cmulTensor11

File: cmulTensor11.mw

Date: June 28, 2012

Authors: Rafal Ablamowicz & Bertfried Fauser (to accompany "Using Periodicity Theorems for Computations in Higher Dimensional Clifford Algebras")

Description: cmulTensor.mw describes how to set up and calculate with tensor product Clifford algebras using a periodicity theorems (here the CL_1,1 case). The tensor product is exported from the Bigebra package, computations in Clifford algebras to different bilinear forms are done using Clifford.

Provides:
-- cmulTensor : Clifford product on a tensor product Clifford algebra CL_p,q (x) CL_1,1
-- Tgradeinv: grade involution for the tensor product Clifford algebra
-- Treversion: (bilinear form dependent) grade reversion on the tensor product Clifford algebra
-- bas2Tbas: graded algebra morphism from a Grassmann basis over \( \Lambda (V1+V2) \) to the tensor basis of \( \Lambda V1 (x) \Lambda V2 \)
-- Tbas2bas: inverse graded algebra morphism to bas2Tbas

Benchmarks: We provide some benchmarks to show that the Clifford product on the tensor algebra is roughly (in the symbolic case) as fast (or even a bit faster) than the Clifford product directly computed (in dim V=dim V1+dim V1<= 9).

Setup Clifford & Bigebra and Helper function:

First we need to load Clifford and Bigebra. Maple packages are loaded using the >with(<package>); command. Functions of packages are also available via their long form >packagename:-function (<args>); Clifford and Bigebra come with a large set of help pages, which can be accessed for example by >?Clifford; or ?command; Most (each) command comes with an explanation who to use it, and some examples. The help pages also contain test cases so that migrating the packages to new Maple versions undergoes detailed testing.

The definition of the actual Clifford algebras in use will be done below when testing/proving some facts of procedures we are going to define.

> restart:with(Clifford);with(Bigebra);with(Cliplus);
[&m, Bsignature, CLIFFORD_ENV, Kfield, LC, LCQ, RC, RCQ, RHnumber, adfmatrix, all_sigs, beta_minus, beta_plus, buildm, bygrade, c_conjug, cbasis, cdfmatrix, cexp, cexpQ, cinv, clibilinear, clcollect, clidata, clilinear, climpoly, cliparse, cliremove, clisolve, clisort, clierms, cmul, cmulNUM, cmulQ, cmulRS, cmulgen, cocycle, commutingelements, conjugation, dfmatrix, diagonalize, displayid, extract, factoridempotent, find1str, findbasis, gradeinv, init, isVahlenmatrix, isproduct, makealiases, makeclibasmon, matKrepr, maxgrade, maxindex, mdfmatrix, minimalideal, ord, permsign, pseudodet, q_conjug, qdisplay, qinv, qmul, qnorm, rd_clibasmon, rd_climon, rd_clipolynom, reorder, reversion, rmulm, rot3d, scalarpart, sexp, specify_constants, spinorKbasis, spinorKrepr, squaremodf, subs_clipolynom, useproduct, vectorpart, version, wedge, wexp]

Increase verbosity by infolevel[\`function\`]=val -- use online help > ?Bigebra[help]
[&cco, &gco, &gco_d, &gco_pl, &map, &v, EV, VERSION, bracket, contract, drop_t, eps, gantipode, gco_unit, gswitch, hodge, linop, linop2, lists2mat, lists2mat2, make_BI_Id]
Climus ver. 13.2 of 3/24/2012 has been loaded. Definitions for type/climon and type/clipolynom now include &C and &C[K]. Type ? cliprod for help.

[LCbig, RCbig, clibasis, clieval, cliexpand, climul, clirev, dottedcbasis, dwedge, makeclialiases] (1)

Helper routine here:

```
mergeBs:=proc(b1,b2)
local row,col,Zero;
row:=linalg[coldim](b1);
col:=linalg[rowdim](b2);
Zero:=linalg[matrix](row,col,(i,j)->0);
linalg:-blockmatrix(2,2,[b1,Zero,linalg[transpose](Zero),b2]);
end proc:
```

---

**Vector space isomorphisms: bas2Tbas and Tbas2bas**

We define two mutually inverse graded vector space maps (isomorphisms)

\[
\begin{align*}
\text{bas2Tbas}: & \quad \text{CL}_{p+1,q+1} \rightarrow \text{CL}_{p,q} (x) \quad \text{CL}_1,1 \\
\text{Tbas2bas}: & \quad \text{CL}_{p,q} (x) \quad \text{CL}_1,1 \rightarrow \text{CL}_{p+1,q+1}
\end{align*}
\]

To show that these maps are graded isomorphisms, we need to show that their compositions fulfill

\[
\begin{align*}
\text{Tbas2bas} \circ \text{bas2Tbas} = \text{Id}_\text{bas} \\
\text{bas2Tbas} \circ \text{Tbas2bas} = \text{Id}_\text{Tbas}
\end{align*}
\]

as we work in finite dimensions, to show one of these equations is sufficient.

The map bas2Tbas is defined as follows on generators, where omega is the volume element of CL_1,1:

\[
\begin{align*}
\text{Id} & \rightarrow \ \&\text{t(Id,Id)}  \\
\text{ei} & \rightarrow \ \&\text{t(ei,omega)} \quad \text{if ei in } V_1 \quad (\text{that is } 1 \leq i \leq (p+q))  \\
\text{ej} & \rightarrow \ \&\text{t(Id,ej)} \quad \text{if ej in } V_2 \quad (\text{that is } p+q < j \leq (p+q)+(r+s))
\end{align*}
\]

and is extended multiplicatively and linearly to the whole vectors space \(\Lambda(V)\) underlying CL_p+r,q+s.

---

```
# bas2Tbas_mon : iso on monomials, handles Id correctly
#
bas2Tbas_mon:=proc(x)
```
local N1,lst,lst1,lst2,i,omega,k;
N1:=linalg:-rowdim(B1);
lst:=Clifford:-extract(x,'integer');
lst1,lst2:=[],[];
for i from 1 to nops(lst) do
    if N1<lst[i] then
        lst2:=[op(lst2),lst[i]-N1];
    else
        lst1:=[op(lst1),lst[i]];
    end if;
end do;
#
# hack due to an problem in cmul[B] with more than two arguments
#
omega:=Id;
k:=nops(lst1);
while k>0 do
    k:=k-1;
    omega:=cmul[B2](omega,e1we2);
end do:
&t(makeclibasmon(lst1),cmul[B2](omega,makeclibasmon(lst2)));
end proc:
#
# bas2Tbas: algebra isomorphism from CL_p+1,q+1 -> CL_p,1 (x)
# CL_1,1
#
bas2Tbas:=proc(x)
    local cf,term;
    if type(x,'+' ) then
        return map(procname,x);
    elif type(x,'*') then
        term,cf:=selectremove(type,x,clibasmon);
        return expand(cf*bas2Tbas_mon(term));
    else
        return bas2Tbas_mon(x);
    end if;
end proc:
#
# bas2Tbas_mon : iso on monomials, handles Id correctly
#
bas2Tbas_mon:=proc(x,y)
local N1,lst,lst1,lst2,i,k,omega,cf,term,yy,ylst,res;
N1:=linalg:-rowdim(B1);
#  #  hack due to an problem in cmul[B] with more than two arguments
#  
lst1:=Clifford:-extract(x,'integer');
omega:=Id;
k:=nops(lst1);
while k>0 do
  k:=k-1;
  omega:=cmul[B2](omega,e1we2);
end do:
yy:=cmul[B2](omega,y);
#
##
if type(yy,’+’) then
  ylst:=[op(yy)];
  res:=[];
  for i from 1 to nops(ylst) do
    term,cf:=selectremove(type,ylst[i],clibasmon);
    res:=[op(res),[cf,makeclibasmon(map(n->n+N1,Clifford:-
extract(term,'integer')))]];
  end do;
else
  ylst:=res;
elif type(yy,’*’) then
  term,cf:=selectremove(type,yy,clibasmon);
  ylst:=[[cf,makeclibasmon(map(n->n+N1,Clifford:-
extract(term,'integer')))]];
else
  ylst:=[[1,makeclibasmon(map(n->n+N1,Clifford:-
extract(yy,'integer')))]];
end if;
#
#  add(ylst[i][1]*wedge(x,ylst[i][2]),i=1..nops(ylst));
#
end proc:
#
#  Tbas2bas: algebra isomorphism from CL_p,1 (x) CL_1,1 ->
#  CL_p+1,q+1
#
Tbas2bas:=proc(x)
  expand(eval(subs(`&t`=Tbas2bas_mon,x)));
Proof of vector space isomorphism:

We prove the isomorphism in (a) particular case(s), where we use arbitrary bilinear forms \( B_1 \) on \( V_1 \), and a normalized matrix \( B_2 \) on \( V_2 \) with \( \det(B)=-1 \), and signature\( (B)=(1,-1) \). We define first the basis for \( CL_{p+1,r+1} \), stored in bas1. Then we apply bas2Tbas to obtain Tbas, a basis for \( CL_{p,q} \times CL_1 \). Next we apply Tbas2bas to get bas2 in \( CL_{p+1,q+1} \). The fact that bas1=bas2 proves the isomorphism property. We do not show output, but the last statement showing the equality.

\[
\begin{align*}
N1:=3: N2:=2: \\
B1:=\text{linalg}[\text{matrix}] (N1,N1, (i,j)->\text{if } i<=j \text{ then } b[i,j] \text{ else } b[i,j] \text{ end if}); \quad \# \text{ works} \\
# B2:=\text{linalg}[\text{matrix}] (N2,N2, (i,j)->\text{if } i=j \text{ then } (-1)^{(i+1)} \text{ else } 0 \text{ end if}); \quad \# \text{ works} \\
# B2:=\text{linalg}[\text{matrix}] (N2,N2, (i,j)->\text{if } i<>j \text{ then } 1 \text{ else } 0 \text{ end if}); \; \# \text{ fails} \\
# B2:=\text{subs} (b=-1/2, \text{linalg}[\text{matrix}] (N2,N2, [1,0,0,-1])); \\
# B2:=\text{linalg}[\text{matrix}] (N2,N2, [(b-1),b,b,(b+1)]); \quad \# \text{ works} \\
# \text{dim}_V:=N1+N2: \\
B:=\text{mergeBs}(B1,B2); \\
B1:=
\begin{bmatrix}
 b_{1,1} & b_{1,2} & b_{1,3} \\
 b_{2,1} & b_{2,2} & b_{2,3} \\
 b_{3,1} & b_{3,2} & b_{3,3}
\end{bmatrix} \\
B2:=
\begin{bmatrix}
 b-1 & b \\
 b & b+1
\end{bmatrix} \\
B:=
\begin{bmatrix}
 b_{1,1} & b_{1,2} & b_{1,3} & 0 & 0 \\
 b_{2,1} & b_{2,2} & b_{2,3} & 0 & 0 \\
 b_{3,1} & b_{3,2} & b_{3,3} & 0 & 0 \\
 0 & 0 & 0 & b-1 & b \\
 0 & 0 & 0 & b & b+1
\end{bmatrix}
\end{align*}
\]

> # test of isomorphisms
#
bas1:=cbasis(dim_V); 
## basis for \( CL_{p+r,q+s} \)
Definition of the tensor Clifford product

The (binary) tensor Clifford product \texttt{cmulTensor} is defined as follows. Let \( A \) be Clifford algebra, and \( B \) the Clifford algebra \( \text{CL}_{1,1} \) with Clifford multiplications \( m_A \) and \( m_B \), then the tensor product algebra \( A \times B \) has the product:

\[
m_{A (x) B} = (m'_A (x) m_B) \circ (1 (x) \text{\textbackslash switch} (x) 1)
\]

\[
m_{A (x) B}(X1 (x) X2, Y1 (x) Y2) = t(m'_A(X1,Y1), m_B(X2,Y2))
\]

where \( m'_A \) is the Clifford multiplication for the metric \( \lambda*B1 \), and \( \lambda \) is the square of the volume element, that is proportional to \( \det(B) \), in the \( \text{CL}_{1,1} \) case \( \lambda=1 \), and we ignore it. Using the facilities of Bigebra we can literally use this definition:

\[
\text{switch}(\text{\&t}(X1,X2,...,Xl), i) = \text{\&t}(X1,..,X(i-1),X(i+1),Xi,X(i+2),...Xl)
\]

remember that \texttt{cmul}[B](x,y) takes an argument `B` holding a bilinear form, so that \( m'_A=\text{cmul}[B1] \) and \( m_B=\text{cmul}[B2] \) and we hand \texttt{cmulTensor} two bilinear forms defining the tensor product factor Clifford algebras \( A=\text{CL}_{p,q} \) and \( B=\text{CL}_{1,1} \).

**Note:** In this procedure it is silly to use switch to switch the basis elements, which could be done without additional effort by the \texttt{f4} helper routine (and thus speeding up the procedure). We leave the switch in the routine in this example document to show how the mathematical form can be directly implemented using Bigebra facilities. A speed optimized version would remove such glitches.
Proof: bas2GTbas (and its inverse GTbas2bas) is a graded Clifford algebra isomorphism

Let A=CL_p,q, B=CL_r,s. To show this we need to show that a diagram that has the two edges is commutative:

i) first multiply in A (x) B, then apply the isomorphism GTbas2bas
ii) first apply to both arguments of the multiplication the isomorphism GTbas2bas, and then multiply in CL_p+r,q+s

To show this, we use a multiplication table, computing for all basis elements. This gives two matrices mat1 and mat2, and their equivalence (needs to collect terms first) shows that the diagram described above commutes. That is, the difference mat1-mat2 is the zero matrix. Note that we compute with general (symmetric) bilinear forms.

We do not show output, but only the logical test at the end showing the matrix identity to be true.

> N1:=3:N2:=2:
B1:=linalg[Matrix](N1,N1,(i,j)->if i<=j then b[i,j] else b[i,j] end if);
# B2:=linalg[Matrix](N2,N2,(i,j)->if i=j then (-1)^(i+1) else 0 end if);
# B2:=linalg[Matrix](N2,N2,(i,j)->if i<>j then 1 else 0 end if):
B2:=linalg[Matrix](2,2,[b+1,b,b,b-1]);

> B:=mergeBs(B1,B2);

\[
\begin{align*}
B &:= \begin{bmatrix}
0 & 0 & 0 & b + 1 & b & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & b_1,1 & b_1,2 & b_1,3 & 0 & 0 & 0 \\
0 & 0 & 0 & b_2,1 & b_2,2 & b_2,3 & 0 & 0 & 0 \\
0 & 0 & 0 & b_3,1 & b_3,2 & b_3,3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & b + 1 & b & b - 1
\end{bmatrix} \\
B2 &:= \begin{bmatrix}
0 & 0 & 0 & b + 1 & b & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & b_1,1 & b_1,2 & b_1,3 & 0 & 0 & 0 \\
0 & 0 & 0 & b_2,1 & b_2,2 & b_2,3 & 0 & 0 & 0 \\
0 & 0 & 0 & b_3,1 & b_3,2 & b_3,3 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & b + 1 & b & b - 1
\end{bmatrix}
\end{align*}
\]
\[
Tbas := \text{map}(\text{bas2Tbas}, \text{bas});
\]
\[
Tbas := \left[\text{Id} \& \text{Id}, e1 \& \text{elwe2}, e2 \& \text{elwe2}, e3 \& e1we2, Id \& e1, Id \& e2, e1we2 \& Id, e1we3 \& Id, b e1 \& t e1 + (-b - 1) e1 \& t e2, (b - 1) e1 \& t e1 - e1 \& t e2 b, e2we3 \& Id, b e2 \& t e1 + (-b - 1) e2 \& t e2, (b - 1) e2 \& t e1 - e2 \& t e2 b, b e3 \& t e1 + (-b - 1) e3 \& t e2, (b - 1) e3 \& t e1 - e3 \& t e2 b, Id \& t e1we2, e1we2we3 \& t e1we2, e1we2 \& t e1, e1we2 \& t e2, e1we3 \& t e1, e1we3 \& t e2, e1 \& t Id, e2we3 \& t e1, e2we3 \& t e2, e2 \& t Id, e3 \& t e1we2we3 \& t e1 + (-b - 1) e1we2we3 \& t e2, (b - 1) e1we2we3 \& t e1 - e1we2we3 \& t e2 b, e2we3 \& t e1we2, e2we3 \& t e2we3 \& t e1we2, e1we2we3 \& t Id\right]
\]

\[
\text{mat1} := \text{linalg[matrix]}(\text{nops(bas)}, \text{nops(bas)}, (i, j) \rightarrow \text{cmul}[B](\text{bas}[i], \text{bas}[j]));
\]
\[
\text{mat2} := \text{linalg[matrix]}(\text{nops(Tbas)}, \text{nops(Tbas)}, (i, j) \rightarrow \text{clicollect}(
\text{Tbas2bas}(\text{cmulTensor(Tbas}[i], \text{Tbas}[j], B1, B2))));
\text{evalb(\text{convert(\text{map(\text{clicollect@simplify, \text{evalm(mat1-mat2))}, set}=\{0\})}});
\]

\[
\text{true}
\]

**Benchmarks : cmul versus cmulTensor (CL_p,q (x) CL_1,1 case)**

We compute on a set of randomly chosen (equally distributed over the basis) basis monomials, and compare the timings.

**Note1:** Clifford comes with several multiplication algorithms which perform differently in different cases. The user can switch between them (see >?Clifford:-useproduct) or even define own product routine (not necessarily computing a Clifford product). By default Clifford uses:
-- cmulRS, using the Rota Stein Hopf algebra based algorithm, optimised for symbolic bilinear forms with off diagonal entries
-- cmulNUM, using the Chevalley construction, optimised for bilinear forms with numeric entries and possibly many zeros in it.
-- cmulWalsh, experimental multiplication, optimized for diagonal bilinear forms, that is orthogonal generators (not yet part of Clifford, needs to be loaded by the user).
Our benchmarks are for the symbolic case using cmulRS.

**Note2:** Clifford type checking and functions heavily rely on the remember mechanism of Maple in internal functions to speed up (significantly) computations. For that reason recomputing a result is much faster than doing it the first time. Hence computing a loop over the same calculation would just benchmark largely the overhead of type checking and function calls done internally in Clifford. For that reason we use randomly chosen elements for testing.

**Note3:** We would expect that cmulTensor is slower than cmul, as it operates on more complex data types. However, the symbolic forms involved are of dimension dim B1=N1, dim B2=N2 and dim B=N1+N2, so cmul[B] computes on all the (zero) off diagonal elements in the block structure
\[
B = \begin{bmatrix} B1 & 0 \\ 0 & B2 \end{bmatrix}
\]
Now cmulGTensor does not compute these off diagonal terms, and hence gains some advantage. This effect becomes larger for larger N1,N2.

As this computation is randomized, it should be run several times (actually after restarting Maple) and the average time should be taken.
N1:=4; N2:=2; ## <= vary here the first!! index N1=p+q; N2 has to be 2
B1:=linalg[\text{matrix}](N1,N1,(i,j)\rightarrow\text{if } i\leq j \text{ then } b[i,j] \text{ else } b[i,j] \text{ end if});
# B2:=linalg[\text{matrix}](N2,N2,(i,j)\rightarrow\text{if } i=j \text{ then } (-1)^{(i+1)} \text{ else } 0 \text{ end if});
# B2:=linalg[\text{matrix}](N2,N2,(i,j)\rightarrow\text{if } i\neq j \text{ then } 1 \text{ else } 0 \text{ end if});  \quad \# \text{ fails}
B2:=linalg[\text{matrix}](2,2,[b+1,b,b,b-1]);  \quad \# \text{ general symmetric } B2
\text{with } \det(B2)=-1, \text{ signature}(B2)=(1,-1)
#
dim_V:=N1+N2:
B:=mergeBs(B1,B2);
#
# test the graded algebra isomorphism
# \quad \text{CL}_{p+r,q+s} \approx \text{CL}_{p,q}(x) \text{ CL}_{r,s}
#
bas:=cbasis(dim_V):
Tbas:=map(bas2Tbas,bas):
#
# benchmarks:
#
rnd:=rand(1..nops(GTbas)):
MaxIter:=200:
run1:=time():
for i from 1 to MaxIter do
\quad cmul[B](Tbas2bas(Tbas[rnd()]),Tbas2bas(Tbas[rnd()]));
end do:
printf("===============================\n");
printf(" cmul[B] needed %.2f sec\n",time()-run1);
#
run2:=time():
for i from 1 to MaxIter do
\quad Tbas2bas(cmulTensor(Tbas[rnd()]),Tbas[rnd()],B1,B2));
end do:
printf(" cmulTensor needed %.2f sec\n",time()-run2);
printf("===============================\n");

B2:= \begin{bmatrix} b+1 & b \\ b & b-1 \end{bmatrix}

cmul[B] needed .797 sec
cmulTensor needed .656 sec
Grade involution and reversion involutions for the tensor algebra $\text{CL}_p,q \times \text{CL}_1,1$

We extend the grade and reversion involutions to the tensor case as follows:

$T_{\text{grade inv}} = \text{grade inv}(x) \times \text{grade inv}$

$T_{\text{reversion}}[B]$ is more complex, as it needs a case distinction due to the degree of its first argument, see code.

The grade involution is straightforward. The reversion is complicated as it depends on the bilinear forms, and it needs case distinction by grade. We give three ways to code the $T_{\text{reversion}}$ functions.

```plaintext
> #
> # Tgradeinv : grade involution on $\text{CL}_p,q \times \text{CL}_r,s$
> #
> Tgradeinv:=proc(x)
>     local f2;
>     f2:=(a,b)->&t(gradeinv(a),gradeinv(b));
>     eval(subs(`&t`=f2,x));
> end proc:
> #
> # Tversion : reversion involution on $\text{CL}_p,q \times \text{CL}_r,s$
> #       !!! needs symmetric B1, B2 !!!
> #
> Tversion1:=proc(x,B1,B2)
>     local f2;
>     f2:=proc(a,b)
>         if nops(Clifford:-extract(b)) mod 2 = 0 then
>             ## versions 1) and 2)
>             return &t(reversion(b,B1),reversion(a,B2));
>         else
>             ## version 1)
>             return &t(gradeinv(reversion(b,B1)),(reversion(a,B2)));
>         end if;
>     end proc:
>     eval(subs(`&t`=f2,switch(x,1)));
> end proc:
> #
> Tversion2:=proc(x,B1,B2)
>     local f2;
>     f2:=proc(a,b)
>         if nops(Clifford:-extract(b)) mod 2 = 0 then
>             
```
Proof that Tgradeinv and Treversion are the correct involutions on $C = A (x) CL_{1,1}$

To proof that Tgradeinv and Treversion are the correct involutions on $A (x) B$, we need to verify that they coincide with the respective involutions on the ambient Clifford algebra. Doing so means that they need to commute with the graded algebra isomorphisms bas2Tbas and Tbas2bas. That is, we need to show that:

(i) $bas2Tbas \circ gradeinv = Tgradeinv \circ bas2GTbas$

(ii) $bas2Tbas \circ reversion[B] = Treversion[B1,B2] \circ bas2Tbas$

where we have written the dependency of the reversion involutions on the respective bilinear forms as
indicated. In actual code they are passed as a second (and third) argument, hence \texttt{Treversion(X, B1, B2)} will take the two bilinear forms B1,B2, making up the bilinear form B=B1+B2 (direct sum).

\textbf{Note:} If we choose a basis for the Clifford algebra C=\texttt{CL\_p+r,q+s} such that the bilinear form B does not decompose as a direct sum B=B1+B2, then the reversion needs to be modified. This needs a smash product structure on the tensor (R-matrix).

\begin{verbatim}
> N1:=3:N2:=2:
> B1:=linalg'[matrix](N1,N1,(i,j)->b[i,j]):  ## allow (nonsymmetric) general bilinear form
#   B2:=linalg'[matrix](N2,N2,[1,0,0,-1]   );  ##
> B2:=linalg'[matrix](N2,N2,(i,j)->if i<>j then 1 else 0 end if):;
#   fails
#
> dim_V:=N1+N2:
> B:=mergeBs(B1,B2):;
#  test the graded algebra isomorphism
#    CL_p+r,q+s ~= CL_p,q (x) CL_r,s
#
> bas:=cbasis(dim_V):
> Tbas:=map(bas2Tbas,bas):
> bas1:=map(bas2Tbas,map(gradeinv, bas)):
> bas2:=map(Tgradeinv,Tbas,B1,B2):
> printf("For general B=B1+B2 we have:\n");
> printf(" grade involution is an graded algebra isomorphism : %a\n",evalb(bas1=bas2));
>
> bas1:=map(bas2Tbas,map(reversion[B], bas)):
> bas2:=map(Treversion,Tbas,B1,B2):
> printf("reversion involution is an graded algebra isomorphism : %a\n",evalb(bas1=bas2));
> printf("="\n");
> bas;
> #bas1-bas2;

===============================================================
For general B=B1+B2 we have:
    grade involution is an graded algebra isomorphism : true
reversion involution is an graded algebra isomorphism : true
===============================================================
\end{verbatim}