Reference Manual for V3TOOLS

Thomas Wolf
Department of Mathematics
Brock University
St.Catharines
Ontario, Canada L2S 3A1
twolf@brocku.ca

January 13, 2025

1 Purpose

Procedures of this package perform computationally expensive and non-trivial computations with polynomials of products of 3-component vectors. The individual procedures can be grouped into the following categories:

- initialization of the package,
- generation of scalar vector expressions according to a multiple weighting scheme where each vector has multiple weights,
- conversions of scalar vector expressions between different representations,
- the computation of Poisson brackets between vector expressions,
- a computation determining whether a given scalar vector expression is functionally depending on a list of other scalar vector expressions.

A possible application of these procedures is the classification of integrable 'vectorial' Hamiltonians, the study whether obtained expressions are functionally independent of known expressions and the compactification of results for publication.

These routines were designed and implemented to support the classification of integrable Hamiltonians. Examples are chosen from this application. More details are given in [1].

2 Notation

The chosen conventions aim to display large vector expressions as compact as possible. All vectors have 3 components and are represented by a single letter, like A or B. In this manual we use capital letters in mathematical formulas and lower case letters for REDUCE input and output. REDUCE is not case sensitive but output shows in lower case letters. Vector components have an extra digit 1, 2 or 3, like $A2, B3$. Products of vectors are represented by a single identifier where the following convention is used: $ABCD...$ stands for $(A\ ,\ B \times (C \times (D \times ...)))$ where $(\ ,\ )$ denotes the scalar vector product and $\times$ the skew symmetric vector product. For example, we have $AB = (A, B)$ and $ABC = (A, B \times C)$. In the following we distinguish between three forms of scalar vector expressions:

- **the component form** involving only the components $A1, A2, A3, B1, ...,$

- **the standard vector form** involving only scalar products $AB\,(=(A,B))$ and triple products $ABC\,(=(A, B\times C))$,

- **the extended vector form** involving any product $ABCD....$

Because of the commutativity of the scalar product $(A,B) = (B,A)$ there seem to be two identifiers AB and BA suited to represent the same product. Similarly for triple products we have the relation $ABC = BCA = CAB = -ACB = -CBA = -BAC$. In order to have an unambiguous notation such that vanishing rational expressions simplify to zero we adopt then convention that for scalar products and triple products only identifiers are used in which letters are sorted alphabetically. For example, the program uses *AB,AV,ABV,BUV* but not *BA, VA,BVA, VBU.* In order to simplify manual input one can assign expressions using identifiers, like *BA,BUA*, and afterwards convert them into proper notation using the procedure `e2s` (see below or the beginning of the file `v3tools.tst`).

In order to use non-vectorial parameters, like *ALPHA* or *BETA2* every non-vectorial parameter is preceded by `!&`, e.g. one would input `!&alpha` or `!&beta2`.

3 Initialization

Scalar and triple products of 3-component vectors satisfy identities, for example, the four vectors A, B, U, V satisfy

$$0 = AB\ BUV - ABU\ BV + ABV\ BU - AUV\ BB.$$

Therefore, scalar vector expressions, i.e. polynomials of scalar vector products do usually not have a unique form. They can be re-written using these identities, for example, in order to minimize the number of terms of the polynomial or to bring the expression into a canonical form. Sometimes computations have to be done modulo these identities, more precisely, modulo a Gröbner basis of these identities. The computation of the identities and their Gröbner basis has to be done only once with the procedure `vinit` which initializes the global variables `v_,gbase_,heads_`. The procedure `vinit` takes as argument the list of all vectors that occur in the computation, for example, `vinit({u,v,w,z})`. In the default case of vectors `a,b,u,v` the global variables `v_,gbase_,heads_` are already assigned and `vinit` does not have to be called initially. But if some of the vectors satisfy a special relation, like $AB = 0$ then this relations should be assigned (like `ab:=0;`) and `vinit` should be run afterwards. Here an overview of the three global variables:

`v_` : a list of vectors involved in all the computations. The global variable `v_` is needed in the procedure `poisson_v`. The default value is `v_={a,b,u,v}` .

`gbase_` : a list of polynomials which are a Gröbner basis for all identities between scalar products AB and triple products ABC of any vectors in the list `v_`.

`heads_` : a list of leading terms of the polynomials in `gbase_`.

4 Main Routines

Which routines work only for homogeneous vectorial expressions and which for any vectorial expressions?

vinit : initializes all three global variables `vl_, gbase_, heads_` for a given list of vectors. This procedure is not necessary if the list of vectors is *A,B,U,V*. The procedure is necessary if some of the vectors satisfy extra conditions, like *AB=0*. Example:

```
vinit({a,b,c,d});
```

e2s : converts a vectorial polynomial from extended vector form into standard vector form by replacing products *ABCD...* through scalar and triple products. Example:

```
e2s(buuav);  -->  abv*uu - auv*bu
```

It is also useful to convert from standard vector form into standard vector form if one wants to sort factors in scalar and triple products lexicographically. This is useful for working with this package on expressions that have been generated outside this package and that do not obey the lexicographical ordering of factors. Example:

```
e2s(avb*uu + uva*bu);  -->  - abv*uu + auv*bu
```

v2c : converts an expression from any vector form (extended or standard) into component form. Example:

```
v2c(abu); --> a1*b2*u3-a1*b3*u2-a2*b1*u3+a2*b3*u1+a3*b1*u2-a3*b2*u1
```

c2sl : converts an expression from component form into standard vector form. The resulting vector expression is returned partitioned as a list of sublists. Each sublist contains expressions with the same multiplicity of each vector (the same homogeneity). The first expression of each list is the vector equivalent of the corresponding input terms in component form. All other expressions in each sublist are identically vanishing vector expressions, i.e. vector identities with the same multiplicity as the first expression in each sublist. That means, the second, third,... expression in each sublist multiplied with a constant could be added to the first expression of each sublist in order to change its form but not its value. Example:

```
c2sl(a1*b2*u3 - a1*b3*u2 - a2*b1*u3 + a2*b3*u1 + a3*b1*u2 - a3*b2*u1 +
     a1*b1*b2*u3*v1 - a1*b1*b2*u1*v3 + a1*b1*b3*u1*v2 - a1*b1*b3*u2*v1 -
     a1*b2**2*u2*v3 + a1*b2**2*u3*v2 - a1*b3**2*u2*v3 + a1*b3**2*u3*v2 +
     a2*b1**2*u1*v3 - a2*b1**2*u3*v1 + a2*b1*b2*u2*v3 - a2*b1*b2*u3*v2 +
     a2*b2*b3*u1*v2 - a2*b2*b3*u2*v1 + a2*b3**2*u1*v3 - a2*b3**2*u3*v1 -
     a3*b1**2*u1*v2 + a3*b1**2*u2*v1 + a3*b1*b3*u2*v3 - a3*b1*b3*u3*v2 -
     a3*b2**2*u1*v2 + a3*b2**2*u2*v1 - a3*b2*b3*u1*v3 + a3*b2*b3*u3*v1 );

--> {{abu},{abu*bv - abv*bu, ab*buv - abu*bv + abv*bu - auv*bb}}
```

The input expression is equal to the sum of the first expressions of all sublists, i.e. equal *ABU + ABU BV - ABV BU*. The second expression of the second sublist is an identity, i.e. *0=AB BUV - ABU BV + ABV BU - AUV BB*, where each vector occurs in each term equally often as in each term of the first expression in the second sublist.

c2s : is equivalent to `c2sl` with the difference that all expressions of all sublists are added up with all identity expressions obtaining a coefficient `!&c1, !&c2,....` Example:

```
c2s( same input as for c2sl );
--> abu + abu*bv - abv*bu + !&c1*(ab*buv - abu*bv + abv*bu - auv*bb)
```

s2s : generates and uses vector identities to reduce the length of an expression in standard vector form. Example:

```
s2s( - abu*vv + abv*uv + av*buv);  -->  auv*bv
```

s2e : like s2s but also uses identities involving products $ABCD..$ and thus transforms standard vector form into extended vector form. Example:

```
s2e(abv*uu - auv*bu);  -->  buuav
```

In its current form it only uses products that involve all the vectors involved in each term of the standard vector form. In the above example it would not try to use products with 4 factors but only with 5 factors, like `buuav`.

genpro : generation of all scalar and triple products for a given vector list. Example:

```
genpro({a,b,u,v}); --> {aa,ab,au,av,bb,bu,bv,uu,uv,vv,abu,abv,auv,buv}
```

genpro_wg : similar to `genpro` with the difference that each vector is accompanied by a list of weights and the resulting scalar and triple products also come with a list of weights. Example:

```
genpro_wg({{a,1,0,0},{b,0,1,0},{u,0,0,1},{v,1,0,1}});
--> {{aa,2,0,0},{ab,1,1,0},{au,1,0,1},{av,2,0,1},{bb,0,2,0},
     {bu,0,1,1},{bv,1,1,1},{uu,0,0,2},{uv,1,0,2},{vv,2,0,2},
     {abu,1,1,1},{abv,2,1,1},{auv,2,0,2},{buv,1,1,2}}
```

poisson_c : computes the poisson bracket of two arbitrary expressions. This procedure needs as input the Poisson structure matrix which we call `struc_cons` below. This is encoded as a list of lists, specifying all non-vanishing Poisson brackets between any two dynamical variables, here `u1,u2,u3,v1,v2,v3`. For example, the first sublist `{u1,u2,u3}` of `struc_cons` below, encodes the Poisson bracket relation $\{U1, U2\} = -\{U2, U1\} = U3$. Poisson brackets associated to other Lie-algebras are appended to the file `v3tools.red`. The structure matrix below encodes the Lie-Poisson bracket $e(3)$ for `!&kap=0`, $so(4)$ for `!&kap=1` and $so(3,1)$ for `!&kap=-1`. `poisson_c` needs *all* of its arguments in component form. Example:

```
ham:=v2c(ab*uu-2*au*bu+buv)$
fi :=v2c(bu*(2*auv-!&kap*uu+vv))$
!&kap:=-a1**2-a2**2-a3**2$
struc_cons:={{u1,u2, u3}, {u2,u3, u1}, {u3,u1, u2},
             {u1,v2, v3}, {u2,v3, v1}, {u3,v1, v2},
             {u1,v3,-v2}, {u2,v1,-v3}, {u3,v2,-v1},
             {v1,v2, !&kap*u3}, {v2,v3, !&kap*u1}, {v3,v1, !&kap*u2}}$

poisson_c(ham,fi,struc_cons);  -->  0
```

The example shows that `fi` is a first integral of `ham` under the assumption `!&kap = - aa`. Note that `!&kap` could not have been expressed in terms of components of vectors before assigning `fi` as then the argument to `v2c` would not be a purely vectorial expression.

poisson_v : computes the Poisson bracket for two vector expressions. It uses the global variable `gbase_`. When `poisson_v` is called the first time it computes the Poisson bracket between any scalar and triple product once using the procedure `poisson_c` and stores these elementary Poisson brackets under the global operator `poi_`. Therefore, the third argument to `poisson_v` (the Poisson structure matrix) has to be given in component form. Example: Compare the following with the example for the call of `poisson_c`.

```
ham:=ab*uu-2*au*bu+buv$
fi :=bu*(2*auv-!&kap*uu+vv)$
!&kap:=-a1**2-a2**2-a3**2$
struc_cons:={{u1,u2, u3}, {u2,u3, u1}, {u3,u1, u2},
             {u1,v2, v3}, {u2,v3, v1}, {u3,v1, v2},
             {u1,v3,-v2}, {u2,v1,-v3}, {u3,v2,-v1},
             {v1,v2, !&kap*u3}, {v2,v3, !&kap*u1}, {v3,v1, !&kap*u2}}$
!&kap:=-aa$
poisson_v(ham,fi,struc_cons);  -->  0
```

When calling `poisson_v` a second time the computation proceeds much faster as the Poisson brackets between scalar and triple products had been stored with the operator `poi_`. Note that at first `!&kap` is expressed in component form to have `struc_cons` in component form but afterwards `!&kap` is expressed in vector form to have `ham,fi` in vector form at the time of calling `poisson_v`.

gfi : generates a vector expression with unknown coefficients according to a specific list of weights $\{w_1, w_2, ...\}$. The number of weights is arbitrary but fixed. For example, the vector V could be given weights $\{w_1, w_2, w_3\} = \{1, 0, 1\}$ which will be input as `{v,1,0,1}` in the second argument to `gfi`. In the following example vectors A, B, U, V are each given 3 weights $\{w_1, w_2, w_3\}$ and the most general polynomial is generated where the sum of all $\{w_1, w_2, w_3\}$ values of all vectors in each term are equal $\{2, 1, 3\}$. Example:

```
gfi({2,1,3},
    {{a,1,0,0},{b,0,1,0},{u,0,0,1},{v,1,0,1}},
    {aa, ab, bb, bu, uv, -aa*uu + vv, ab*uu - 2*au*bu + buv},
    heads_
   );
 --> {&r1*abu*uv + &r2*abv*uu + &r3*bv*uv + &r4*auv*bu + &r5*bu*vv
                2
     + &r6*au *bu + &r7*ab*au*uu,
     &r7,&r6,&r5,&r4,&r3,&r2,&r1}
```

The meaning of the parameters:

1. The first parameter of `gfi` is a list of the total weights of each term in the generated polynomial.
2. The second parameter is the list of vectors to be used for generating the expression, each comes with its list of weight values, like `{a,1,0,0}`.
3. The third parameter is a list of homogeneous vector expressions. The polynomial returned by `gfi` must not include any expression that is functionally dependent on the expressions in this list. In other words, from the most general polynomial with proper homogeneity that is generated at first within `gfi` terms are dropped so that it is not possible to choose in the remaining polynomial the undetermined coefficients

`!&r1,!&r2,...` such that an expression results which is functionally dependent only on the expressions of the third parameter list. In this example we want to formulate an ansatz for a first integral that is functionally independent of the expressions `aa, ab, bb` (because A, B are assumed to be constant vectors and U, V to be dynamical vectors), `bu` (a known first integral), `uv, -aa*uu + vv` (two Casimirs, i.e. first integrals) and `(ab*uu - 2*au*bu + buv)` which is the Hamiltonian. The third parameter prevents the generation of the term `bu*aa*uu` which also has weights $\{2, 1, 3\}$ (as `b` has weights $\{0, 1, 0\}$, `u` has weights $\{0, 0, 1\}$ and `a` has weights $\{1, 0, 0\}$). The term `bu*aa*uu` is dropped because the resulting polynomial would otherwise contain the functionally dependent expression `bu*(-aa*uu+vv)`.

4. The fourth parameter is the list of leading terms of the Gröbner basis of identities between the vectors. The resulting polynomial contains only terms which are not a multiple of any one of the terms in this list. By excluding terms which are not multiples of leading terms of identities in the Gröbner basis one defines a canonical form which vanishes only if all terms vanish.

The result of `gfi` is a list. Its first element is the most general polynomial that has the required properties. The list of undetermined coefficients `!&r..` is appended.

fnc_dep : investigates whether a given vectorial expression (the first parameter) is functionally dependent on the elements of a list (the second parameter). The third parameter is the list of occurring vectors with their weights. In the following example it is to be checked whether the first integral `finew` is new or merely the known one `fiold` in disguised form. Example:

```
ham:=ab*uu-1/2*au*bu+buv$
fiold:=bu**2*((bu*aa-ab*au)**2*uu+2*(bu*aa-ab*au)*(bu*auv-buv*au)
       -vv*abu**2-(bu*av-bv*au)**2-uu*vv*ab**2+aa*uu*vv*bb)$
finew:=( - 16*aa**2*bu**4*uu + 96*aa*ab**2*bb*uu**3 -
96*aa*ab*au*bb*bu*uu**2 + 32*aa*ab*au*bu**3*uu + 32*aa*abu*bu**3*uv -
32*aa*abv*bu**3*uu + 24*aa*au**2*bb*bu **2*uu - 16*aa*bu**4*vv +
56*ab**4*uu**3 - 84*ab**3*au*bu*uu**2 + 168*ab**2*abu* bv*uu**2 -
168*ab**2*abv*bu*uu**2 + 26*ab**2*au**2*bu**2*uu + 360*ab**2*auv*bb*
uu**2 + 168*ab**2*bb*uu**2*vv - 184*ab**2*bb*uu*uv**2 -
168*ab**2*bu**2*uu*vv + 336*ab**2*bu*bv*uu*uv - 168*ab**2*bv**2*uu**2
+ 264*ab*abu*au*bb*uu*uv - 32*ab* abu*au*bu**2*uv -
168*ab*abu*au*bu*bv*uu + 192*ab*abu*av*bb*uu**2 - 456*ab*abv*
au*bb*uu**2 + 200*ab*abv*au*bu**2*uu - 7*ab*au**3*bu**3 -
180*ab*au*bb*bu*uu*vv + 44*ab*au*bb*bu*uv**2 + 192*ab*au*bb*bv*uu*uv +
116*ab*au*bu**3*vv - 168*ab*au* bu**2*bv*uv + 84*ab*au*bu*bv**2*uu +
192*ab*av*bb*bu*uu*uv - 192*ab*av*bb*bv*uu **2 -
264*abu*au**2*bb*bv*uu + 42*abu*au**2*bu**2*bv - 96*abu*au*av*bb*bu*uu
+ 96*abu*bb*bu*uv*vv - 136*abu*bb*bv*uu*vv + 24*abu*bb*bv*uv**2 -
56*abu*bu**2*bv* vv + 112*abu*bu*bv**2*uv - 56*abu*bv**3*uu +
360*abv*au**2*bb*bu*uu - 42*abv*au **2*bu**3 + 40*abv*bb*bu*uu*vv -
24*abv*bb*bu*uv**2 + 56*abv*bu**3*vv - 112*abv* bu**2*bv*uv +
56*abv*bu*bv**2*uu - 264*au**2*auv*bb**2*uu + 42*au**2*auv*bb*bu** 2 +
96*au**2*bb**2*uu*vv - 40*au**2*bb*bu**2*vv - 96*au**2*bb*bv**2*uu +
16*au** 2*bu**2*bv**2 - 192*au*av*bb**2*uu*uv + 192*au*av*bb*bu*bv*uu
- 32*au*av*bu**3* bv - 136*auv*bb**2*uu*vv + 24*auv*bb**2*uv**2 +
40*auv*bb*bu**2*vv + 112*auv*bb* bu*bv*uv - 56*auv*bb*bv**2*uu +
96*av**2*bb**2*uu**2 - 96*av**2*bb*bu**2*uu + 16 *av**2*bu**4 -
96*bb**2*uu*vv**2 + 96*bb**2*uv**2*vv + 96*bb*bu**2*vv**2 - 192*
bb*bu*bv*uv*vv + 96*bb*bv**2*uu*vv)/8$

if fnc_dep(finew,{aa,ab,bb,bu,uv,uu*aa-vv,ham,fiold},
          {{A,1,0,0},{B,0,1,0},{U,0,0,1},{V,1,0,1}}) then
write"This is a known first integral."                          else
write"This is a new first integral!"$

 -->

The expression in question is functionally dependent on
the list of expressions {p_1,p_2,...} in the following way:
              2                3                  2
10*p_2*p_3*p_5 *p_7 + 7*p_2*p_7  + 12*p_3*p_6*p_7  - 2*p_8

This is a known first integral.
```

5 Complete list of global variables

Type	name	Purpose
algebraic	v_, gbase_, heads_	needed for `poisson_v()`, `gfi()`, `fnc_dep()`
	!&c1, !&c2, ...	constant coefficients introduced by `c2s()`
symbolic	fino_, wgths_	used internally in `gen()`
	!&r1, !&r2, ...	constant coefficients introduced by `gfi()`
	tr_vec	global tracing flag
operator	poi_	stores Poisson brackets between scalar and triple products

6 Requirements

```
LOAD_PACKAGE crack, v3tools$
```

References

[1] V.V.Sokolov, T.Wolf, Integrable quadratic Hamiltonians on $so(4)$ and $so(3,1)$, preprint nlin.SI/0405066 on arXiv.