17
-15
```

In [62]:
```
v.comp(f)[:]
```

Out[62]: $[-3, 17, -15]$

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

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

Out[63]:
$$\begin{array}{lcl} v^{1} & = & -3 \\ v^{2} & = & 17 \\ v^{3} & = & -15 \end{array}$$

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

In [64]:
```
v[f,2]
```

Out[64]: 17

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

Out[65]: $[-3, 17, -15]$

For the default basis, the basis can be omitted:

In [66]:
```
v[:]
```

Out[66]: $[3, -4, 1]$

In [67]:
```
v[2]
```

Out[67]: -4

A specific module element is the zero one:

In [68]:
```
print(M.zero())
```

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

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

Out[69]: $[0, 0, 0]$

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

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

In [71]:
```
v + M.zero() == v
```

Out[71]: True

Linear forms

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

In [72]:
```
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 [73]:
```
a[:] = [2,-1,3]
a.disp()
```

Out[73]: $a = 2e^1 - e^2 + 3e^3$

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 [74]:
```
ed = e.dual_basis()
ed
```

Out[74]: $\left(e^1, e^2, e^3\right)$

In [75]:
```
print(ed[1])
```

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

In [76]:
```
ed[1](e[1]), ed[1](e[2]), ed[1](e[3])
```

Out[76]: $(1, 0, 0)$

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

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

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

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

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

In [79]:
```
a[f,:] = [2,-1,3]
a.disp(f)
```

Out[79]: $a = 2f^1 - f^2 + 3f^3$

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 [80]:
```
a.disp()
```

Out[80]: $a = -36e^1 - 9e^2 + 4e^3$

By definition, linear forms belong to the dual module:

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

Out[81]: M^*

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

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

In [83]:
```
a.parent() is M.dual()
```
Out[83]: True

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

In [84]:
```
isinstance(M.dual(), FiniteRankFreeModule)
```
Out[84]: True

In [85]:
```
M.dual().rank()
```
Out[85]: 3

Linear forms map module elements to ring elements:

In [86]:
```
a(v)
```
Out[86]: -68

In [87]:
```
a(u)
```
Out[87]: 10

in a linear way:

In [88]:
```
a(u+2*v) == a(u) + 2*a(v)
```
Out[88]: True

Alternating forms

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

In [89]:
```
b = M.linear_form('b')
b[:] = [-4,2,5]
```

and take its exterior product with the linear form a:

In [90]:
```
c = a.wedge(b)
print(c)
c
```
```
Alternating form a/\b of degree 2 on the Rank-3 free module M over the
Integer Ring
```
Out[90]: $a \wedge b$

In [91]:
```
c.disp()
```
Out[91]: $a \wedge b = -108e^1 \wedge e^2 - 164e^1 \wedge e^3 - 53e^2 \wedge e^3$

In [92]:
```
c.disp(f)
```
Out[92]: $a \wedge b = 12f^1 \wedge f^2 + 70f^1 \wedge f^3 - 53f^2 \wedge f^3$

In [93]:
```
c(u,v)
```
Out[93]: 8894

c is antisymmetric:

In [94]:
```
c(v,u)
```
Out[94]: -8894

and is multilinear:

In [95]:
```
c(u+4*w,v) == c(u,v) + 4*c(w,v)
```
Out[95]: True

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

In [96]:
```
c(u,v) == a(u)*b(v) - b(u)*a(v)
```
Out[96]: True

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

In [97]:
```
c == a*b - b*a
```
Out[97]: 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 [98]:
```
c.parent()
```
Out[98]: $\Lambda^2\left(M^*\right)$

In [99]:
```
print(c.parent())
```
```
2nd exterior power of the dual of the Rank-3 free module M over the Int
eger Ring
```

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

In [100]:
```
c.tensor_type()
```
Out[100]: $(0,2)$

whose components with respect to any basis are antisymmetric:

In [101]:
```
c[:]  # components with respect to the default basis (e)
```
Out[101]:
$$\left(\begin{array}{rrr} 0 & -108 & -164 \\ 108 & 0 & -53 \\ 164 & 53 & 0 \end{array}\right)$$

In [102]:
```
c[f,:] # components with respect to basis f
```

Out[102]:
$$\begin{pmatrix} 0 & 12 & 70 \\ -12 & 0 & -53 \\ -70 & 53 & 0 \end{pmatrix}$$

In [103]:
```
c.comp(f)
```

Out[103]:
```
Fully antisymmetric 2-indices components w.r.t. Basis (f_1,f_2,f_3) or
```

An alternating form can be constructed from scratch:

In [104]:
```
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 [105]:
```
c1[1,2] = -108
c1[1,3] = -164
c1[2,3] = -53
```

Then

In [106]:
```
c1[:]
```

Out[106]:
$$\begin{pmatrix} 0 & -108 & -164 \\ 108 & 0 & -53 \\ 164 & 53 & 0 \end{pmatrix}$$

In [107]:
```
c1 == c
```

Out[107]: True

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

In [108]:
```
c.comp(e)._comp
```

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

In [109]:
```
c.comp(f)._comp
```

Out[109]: $\{(1,2):12,(1,3):70,(2,3):-53\}$

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 [110]:
```
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 t
he Integer Ring
```

In [111]:
```
s.disp()
```

Out[111]: $d \wedge a \wedge b = -707e^1 \wedge e^2 \wedge e^3$

In [112]:
```
s.disp(f)
```

Out[112]: $d \wedge a \wedge b = 707f^1 \wedge f^2 \wedge f^3$

In [113]:
```
s(e[1], e[2], e[3])
```

Out[113]: -707

In [114]:
```
s(f[1], f[2], f[3])
```

Out[114]: 707

s is antisymmetric:

In [115]:
```
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[115]: $(-144228, 144228, -144228, 144228, -144228, 144228)$

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 [116]:
```
M.tensor_module(1,0) is M
```

Out[116]: True

In [117]:
```
v.tensor_type()
```

Out[117]: $(1, 0)$

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

In [118]:
```
a in M.tensor_module(0,1)
```

Out[118]: True

Note that, at the Python level, we do *not* have the identification of $T^{(0,1)}(M)$ with M^*:

In [119]:
```
M.tensor_module(0,1) is M.dual()
```

Out[119]: False

This is because $T^{(0,1)}(M)$ and M^* are different objects:

In [120]:
```
type(M.tensor_module(0,1))
```

Out[120]: <class 'sage.tensor.modules.tensor_free_module.TensorFreeModule_with_

In [121]:
```
type(M.dual())
```

Out[121]: <class 'sage.tensor.modules.ext_pow_free_module.ExtPowerFreeModule_w:

However, we have coercion (automatic conversion) of elements of M^* into elements of $T^{(0,1)}(M)$:

In [122]:
```
M.tensor_module(0,1).has_coerce_map_from(M.dual())
```

Out[122]: True

as well as coercion in the reverse direction:

In [123]:
```
M.dual().has_coerce_map_from(M.tensor_module(0,1))
```

Out[123]: 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 [124]:
```
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 [125]:
```
t[e,1,2] = -3
```

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

In [126]:
```
t[1,2] = -3
```

The unset components are zero:

In [127]:
```
t[:]
```

Out[127]:
$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Components can be set at any time:

In [128]:
```
t[2,3] = 4
t[:]
```

Out[128]:
$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

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

In [129]:
```
t[f,:]
```

Out[129]:
$$\begin{pmatrix} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{pmatrix}$$

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

In [130]:
```
t.display()
```

Out[130]: $t = -3e_1 \otimes e^2 + 4e_2 \otimes e^3$

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 [131]:
```
t.display(f)
```

Out[131]:
$$t = 6f_1 \otimes f^1 + 3f_1 \otimes f^2 + 3f_1 \otimes f^3 - 108f_2 \otimes f^1 - 6f_2 \otimes f^2 + 6f_2 \otimes f^3 + 80f_3 \otimes f^1 + 8f_3 \otimes f^2$$

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

In [132]:
```
t(a,v)
```

Out[132]: -468

In [133]:
```
t(a,v).parent()
```

Out[133]: $\mathbf{Z}$

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 [134]:
```
tt = End(M)(t)
print(tt)
```

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

In [135]:
```
tt.parent()
```

Out[135]: $\mathrm{Hom}(M, M)$

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 [136]:
```
tt.matrix(e)
```

Out[136]:
$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

In [137]:
```
t[e,:]
```

Out[137]:

$$\begin{pmatrix} 0 & -3 & 0 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

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

Out[138]: True

In [139]:
```
tt.matrix(f)
```

Out[139]:

$$\begin{pmatrix} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{pmatrix}$$

In [140]:
```
t[f,:]
```

Out[140]:

$$\begin{pmatrix} 6 & 3 & 3 \\ -108 & -6 & 6 \\ 80 & 8 & 0 \end{pmatrix}$$

In [141]:
```
tt.matrix(f) == t[f,:]
```

Out[141]: True

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

In [142]:
```
tt(v)
```

Out[142]: $t(v)$

In [143]:
```
tt(v).parent()
```

Out[143]: M

In [144]:
```
tt(v).disp()
```

Out[144]: $t(v) = 12e_1 + 4e_2$

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

In [145]:
```
t in M.tensor_module(1,1)
```

Out[145]: True

or, in Sage terminology,

In [146]:
```
t.parent() is M.tensor_module(1,1)
```

Out[146]: True

$T^{(1,1)}(M)$ is itself a free module of finite rank over $\mathbb{Z}$:

In [147]:
```
isinstance(M.tensor_module(1,1), FiniteRankFreeModule)
```
Out[147]: True

In [148]:
```
M.tensor_module(1,1).base_ring()
```
Out[148]: $\mathbf{Z}$

In [149]:
```
M.tensor_module(1,1).rank()
```
Out[149]: 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 [150]:
```
ta = t*a
print(ta)
```
```
Type-(1,2) tensor t*a on the Rank-3 free module M over the Integer Ring
```

In [151]:
```
ta
```
Out[151]: $t \otimes a$

In [152]:
```
ta.disp()
```
Out[152]:
$$t \otimes a = 108 e_1 \otimes e^2 \otimes e^1 + 27 e_1 \otimes e^2 \otimes e^2 - 12 e_1 \otimes e^2 \otimes e^3 - 144 e_2 \otimes e^3 \otimes e^1 - 36 e_2 \otimes e^3 \otimes e^2 + 16 e_2 \otimes e^3 \otimes e^3$$

In [153]:
```
ta.disp(f)
```
Out[153]:
$$t \otimes a = 12 f_1 \otimes f^1 \otimes f^1 - 6 f_1 \otimes f^1 \otimes f^2 + 18 f_1 \otimes f^1 \otimes f^3 + 6 f_1 \otimes f^2 \otimes f^1 - 3 f_1 \otimes f^2 \otimes f^2 + 9 f_1 \otimes f^2 \otimes f^3 + 6 f_1 \otimes f^3 \otimes f^1 - 3 f_1 \otimes f^3 \otimes f^2 + 9 f_1 \otimes f^3 \otimes f^3 - 216 f_2 \otimes f^1 \otimes f^1 + 108 f_2 \otimes f^1 \otimes f^2 - 324 f_2 \otimes f^1 \otimes f^3 - 12 f_2 \otimes f^2 \otimes f^1 + 6 f_2 \otimes f^2 \otimes f^2 - 18 f_2 \otimes f^2 \otimes f^3 + 12 f_2 \otimes f^3 \otimes f^1 - 6 f_2 \otimes f^3 \otimes f^2 + 18 f_2 \otimes f^3 \otimes f^3 + 160 f_3 \otimes f^1 \otimes f^1 - 80 f_3 \otimes f^1 \otimes f^2 + 240 f_3 \otimes f^1 \otimes f^3 + 16 f_3 \otimes f^2 \otimes f^1 - 8 f_3 \otimes f^2 \otimes f^2 + 24 f_3 \otimes f^2 \otimes f^3$$

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

In [154]:
```
ta[:] # components w.r.t. the default basis (e)
```
Out[154]:
$$[[[0,0,0],[108,27,-12],[0,0,0]],[[0,0,0],[0,0,0],[-144,-36,16]],[[0,0,0],[0,0,0],[0,0,0]]]$$

In [155]:
```
ta[f,:] # components w.r.t. basis f
```
Out[155]:

$$[[[12, -6, 18], [6, -3, 9], [6, -3, 9]], [[-216, 108, -324], [-12, 6, -18], [12, -6, 18]], [[160, -80, 240], [16, -8, 24], [0, 0, 0]]]$$

Each component ca be accessed individually:

In [156]:
```
ta[1,2,3] # access to a component w.r.t. the default basis
```
Out[156]: -12

In [157]:
```
ta[f,1,2,3]
```
Out[157]: 9

In [158]:
```
ta.parent()
```
Out[158]: $T^{(1,2)}(M)$

In [159]:
```
ta in M.tensor_module(1,2)
```
Out[159]: True

The tensor product is not commutative:

In [160]:
```
print(a*t)
```
```
Type-(1,2) tensor a*t on the Rank-3 free module M over the Integer Ring
```

In [161]:
```
a*t == t*a
```
Out[161]: False

Forming a tensor of rank 4:

In [162]:
```
tav = ta*v
print(tav)
```
```
Type-(2,2) tensor t*a*v on the Rank-3 free module M over the Integer Ri
ng
```

In [163]:
```
tav.disp()
```
Out[163]:

$$t\otimes a\otimes v = 324e_1\otimes e_1\otimes e^2\otimes e^1 + 81e_1\otimes e_1\otimes e^2\otimes e^2 - 36e_1\otimes e_1\otimes e^2\otimes e^3 - 432e_1\otimes e_2\otimes e^2\otimes e^1 - 108e_1\otimes e_2\otimes e^2\otimes e^2 + 48e_1\otimes e_2\otimes e^2\otimes e^3 + 108e_1\otimes e_3\otimes e^2\otimes e^1 + 27e_1\otimes e_3\otimes e^2\otimes e^2 - 12e_1\otimes e_3\otimes e^2\otimes e^3 - 432e_2\otimes e_1\otimes e^3\otimes e^1 - 108e_2\otimes e_1\otimes e^3\otimes e^2 + 48e_2\otimes e_1\otimes e^3\otimes e^3 + 576e_2\otimes e_2\otimes e^3\otimes e^1 + 144e_2\otimes e_2\otimes e^3\otimes e^2 - 64e_2\otimes e_2\otimes e^3\otimes e^3 - 144e_2\otimes e_3\otimes e^3\otimes e^1 - 36e_2\otimes e_3\otimes e^3\otimes e^2 + 16e_2\otimes e_3\otimes e^3\otimes e^3$$

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 [164]:
```
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[164]:
$$\begin{pmatrix} 0 & 2 & 0 \\ -4 & 8 & 0 \\ 0 & 2 & -6 \end{pmatrix}$$

In [165]:
```
s = g.symmetrize() ; s
```

Out[165]: Symmetric bilinear form on the Rank-3 free module M over the Integer Ring

In [166]:
```
s.symmetries()
```

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

In [167]:
```
s[:]
```

Out[167]:
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 8 & 1 \\ 0 & 1 & -6 \end{pmatrix}$$

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

In [168]:
```
s == g.symmetrize(0,1)
```

Out[168]: True

One may use index notation to specify the symmetry:

In [169]:
```
s == g['_(ab)']
```

Out[169]: True

In [170]:
```
s == g['_{(ab)}'] # LaTeX type notation
```

Out[170]: True

Of course, since s is already symmetric:

In [171]:
```
s.symmetrize() == s
```

Out[171]: True

The antisymmetrization proceeds accordingly:

In [172]:
```
s = g.antisymmetrize() ; s
```

Out[172]: Alternating form of degree 2 on the Rank-3 free module M over the Integer Ring

In [173]:
```
s.symmetries()
```

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

In [174]:
```
s[:]
```

Out[174]:
$$\begin{pmatrix} 0 & 3 & 0 \\ -3 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

In [175]:
```
s == g.antisymmetrize(0,1)
```

Out[175]: True

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

In [176]:
```
s == g['_[ab]']
```

Out[176]: True

In [177]:
```
s == g['_{[ab]}'] # LaTeX type notation
```

Out[177]: True

Of course, since s is already antisymmetric:

In [178]:
```
s == s.antisymmetrize()
```

Out[178]: True

Tensor contractions

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

In [179]:
```
t.contract(v)
```

Out[179]: 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 [180]:
```
t.contract(v)[i] == sum(t[i,j]*v[j] for j in M.irange())
```

Out[180]: True

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

In [181]:
```
t.contract(v) == tt(v)
```

Out[181]: True

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

In [182]:
```
t['^i_j']*v['j'] == t.contract(v)
```

Out[182]: True

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

In [183]:
```
a.contract(v)
```

Out[183]: -68

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

In [184]:
```
a.contract(v) == a(v)
```

Out[184]: 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 [185]:
```
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 [186]:
```
z = t.contract(0, ta, 1)  # 0 -> first index of t, 1 -> second index of
 ta
print(z)
```

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

or, in terms of index notation:

In [187]:
```
z1 = t['^l_j']*ta['^i_lk']
z1 == z
```

Out[187]: True

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

In [188]:
```
t.contract?
```

In [ ]:
```

```