This notebook:

- displays the help for the GA13 module, and its functions
- has a number of test cases.
- some manual tests that require visual verification.

These tests can also be used to see examples of how to use the GA functions and operators defined by the GA13 package.

In[1]:=

```
<< GA13`;

? GA13
(*?grade*)
? Scalar
? Vector
? Bivector
? Trivector
? Quadvector
? gradeQ
? scalarQ
? vectorQ
? bivectorQ
? trivectorQ
? quadvectorQ
? bladeQ
? gradeAnyQ
? notGradeQ
? GradeSelection
? ScalarSelection
? VectorSelection
? BivectorSelection
? TrivectorSelection
? QuadvectorSelection
? ScalarValue
? ScalarProduct
On[Assert]
```

GA13: An implementation of a Minkowski (CL(1,3)) Geometric Algebra.

Dirac matrices are used to represent the algebraic

elements. This provides an fairly efficient and compact representation
of the entire algebraic space. This representation unfortunately has a built in redundancy, since the complex 4x4 matrix has 32 degrees of freedom, while there are only 16 elements in the algebraic space.

Internally, a multivector is represented by a pair (grade, dirac–representation). The grade portion will be obliterated when adding objects that have different grade, or multiplying vectors or bivectors. When it is available, certain operations can be optimized. Comparison ignores the cached grade if it exists.

Elements of the algebra can be constructed with one of

```
Scalar[ v ]
Vector[ v, n ]
Bivector[ v, n, m ]
Trivector[ v, n, m, o ]
Quadvector[ v ]
```

Example:

```
m = Scalar[ Sin[ x ] ] + Vector[ Log[ z ], 3 ] + Trivector[ 7, 0, 1, 3 ] ;
m // StandardForm
```

```
> 7 e[ 123 ] + e[ 3 ] Log[ z ] + Sin[ x ]
```

A few operators are provided:

```
==        Compare two multivectors, ignoring the cached grade if any.
m1 + m2
m1 – m2
– m
st * vb   Scalars and trivectors can multiply vectors and bivectors in any order
vb1 ** vb1 Vectors and bivectors when multiplied have
   to use the NonCommutativeMultiply operator, but any grade object may also.
m1 . m2   Dot product. The functional form Dot[ m1, m2 ] may also be used.
m1 ^ m2   Wedgeproduct. Enter with m1 [ Esc ]^[ Esc ] m2. The functional form Wedge[ m1, m2 ]
<m>       Scalar selection. Enter with [ Esc ]<[ Esc ] m [ Esc ]>[ Esc
   ]. The functional form ScalarValue[ m ] may also be used. This returns the numeric
   (or expression) value of the scalar grade of the multivector, and not a grade[ ] object.
<m1,m2>   Scalar product. Enter with [ Esc ]<[ Esc ] m1,m2 [ Esc ]>[ Esc ]. The functional
   form ScalarProduct[ m1, m2 ] may also be used. This returns the numeric (or
   expression) value of the scalar product of the multivectors, and not a grade[ ] object.
```

Functions provided:

```
– GradeSelection
– ScalarSelection
– VectorSelection
– BivectorSelection
– TrivectorSelection
– QuadvectorSelection
– ScalarValue, < m >
– ScalarProduct, < m1, m2 >
```

The following built–in methods are overridden:

```
– TraditionalForm
```

– DisplayForm
– StandardForm

Internal functions:

– scalarQ
– vectorQ
– bivectorQ
– trivectorQ
– quadvectorQ
– bladeQ
– gradeAnyQ
– notGradeQ

TODO:

1) How to get better formatted output by default without using one of TraditionalForm, DisplayForm, StandardForm ?

2) Can a package have options (i.e. to define the name
of the e[ ] operator used in StandardForm that represents a basis vector).

3) proper packaging stuff: private for internals.

Scalar[v] constructs a scalar grade quantity with value v.

Vector[v, n], where n = {0,1,2,3} constructs a vector grade quantity with value v in direction n.

Bivector[v, n1, n2], where n1,n2 = {1,2,3} constructs a bivector grade quantity with value v in the plane n1,n2.

Trivector[v, k, l, m] constructs a trivector (pseudoscalar) grade quantity scaled by v.

Quadvector[v] constructs a quadvector (pseudoscalar) grade quantity scaled by v.

gradeQ[m, n] tests if the multivector m is of grade
n. n = –1 is used internally to represent values of more than one grade.

scalarQ[m] tests if the multivector m is of grade 0 (scalar)

vectorQ[m] tests if the multivector m is of grade 1 (vector)

bivectorQ[m] tests if the multivector m is of grade 2 (bivector)

trivectorQ[m] tests if the multivector m is of grade 3 (trivector)

quadvectorQ[m] tests if the multivector m is of grade 4 (quadvector)

bladeQ[m] tests if the multivector is of a single grade.

gradeAnyQ[ ]. predicate pattern match for grade[_]

notGradeQ[ ]. predicate pattern match for !grade[ ]

GradeSelection[m, k] selects the grade k elements from the multivector m. The selected result is represented internally as a grade[ ] type (so scalar selection is not just a number).

ScalarSelection[m] selects the grade 0 (scalar) elements from the multivector m. The selected result is represented internally as a grade[ ] type (not just a number or an expression).

VectorSelection[m] selects the grade 1 (vector) elements from the multivector m. The selected result is represented internally as a grade[ ] type.

BivectorSelection[m] selects the grade 2 (bivector) elements from the multivector m. The selected result is represented internally as a grade[ ] type.

TrivectorSelection[m] selects the grade 3 (trivector) element from the multivector m if it exists. The selected result is represented internally as a grade[ ] type (not just an number or expression).

QuadvectorSelection[m] selects the grade 4 (trivector) element from the multivector m if it exists. The selected result is represented internally as a grade[ ] type (not just an number or expression).

ScalarValue[m]. Same as AngleBracket[m], aka [Esc]<[Esc] m1 [Esc]>[Esc].

ScalarProduct[ ]. Same as AngleBracket[m1, m2], aka [Esc]<[Esc] m1, m2 [Esc]>[Esc].

Predicate tests (automatic)

In[26]:=
```
ClearAll[es, e0, e1, e2, e3, e01, e02, e03, e23, e31,
  e12, e10, e20, e30, e32, e13, e21, e123, e012, e013, e023, e0123];
es = Scalar[1];

e0 =  Vector[1, 0];
e1 = Vector[1, 1] ;
e2 = Vector[1, 2];
e3 = Vector[1, 3];

e01 = Bivector[1, 0, 1] ;
e02 = Bivector[1, 0, 2];
e03 = Bivector[1, 0, 3];
e23 = Bivector[1, 2, 3] ;
```

```
e31 = Bivector[1, 3, 1];
e12 = Bivector[1, 1, 2];

e10 = -e01 ;
e20 = -e02 ;
e30 = -e03 ;
e32 = -e23 ;
e13 = -e31 ;
e21 = -e12 ;

e123 = Trivector[1, 1, 2, 3];
e012 = Trivector[1, 0, 1, 2];
e013 = Trivector[1, 0, 1, 3];
e023 = Trivector[1, 0, 2, 3];

e0123 = Quadvector[1];

ClearAll[ms, msb, msbq, msbt, msbtq, msq, mst, mstq,
 msv, msvb, msvbq, msvbt, msvbtq, msvq, msvt, msvtq, mb, mbq,
 mbt, mbtq, mt, mtq, mv, mvb, mvt, mvq, mvbt, mvbtq, mvbq, mq]
ms = es;
msb = es + e23;
msbq = es + e23 + e0123;
msbt = es + e23 + e123;
msbtq = es + e23 + e123 + e0123;
msq = es + e0123;
mst = es + e123;
mstq = es + e123 + e0123;
msv = es + e1;
msvb = es + e1 + e23;
msvbq = es + e1 + e23 + e0123;
msvbt = es + e1 + e23 + e123;
msvbtq = es + e1 + e23 + e123 + e0123;
msvq = es + e1 + e123;
msvt = es + e1 + e123;
msvtq = es + e1 + e123 + e0123;
mb = e23;
mbq = e23 + e0123;
mbt = e23 + e123;
mbtq = e23 + e123 + e0123;
mt = e123;
mtq = e123 + e0123;
mv = e1;
mvb = e1 + e23;
```

```
mvt = e1 + e123;
mvq = e1 + e0123;
mvbt = e1 + e23 + e123;
mvbtq = e1 + e23 + e123 + e0123;
mvbq = e1 + e23 + e123;
mq = e0123;
```

```
{Assert[bladeQ[#]]} & /@ {es, e1, e2, e3, e23, e31, e12, e123};
{Assert[! bladeQ[#]]} & /@ {msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[gradeAnyQ[#]]} & /@ {e1, e2, e3, e23, e31,
   e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[! gradeAnyQ[#]]} & /@ {1, Sin[x], Exp[I theta]};
{Assert[! notGradeQ[#]]} & /@ {es, e1, e2, e3, e23, e31,
   e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[notGradeQ[#]]} & /@ {1, Sin[x], Exp[I theta]};
{Assert[gradeQ[#, 0]], Assert[scalarQ[#]]} & /@ {es};
{Assert[! gradeQ[#, 0]], Assert[! scalarQ[#]]} & /@ {e1, e2, e3, e23,
   e31, e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[gradeQ[#, 1]], Assert[vectorQ[#]]} & /@ {e1, e2, e3};
{Assert[! gradeQ[#, 1]], Assert[! vectorQ[#]]} & /@
  {es, e23, e31, e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[gradeQ[#, 2]], Assert[bivectorQ[#]]} & /@ {e23, e31, e12};
{Assert[! gradeQ[#, 2]], Assert[! bivectorQ[#]]} & /@
  {es, e1, e2, e3, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[gradeQ[#, 3]], Assert[trivectorQ[#]]} & /@ {e123};
{Assert[! gradeQ[#, 3]], Assert[! trivectorQ[#]]} & /@
  {es, e1, e2, e3, e23, e31, e12, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[gradeQ[#, -1]]} & /@ {msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt};
{Assert[! gradeQ[#, -1]]} & /@ {es, e1, e2, e3, e23, e31, e12, e123};

(*Grade selection tests.*)

{Assert[GradeSelection[#, 0] == es], Assert[ScalarSelection[#] == es]} & /@
  {es, msv, msb, mst, msvt, msvb, msbt};
{Assert[GradeSelection[#, 0] == 0], Assert[ScalarSelection[#] == 0]} & /@
  {e1, e2, e3, e23, e31, e12, e123, mvb, mvt, mbt, mvbt};

{Assert[GradeSelection[#, 1] == e1], Assert[VectorSelection[#] == e1]} & /@
  {e1, msv, mvb, mvt, msvb, msvt, mvbt};
{Assert[GradeSelection[#, 1] == 0], Assert[VectorSelection[#] == 0]} & /@
  {es, e23, e31, e12, e123, msb, mst, mbt, msbt};
```

```
{Assert[GradeSelection[#, 2] == e23], Assert[BivectorSelection[#] == e23]} & /@
  {e23, msb, mvb, mbt, msbt, mvbt, msvb};
{Assert[GradeSelection[#, 2] == 0], Assert[BivectorSelection[#] == 0]} & /@
  {es, e1, e2, e3, e123, msv, mst, mvt, msvt};

{Assert[GradeSelection[#, 3] == e123], Assert[TrivectorSelection[#] == e123]} & /@
  {e123, mst, mvt, mbt, msvt, msbt, mvbt};
{Assert[GradeSelection[#, 3] == 0], Assert[TrivectorSelection[#] == 0]} & /@
  {es, e1, e2, e3, e23, e31, e12, msv, msb, mvb, msvb};

(*Minus tests*)
Assert[-es == Scalar[-1]];
Assert[-e1 == Vector[-1, 1]];
Assert[-e2 == Vector[-1, 2]];
Assert[-e3 == Vector[-1, 3]];
Assert[-e23 == Bivector[-1, 2, 3]];
Assert[-e31 == Bivector[-1, 3, 1]];
Assert[-e12 == Bivector[-1, 1, 2]];
Assert[-e123 == Trivector[-1, 1, 2, 3]];
Assert[-msv == -es - e1];
Assert[-msb == -es - e23];
Assert[-mst == -es - e123];
Assert[-mvb == -e1 - e23];
Assert[-mvt == -e1 - e123];
Assert[-mbt == -e23 - e123];
Assert[-msvb == -es - e1 - e23];
Assert[-msvt == -es - e1 - e123];
Assert[-msbt == -es - e23 - e123];
Assert[-mvbt == -e1 - e23 - e123];


(* Scalar/Pseudoscalar multiplication tests*)

{Assert[(#[[1]]) (#[[2]]) == #[[3]]], Assert[(#[[2]]) (#[[1]]) == #[[3]]]} & /@
  {{2, es, Scalar[2]}, {2, e1, Vector[2, 1]}, {2, e2, Vector[2, 2]},
   {2, e3, Vector[2, 3]}, {2, e23, Bivector[2, 2, 3]}, {2, e31, Bivector[2, 3, 1]},
   {2, e12, Bivector[2, 1, 2]}, {2, e123, Trivector[2, 1, 2, 3]},
   {2, msv, 2 es + 2 e1}, {2, msb, 2 es + 2 e23}, {2, mst, 2 es + 2 e123},
   {2, mvb, 2 e1 + 2 e23}, {2, mvt, 2 e1 + 2 e123}, {2, mbt, 2 e23 + 2 e123},
   {2, msvb, 2 es + 2 e1 + 2 e23}, {2, msvt, 2 es + 2 e1 + 2 e123},
   {2, msbt, 2 es + 2 e23 + 2 e123}, {2, mvbt, 2 e1 + 2 e23 + 2 e123}} ;

(* Commutative multiplication pair tests *)
```

```
{ Assert[(#[[1]]) (#[[2]]) == #[[3]]],
    Assert[(#[[2]]) (#[[1]]) == #[[3]]],
    Assert[(#[[1]]) ** (#[[2]]) == #[[3]]],
    Assert[(#[[2]]) ** (#[[1]]) == #[[3]]]
   } & /@
  {
   {es, es, Scalar[1]},
   {es, e1, Vector[1, 1]},
   {es, e2, Vector[1, 2]},
   {es, e3, Vector[1, 3]},
   {es, e23, Bivector[1, 2, 3]},
   {es, e31, Bivector[1, 3, 1]},
   {es, e12, Bivector[1, 1, 2]},
   {es, e123, Trivector[1, 1, 2, 3]},
   {es, msv, es + e1},
   {es, msb, es + e23},
   {es, mst, es + e123},
   {es, mvb, e1 + e23},
   {es, mvt, e1 + e123},
   {es, mbt, e23 + e123},
   {es, msvb, es + e1 + e23},
   {es, msvt, es + e1 + e123},
   {es, msbt, es + e23 + e123},
   {es, mvbt, e1 + e23 + e123}
  };

{ Assert[(#[[1]]) ** (#[[2]]) == #[[3]]],
    Assert[(#[[2]]) ** (#[[1]]) == #[[3]]]
   } & /@
  {
   {Bivector[-1, 1, 2], Bivector[1, 3, 0], Quadvector[1]}
  };
```

Need to generalize this to Cl(1,3).

```
(*
(*Tests for (non-commutitive) multiplication, dot and wedge.*)
ClearAll[ mbasis, ptable, dtable, wtable, stable ] ;
mbasis={e1,e2,e3,e23,e31,e12,e123};

ptable=(*e1,e2,e3,e23,e31,e12,e123*)
 (*e1*){{es,e12,e13,e123,-e3,e2,e23},
  (*e2*){e21,es,e23,e3,e123,-e1,e31},
```

```
   (*e3*){e31,e32,es,-e2,e1,e123,e12},
   (*e23*){e123,-e3,e2,-es,e21,e31,-e1},
   (*e31*){e3,e123,-e1,e12,-es,e32,-e2},
   (*e12*){-e2,e1,e123,e13,e23,-es,-e3},
   (*e123*){e23,e31,e12,-e1,-e2,-e3,-es}};

dtable=(*e1,e2,e3,e23,e31,e12,e123*)
  (*e1*){{es,0,0,0,-e3,e2,e23},
   (*e2*){0,es,0,e3,0,-e1,e31},
   (*e3*){0,0,es,-e2,e1,0,e12},
   (*e23*){0,-e3,e2,-es,0,0,-e1},
   (*e31*){e3,0,-e1,0,-es,0,-e2},
   (*e12*){-e2,e1,0,0,0,-es,-e3},
   (*e123*){e23,e31,e12,-e1,-e2,-e3,-es}};

wtable=(*e1,e2,e3,e23,e31,e12,e123*)
  (*e1*){{0,e12,e13,e123,0,0,0},
   (*e2*){e21,0,e23,0,e123,0,0},
   (*e3*){e31,e32,0,0,0,e123,0},
   (*e23*){e123,0,0,0,0,0,0},
   (*e31*){0,e123,0,0,0,0,0},
   (*e12*){0,0,e123,0,0,0,0},
   (*e123*){0,0,0,0,0,0,0}};

stable=(*e1,e2,e3,e23,e31,e12,e123*)
  (*e1*){{1,0,0,0,0,0,0},
   (*e2*){0,1,0,0,0,0,0},
   (*e3*){0,0,1,0,0,0,0},
   (*e23*){0,0,0,-1,0,0,0},
   (*e31*){0,0,0,0,-1,0,0},
   (*e12*){0,0,0,0,0,-1,0},
   (*e123*){0,0,0,0,0,0,-1}};


Table[
  {
   Assert[ mbasis[[i]] ** mbasis[[j]] == ptable[[i,j]]],
   Assert[ NonCommutativeMultiply[ mbasis[[i]],mbasis[[j]]] == ptable[[i,j]]]
  },
  {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] ;

Table[
  {
   Assert[ mbasis[[i]] . mbasis[[j]] == dtable[[i,j]]],
```

```
  Assert[ Dot[mbasis[[i]], mbasis[[j]]] == dtable[[i,j]]]
 },
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] ;

Table[
 {
  Assert[ mbasis[[i]] ∧ mbasis[[j]] == wtable[[i,j]]],
  Assert[ Wedge[mbasis[[i]], mbasis[[j]]] == wtable[[i,j]]]
 },
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] ;

Table[
 {
  Assert[ ⟨mbasis[[i]] , mbasis[[j]]⟩ == stable[[i,j]]],
  Assert[ ScalarProduct[mbasis[[i]] , mbasis[[j]]] == stable[[i,j]]]
 },
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] ;

Table[
 {
  Assert[ ⟨mbasis[[i]] ** mbasis[[j]]⟩ == stable[[i,j]]],
  Assert[ ScalarValue[ (mbasis[[i]] ** mbasis[[j]]) ] == stable[[i,j]]]
 },
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] ;*)

(*Table[
     Row[{
       mbasis[[i]] // TraditionalForm,
       " ",
       mbasis[[j]] // TraditionalForm,
       " = ",
       (mbasis[[i]] ** mbasis[[j]])// TraditionalForm
      }],
     {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] // Grid

     Table[
      Row[{
        mbasis[[i]] // TraditionalForm,
        ".",
        mbasis[[j]] // TraditionalForm,
        " = ",
        (mbasis[[i]]. mbasis[[j]])// TraditionalForm
       }],
      {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] // Grid
```

```
Table[
 Row[{
   mbasis[[i]] // TraditionalForm,
   "^",
   mbasis[[j]] // TraditionalForm,
   " = ",
   (mbasis[[i]]^ mbasis[[j]])// TraditionalForm
  }],
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] // Grid

Table[
 Row[{
   "<",
   mbasis[[i]] // TraditionalForm,
   " ",
   mbasis[[j]] // TraditionalForm,
   ">",
   " = ",
   (⟨mbasis[[i]], mbasis[[j]]⟩)// TraditionalForm
  }],
 {i, 1, mbasis // Length}, {j, 1, mbasis // Length}] // Grid*)
```

Manual tests, showing the results of various products in traditional form.

In[125]:=
```
ClearAll[x, y]

Row[{ "(",
      (#[[1]]) // TraditionalForm,
      ")(",
      (#[[2]]) // TraditionalForm,
      ") = ",
      ((#[[1]]) ** (#[[2]])) // TraditionalForm
     }] &/@ {
    {e2, e2},
    {e2, e21},
    {e2 - 5 e21, e2},
    {e2, e2 + 3 e21} ,
    {e2 + Tan[y] e21, e2},
    {Cos[y] e2, e2 + Sin[x] e21}
  } // Column
```

```
(*XXForm tests (manual verification) *)
Column[(# // TraditionalForm) &/@ {es, e1, e2, e3, e23, e31,
   e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt}]

Column[(# // StandardForm) &/@ {es, e1, e2, e3, e23, e31,
   e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt}]

Column[(# // DisplayForm) &/@ {es, e1, e2, e3, e23, e31,
   e12, e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt}]

Column[(#) &/@ {es, e1, e2, e3, e23, e31, e12,
   e123, msv, msb, mst, mvb, mvt, mbt, msvb, msvt, msbt, mvbt}]
```

Out[126]=
$$(\gamma_2)(\gamma_2) = -1$$
$$(\gamma_2)(-\gamma_{12}) = -\gamma_1$$
$$(5\,\gamma_{12} + \gamma_2)(\gamma_2) = -5\,\gamma_1 - 1$$
$$(\gamma_2)(\gamma_2 - 3\,\gamma_{12}) = -3\,\gamma_1 - 1$$
$$(\gamma_2 - \gamma_{12}\tan(y))(\gamma_2) = \gamma_1\tan(y) - 1$$
$$(\gamma_2\cos(y))(\gamma_2 - \gamma_{12}\sin(x)) = \gamma_1(-\sin(x))\cos(y) - \cos(y)$$

Out[127]=

$$1$$
$$\gamma_1$$
$$\gamma_2$$
$$\gamma_3$$
$$\gamma_{23}$$
$$-\gamma_{13}$$
$$\gamma_{12}$$
$$\gamma_{123}$$
$$\gamma_1 + 1$$
$$\gamma_{23} + 1$$
$$\gamma_{123} + 1$$
$$\gamma_{23} + \gamma_1$$
$$\gamma_{123} + \gamma_1$$
$$\gamma_{123} + \gamma_{23}$$
$$\gamma_{23} + \gamma_1 + 1$$
$$\gamma_{123} + \gamma_1 + 1$$
$$\gamma_{123} + \gamma_{23} + 1$$
$$\gamma_{123} + \gamma_{23} + \gamma_1$$

Out[128]=

$$1$$
$$\gamma[1]$$
$$\gamma[2]$$
$$\gamma[3]$$
$$\gamma[2]\,\gamma[3]$$
$$-\gamma[1]\,\gamma[3]$$
$$\gamma[1]\,\gamma[2]$$
$$\gamma[1]\,\gamma[2]\,\gamma[3]$$
$$1 + \gamma[1]$$
$$1 + \gamma[2]\,\gamma[3]$$
$$1 + \gamma[1]\,\gamma[2]\,\gamma[3]$$
$$\gamma[1] + \gamma[2]\,\gamma[3]$$
$$\gamma[1] + \gamma[1]\,\gamma[2]\,\gamma[3]$$
$$\gamma[2]\,\gamma[3] + \gamma[1]\,\gamma[2]\,\gamma[3]$$
$$1 + \gamma[1] + \gamma[2]\,\gamma[3]$$
$$1 + \gamma[1] + \gamma[1]\,\gamma[2]\,\gamma[3]$$
$$1 + \gamma[2]\,\gamma[3] + \gamma[1]\,\gamma[2]\,\gamma[3]$$
$$\gamma[1] + \gamma[2]\,\gamma[3] + \gamma[1]\,\gamma[2]\,\gamma[3]$$

Out[129]=

1
γ_1
γ_2
γ_3
γ_{23}
$-\gamma_{13}$
γ_{12}
γ_{123}
$1 + \gamma_1$
$1 + \gamma_{23}$
$1 + \gamma_{123}$
$\gamma_1 + \gamma_{23}$
$\gamma_1 + \gamma_{123}$
$\gamma_{123} + \gamma_{23}$
$1 + \gamma_1 + \gamma_{23}$
$1 + \gamma_1 + \gamma_{123}$
$1 + \gamma_{123} + \gamma_{23}$
$\gamma_1 + \gamma_{123} + \gamma_{23}$

Out[130]=

1
γ_1
γ_2
γ_3
γ_{23}
$-\gamma_{13}$
γ_{12}
γ_{123}
$1 + \gamma_1$
$1 + \gamma_{23}$
$1 + \gamma_{123}$
$\gamma_1 + \gamma_{23}$
$\gamma_1 + \gamma_{123}$
$\gamma_{123} + \gamma_{23}$
$1 + \gamma_1 + \gamma_{23}$
$1 + \gamma_1 + \gamma_{123}$
$1 + \gamma_{123} + \gamma_{23}$
$\gamma_1 + \gamma_{123} + \gamma_{23}$

TODO : test multivector products : dot, wedge, **

Manual dot product test (BROKEN).

```
(* manual test, or just the dot product *)
ClearAll[m1, m2]
m1 = (*Scalar[1] + Vector[1,2] ++) Bivector[1, 2, 3]
 (*+ Trivector[1, 1,2,3]*);
m2 = (*2Scalar[1] - Vector[1,2] ++) 3 Bivector[1, 3, 1]
 (*- Trivector[1, 1,2,3]*);

(*m1// TraditionalForm
   m2//TraditionalForm*)
m1
m2
m1.m2

(*ClearAll[bbinaryOperator]
bbinaryOperator[f_,b_?bladeQ,m_grade]:=
  Total[f[b,#]&/@(GradeSelection[m,#]&/@(Range[4+1]-1))]
(*bbinaryOperator[f_,m_grade,b_?bladeQ]:=
     Total[f[#,b]&/@(GradeSelection[m,#]&/@(Range[4+1]-1))]*)
    (*bbinaryOperator[f_,m1_grade,m2_grade]:=
     Total[f[#//First,#//Last]&/@({GradeSelection[m1,#]&/@(Range[4+1]-1),
          GradeSelection[m2,#]&/@(Range[4+1]-1)}//Transpose)]
    *)

    (*Dot;
    handle dot products where one or more factors is a multivector.*)

grade/:grade[g1_,m1_].grade[g2_,m2_]:=
   bbinaryOperator[Dot,grade[g1,m1],grade[g2,m2]];*)

(*m1.m2 *)
(*bbinaryOperator[Dot,m1,m2]*)
(*ClearAll[gs]
  gs =(GradeSelection[m2,#]&/@(Range[4+1]-1));
Dot[b,#]&/@ gs*)

grade[2, {{0, complex[0, -1], 0, 0}, {complex[0, -1], 0, 0, 0},
  {0, 0, 0, complex[0, -1]}, {0, 0, complex[0, -1], 0}}]

grade[2, {{0, -3, 0, 0}, {3, 0, 0, 0}, {0, 0, 0, -3}, {0, 0, 3, 0}}]

0
```

(Manual) tests for grad, div, and curl. BROKEN.

In[131]:=
```
ClearAll[s, v, b, tri, q,
 grads,
 gradv, curlv, divv,
 gradb, curlb, divb,
 gradt, curlt, divt,
 gradq, curlq, divq,
 t, x, y, z, e, f, g, h]


s := Scalar[g[t, x, y, z]];
grads = Grad[s, {t, x, y, z}];

v := Vector[f[t, x, y, z], 1] + Vector[g[t, x, y, z], 2] +
   Vector[h[t, x, y, z], 3] + Vector[e[t, x, y, z], 0];
b := Trivector[1, 1, 2, 3] v ;
tri := Trivector[1, 1, 2, 3] s ;
q := Quadvector[1] s ;

gradv := Grad[v, {t, x, y, z}] ;
divv := Div[v, {t, x, y, z}] ;
curlv := Curl[v, {t, x, y, z}] ;

gradb := Grad[b, {t, x, y, z}] ;
divb := Div[b, {t, x, y, z}] ;
curlb := Curl[b, {t, x, y, z}] ;

gradt := Grad[tri, {t, x, y, z}];
divt := Div[tri, {t, x, y, z}];
curlt := Curl[tri, {t, x, y, z}];

gradq := Grad[q, {t, x, y, z}];
divq := Div[q, {t, x, y, z}];
curlq := Curl[q, {t, x, y, z}];


({# // First, " = ", (# // Last) // DisplayForm} & /@
   {{"s", s},
    {"∇ s", grads},
    {"v", v},
    {"∇ v", gradv},
    {"∇ · v", divv},
```

```
    {"∇ ∧ v", curlv}(*,
    {"b",b},
    {"∇ b",gradb},
    {"∇ · b", divb},
    {"∇ ∧ b",curlb},
    {"T",tri},
    {"∇ T",gradt},
    {"∇ · T",divt},
    {"∇ ∧ T",curlt},
    {"q",q},
    {"∇ q",gradq},
    {"∇ · q",divq},
    {"∇ ∧ q",curlq}*)
  }) // Grid
```

Out[150]=

$$
\begin{aligned}
s &= g[t, x, y, z] \\
\nabla\, s &= \text{Vector}[1, 4] ** g^{(0,0,0,1)}[t, x, y, z] + \\
&\quad \gamma_3\, g^{(0,0,1,0)}[t, x, y, z] + \gamma_2\, g^{(0,1,0,0)}[t, x, y, z] + \gamma_1\, g^{(1,0,0,0)}[t, x, y, z] \\
v &= f[t, x, y, z]\, \gamma_1 + g[t, x, y, z]\, \gamma_2 + e[t, x, y, z]\, \gamma_3 + h[t, x, y, z]\, \gamma_3 \\
\nabla\, v &= \text{Vector}[1, 4] ** \left(\gamma_3\, e^{(0,0,0,1)}[t, x, y, z] + \right. \\
&\quad \left. \gamma_1\, f^{(0,0,0,1)}[t, x, y, z] + \gamma_2\, g^{(0,0,0,1)}[t, x, y, z] + \gamma_3\, h^{(0,0,0,1)}[t, x, y, z]\right) + \\
&\quad \gamma_{30}\, e^{(0,0,1,0)}[t, x, y, z] - h^{(0,0,1,0)}[t, x, y, z] + \gamma_{20}\, e^{(0,1,0,0)}[t, x, y, z] - \\
&\quad g^{(0,1,0,0)}[t, x, y, z] + \gamma_{23}\left(-g^{(0,0,1,0)}[t, x, y, z] + h^{(0,1,0,0)}[t, x, y, z]\right) + \\
&\quad \gamma_{10}\, e^{(1,0,0,0)}[t, x, y, z] - f^{(1,0,0,0)}[t, x, y, z] + \\
&\quad \gamma_{12}\left(-f^{(0,1,0,0)}[t, x, y, z] + g^{(1,0,0,0)}[t, x, y, z]\right) + \\
&\quad \gamma_{13}\left(-f^{(0,0,1,0)}[t, x, y, z] + h^{(1,0,0,0)}[t, x, y, z]\right) \\
\nabla \cdot v &= \text{Vector}[1, 4] ** \left(\gamma_3\, e^{(0,0,0,1)}[t, x, y, z] + \right. \\
&\quad \left. \gamma_1\, f^{(0,0,0,1)}[t, x, y, z] + \gamma_2\, g^{(0,0,0,1)}[t, x, y, z] + \gamma_3\, h^{(0,0,0,1)}[t, x, y, z]\right) - \\
&\quad h^{(0,0,1,0)}[t, x, y, z] - g^{(0,1,0,0)}[t, x, y, z] - f^{(1,0,0,0)}[t, x, y, z] \\
\nabla \wedge v &= \gamma_{30}\, e^{(0,0,1,0)}[t, x, y, z] + \gamma_{20}\, e^{(0,1,0,0)}[t, x, y, z] + \\
&\quad \gamma_{23}\left(-g^{(0,0,1,0)}[t, x, y, z] + h^{(0,1,0,0)}[t, x, y, z]\right) + \\
&\quad \gamma_{10}\, e^{(1,0,0,0)}[t, x, y, z] + \gamma_{12}\left(-f^{(0,1,0,0)}[t, x, y, z] + g^{(1,0,0,0)}[t, x, y, z]\right) + \\
&\quad \gamma_{13}\left(-f^{(0,0,1,0)}[t, x, y, z] + h^{(1,0,0,0)}[t, x, y, z]\right)
\end{aligned}
$$