JGSP **43** (2017) 73–105

JOURNAL OF
Geometry and Symmetry in Physics
ISSN 1312-5192

CLIFFORD ALGEBRA IMPLEMENTATIONS IN MAXIMA

DIMITER PRODANOV

Communicated by Ivaïlo M. Mladenov

Abstract. This tutorial focuses on the packages `clifford` and `cliffordan` for the computer algebra system *Maxima*. *Maxima* is the open source descendant of the first computer algebra system and features a rich functionality from a large number of shared packages. The Maxima language is based on the ideas of functional programming, which is particularly well suited for transformations of formal mathematical expressions. While `clifford` implements Clifford algebras $C\ell_{p,q,r}$ of arbitrary signatures and order based on the elementary construction of Macdonald, `cliffordan` features geometric calculus functionality. Using `clifford` expressions containing geometric, outer and inner products can be simplified. Applications of `clifford` and `cliffordan` in linear algebra and calculus are demonstrated.

MSC: 08A70, 11E88, 15A69, 15A75, 94B27
Keywords: Clifford product, computer algebra, Dirac operator, electromagnetism, geometric product, multilinear algebra, outer product, vector derivative

Contents

1. Introduction

Proponents of Geometric Algebra and Geometric Calculus promote the view that these approaches unify, simplify, and generalize vast areas of mathematics that involve geometric ideas. Clifford algebras provide natural generalizations of complex, dual and double numbers into the concept of *Clifford numbers*. It is then natural to use existing *Computer Algebra Systems* (CAS) to implement various geometric algebra instances and provide them as tools for science and engineering. Several Clifford and Geometric algebra packages have been developed for different types of CAS and engineering numerical suites. For Maple since the late 1990's, Ablamowicz and Fauser develop the package `CLIFFORD` [1]. Computations in `CLIFFORD` are based on Chevalley's definition of Clifford algebra as a sub-algebra of the algebra of endomorphisms of the Grassmann algebra. Another package, `BIGEBRA` [2], builds on Maple's `CLIFFORD`, with the aim to explore Hopf algebras and provide a useful tool for experimental mathematics. Finally, the package called `eCLIFFORD` computes the Clifford product in $C\ell_{p,q,r}$ using Walsh functions. For Mathematica, the `clifford.m` package [4] introduces Clifford and Grassmann algebras, multivectors, and the geometric product. Although not in symbolic computer algebra, the recent work of Sangwine and Hitzer [16] is also noteworthy. The package is the first comprehensive work for Matlab. It features operations with matrices of Clifford multivectors, including LU decomposition.

For the Python programming language there is an implementation of Clifford algebra developed by Alan Bromborsky[1]. The Python module `galgebra` supports coordinate free calculations using the basic products operation: geometric, outer,

[1]The package is available from GitHub repository `https://github.com/brombo/galgebra`

and inner. The operations can be defined using a completely arbitrary metric defined by the inner products of a set of arbitrary vectors or the metric can be restricted to enforce orthogonality and signature constraints on the set of vectors. In addition the module includes the geometric, outer and inner derivatives and the ability to define a curvilinear coordinate system. The module requires the numPy and the symPy modules.

For Maxima, `atensor`, since 2004, and the `clifford`-based packages, since 2015, are designed to be a symbolic computational tools for applied mathematicians and physicists. The package `atensor` partially implements generalized (tensor) algebras. The packages `clifford` and `cliffordan` authored by the presenter, implement Clifford algebras $C\ell_{p,q,r}$ of arbitrary signatures and orders. `clifford` emphasizes simplification, including the ability to treat multivectors as "sparse" purely symbolical objects [15].

There is an elementary construction of Clifford algebras given by Macdonald [9]. This construction of $\mathbb{G}^n$ is suitable for direct implementation in a computer algebra system supporting symbolical transformations of expressions. From design perspective it was the preferred choice in `clifford`, while `atensor` implements the inner-product quadratic form- based approach. Before proceeding further with the construction, we give some remarks for computer algebra systems, which are given little attention by pure mathematicians but are rather important in computer science.

1.1. Expression Representation and Transformation in Maxima

Maxima is the open source descendant of the first ever computer algebra system and features a rich functionality from a large number of shared packages. While written in Lisp, *Maxima* has its own programming language. The system also offers the possibility of running batch unit tests. Maxima supports several primitive data types [10]: *numbers* (rational, float and arbitrary precision); *strings* and *symbols*. In addition there are compound data types, such as *lists*, *arrays*, *matrices* and *structs*. There are also special symbolic constants, such as the Boolean constants `true` and `false` or the complex imaginary unit `%i`.

Several types of operators can be defined in Maxima CAS. An operator is a defined symbol that may be *unary* `prefix`, *unary* `postfix`, *binary* `infix`, *n-ary* `matchfix`, or `nofix` types. For example, the inner and outer products defined in `clifford` are of the `binary infix` type. The scripting language allows for defining new operators with specified precedence or redefining the precedence of existing operators.

The relation "=" is considered a symmetric equivalence relation, while the syntactic equality "≅" pertains only to the symbolical structure of the expression. In the computer algebra systems, there is a clear distinction between these two relations with the symbol "=" corresponding only to syntactic equality. However, here I will comply to usual mathematical notational convention and use "=" in both ways. In such way, expressions (or their parts) can be compared syntactically and matched against existing transformation rules, without the need of evaluating them first. In addition, there is an another important concept. Maxima distinguishes between two forms of applications of operators – forms which are `nouns` and forms which are `verbs`. The difference is that the `verb` form of an operator evaluates its arguments and produces an output result, while the `noun` form appears as an inert composite symbol in an expression, without being executed. A `verb` form can be mutated into a `noun` form and vice-versa. This allows for context-dependent evaluation, which is especially suited for symbolic processing.

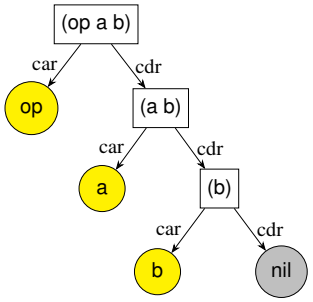


Figure 1. Expression representation in Lisp. A general expression can be represented as a list with the first element being the operator `op` and the rest of the elements representing operator arguments. For instance, the expression `op(a,b)` will be represented by the list `(op a b)`, which is the ordered pair of the atom `op`, and another list, `(a b)`, which, in turn, is represented by another ordered pair.

The Maxima language is based on the ideas of functional programming, which is particularly well suited for transformations of formal mathematical expressions. Maxima programs can be automatically translated and compiled to Lisp within the program environment itself. Third-party Lisp programs can be also loaded and accessed from within the system. The manner in which Maxima represents expressions, function calls and index expressions using the Lisp language is particularly relevant for the design of the `clifford`-based packages. In the underlying Lisp representation a Maxima *expression* is a tree containing sequences of operators, numbers and symbols. Every Maxima expression is simultaneously also a

λ-construct and its value is the value of the last assigned member. This is a design feature inherited from Lisp. Maxima expressions are represented by underlying Lisp constructs. The core concept of the Lisp language is the idea of a *list* representation of the language constructs. The list is represented recursively by an ordered pair, the first element of which is the head (or `car`), the second element the tail (or `cdr`) of the list, which is also a list (see Fig. 1). List elements are themselves either *lists* or *atoms*: e.g., a number, a symbol, or the empty list (`nil`). This representation enables the possibility to define transformation rules. In such way a part of an expression can be matched against a pattern and rewritten (see Listing 1).

A very powerful feature of the system is the ability to define custom transformation rules. Various transformation rules can be associated with any given operator in Maxima. Maxima has an advanced pattern matching mechanism, which supports nesting of operators and simplification. User-defined rules can be added to the built-in simplifier using one of two commands: `tellsimp` or `tellsimpafter`. Rules in both sets are identified by the main operator of the expression. Rules specified using `tellsimp` are applied before the built-in simplification, while `tellsimpafter` rules are applied after the built-in simplification. The augmented simplification is then treated as built-in, so subsequent tellsimp rules are applied before those defined previously. An example is given in Listing 1 used in the implementation of `clifford`.

2. Construction of the Algebra $C\ell_n$

Let us give a brief exposition on the properties of Clifford algebras over the reals following [9]. Any Clifford algebra $\mathbb{G}^n$ is an associative unitary algebra, which is generated by a vector space V spanned by the orthonormal basis $\{e_1 \dots e_n\}$, over the field $\mathbb{K}$ of characteristic different from two. The unit is usually skipped in notation (assuming implicit conversion between the scalar unit of $\mathbb{K}$ and the vector unit of V wherever necessary) and the square of the vector v is denoted conveniently by v^2.

Definition 1 (Indicial map). *Define the indicial map $\imath$ acting on symbols by concatenation*

$$\imath_e : \ g \mapsto e_g$$

Assert the convention $\imath_e : \ \emptyset \mapsto 1$.

Definition 2 (Clifford algebra).

The construction of the algebra goes in the following steps:

1. *Fix a generator symbol* **e** *and adjoin an index* $k \leq n \in \mathbb{N}$ *to the generator symbol as the action of the indicial map* $\mathfrak{l}_e : i \mapsto e_k$ *producing a set of n* ***generators*** *(i.e., orths) of the algebra* $E := \{e_1 \dots e_n\}$.

2. *Fix an order* $\prec$ *over* E.

3. *Define a vector space* $V(E,\mathbb{K})$ *over the set of generators serving as basis with axioms*

   - *Commutativity of vector addition*: $a+b=b+a$.
   - *Associativity of vector addition*: $(a+b)+c=a+(b+c)$.
   - *Existence of additive unity* 0: $0+a=a+0=a$.
   - *For every vector, there exists an additive inverse*: $a+(-a)=0$.
   - *Associativity of scalar multiplication*: $\lambda(\mu a)=(\lambda\mu)a$.
   - *Compatibility with scalar multiplication*: $a\lambda=\lambda a$.
   - *Distributivity of scalar addition*: $(\lambda+\mu)a=\lambda a+\mu a$.
   - *Distributivity of vector addition*: $\lambda(a+b)=\lambda a+\lambda a$.
   - *Scalar multiplication identity*: $1_{\mathbb{K}}\, a=a$.

   for vectors a,b,c *and scalars* λ,μ.

4. *Adjoin an associative algebra* $\mathbb{G}(E,\mathbb{K})$ *over* $V(E,\mathbb{K})$ *using the* Clifford product *or* geometric multiplication *operation with axioms*

   - *Existence of an algebra unity*: $1_{C\ell}a=a1_{C\ell}=a$ *for a non-scalar* a.
   - *Left distributivity*: $a(b+c)=ab+ac$ *for arbitrary elements* a,b,c.
   - *Right distributivity*: $(a+b)c=ab+ac$ *for arbitrary elements* a,b,c.
   - *Associativity*: $a(bc)=(ab)c$ *for arbitrary elements* a,b,c.
   - *Compatibility with scalars*: $(\lambda a)(\mu b)=(\lambda\mu)(ab)$ *for scalars* λ,μ *and non-scalar elements* a,b.

5. *Finally, assert*

   Closure Axiom *For k orths* $e_1,\dots,e_k \in C\ell_n$ *and scalars* λ, μ *the multivector belongs to the algebra*

   $$\lambda+\mu e_1 \dots e_k \in \mathbb{G}^n. \tag{C}$$

Reduction Axiom *For all orths*

$$e_k e_k = \sigma_k 1_{C\ell}, \qquad \sigma_k \in \{1,-1,0\} \tag{R}$$

where σ_k are scalars of the field $\mathbb{K}$.

Anti-Commutativity Axiom *For every two basis vectors, such that $e_i \prec e_j$*

$$e_i e_j = -e_j e_i. \tag{A-C}$$

The notation $C\ell_{p,q,r}(\mathbb{K})$ $(p+q+r=n)$ is interpreted as the convention that p elements of the orthonormal basis square to 1, q elements square to -1 and r (degenerate) elements square to 0. The structure $C\ell_{p,q,r} = \{E, \prec, \mathbb{K}\}$ is called Geometric algebra over $\mathbb{K}$ with operations addition and geometric multiplication.

Further-on generators will be indexed by Latin letters. It should be remarked that the closure and compatibility Axioms are not included in the former construction. This construction can be carried out without modifications in Computer Algebra systems like *Maxima* by defining proper simplification rules for the geometric product operation [13, 15]. The rules defining the construction are presented in Listing 1.

Listing 1. Clifford algebra construction in `clifford`.

```
/*
Abstract Cliford algebra construction
*/
matchdeclare([aa, ee], lambda([u], not freeof(asymbol,u) and
    freeof (``+'', u) and not scalarp(u) ), [bb,cc], true,
[kk, mm, nn], lambda([z],  integerp(z) and z>0) );

if get('clifford,'version)=false then (
       tellsimp(aa[kk].aa[kk], signature[kk] ),
       tellsimpafter(aa[kk].aa[mm], dotsimp2(aa[kk].aa[mm])),
       tellsimpafter(bb.ee.cc, dotsimpc(bb.ee.cc)),
       tellsimp(bb^nn, bb^^nn)
);
```

Here the list `signature` corresponds to the set $\{\sigma\}$. While the reordering of products is executed by the function `dotsimp2`.

3. Consistency of the Extension

Here we present results demonstrating the consistency of the Clifford algebra definition.

Proposition 3. *The scalar and algebra units coincide*

$$1_{\mathbb{K}} = 1_{C\ell} \equiv 1.$$

Proof: Suppose that $e_0 \in V(E)$ is the unit of the algebra. Then by the anti-commutativity: $e_0 e_k + e_k e_0 = 0 \leftrightarrow e_k + e_k = 0$ which is a contradiction. Therefore, $e_0 \in \mathbb{K} \Rightarrow e_0 = 1$ by the uniqueness property. ■

Theorem 4. *$V(E)$ extends over the power set of E*

$$P(E) := \{1, e_1, \ldots, e_n, e_1e_2, e_1e_3, \ldots, e_1e_2 \ldots e_n\}.$$

That is $V(P(E))$ is a vector space and we have the inclusion

$$V(E) \subset V(P(E)).$$

Proof: The proof follows from the Closure Axiom.
Multiplicative properties:

1. Scalar commutativity $\mu e_1 \ldots e_k = e_1 \mu \ldots e_k = \ldots = e_1 \ldots e_k \mu$.
2. In particular $1 e_1 \ldots e_k = e_1 \ldots e_k 1 = e_1 \ldots e_k$.
3. $0 e_1 \ldots e_k = 0 e_2 \ldots e_k = \ldots = 0 e_k = 0$.
4. Scalar compatibility: $\lambda(\mu e_1 \ldots e_k) = \lambda(\mu e_1)(e_2 \ldots e_k) = (\lambda\mu) e_1 \ldots e_k$.

And we extend by linearity and the closure over the whole algebra.
Additive properties:

1. Universality of zero: Let $L = 0 + e_1 \ldots e_k$. Right-multiply by e_k. If $\sigma_k = 0$ then the results follows trivially.

   So let $\sigma_k \neq 0$. Then $L e_k = 0 e_k + e_1 \ldots e_{k-1} \sigma_k = \sigma_k e_1 \ldots e_{k-1}$. Right-multiply by e_k. Then $\sigma_k L = \sigma_k e_1 \ldots e_k \Leftrightarrow L = e_1 \ldots e_k$.
2. Scalar distributivity: $(c+d) e_1 \ldots e_k = ((c+d) e_1) e_2 \ldots e_k = (c e_1 + d e_1) e_2 \ldots e_k$ $= c e_1 \ldots e_k + d e_1 \ldots e_k$.

3. In particular, there is an additive inverse element $e_1 \ldots e_k - e_1 \ldots e_k = 0$.

4. Commutativity of addition: Let $M = e_a + e_b - e_b - e_a$. Then for all possible orders of evaluation $M = 0$. Therefore, by linearity $a + b = b + a$.

5. Associativity of addition: Let $M = (e_a + e_b) + e_c - e_a - (e_b + e_c)$. By commutativity of addition and closure $M = (e_a + e_b)1 - (e_b + e_c)1 - e_c - e_a$. Then for all possible orders of evaluation of the first two summands $M = e_a - e_c + e_c - e_a = 0$. Therefore, by linearity $(a + b) + c = a + (b + c)$.

6. Distributivity of addition follows directly from the distributivity axioms and the closure axioms
$$\lambda(e_a + e_b) = \lambda e_a + \lambda e_b.$$

And we extend by linearity and the closure over the entire algebra. ■

Based on this extension principle we can identify the index set isomorphism

$$j : e_a e_b \ldots e_m \mapsto e_J, \qquad J = \{a, b, \ldots, m\} \tag{1}$$

and use multi-index notation implicitly wherever appropriate. For consistency we also extend the ordering to the power set $\prec_{P(E)}$. However we will not use distinct symbol for this. Multi-indices will be denoted with capital letters.

Definition 5. *The **extended basis** set of the algebra will be defined as the sorted power set*

$$\mathbf{E} := \{P(E), \prec\}.$$

Therefore, we can afford the following multivector definition.

Definition 6. *A multivector of the Clifford algebra is a linear combination of elements over the 2^n-dimensional vector space spanned by* **E***.*

Lemma 7 (Well posedness). *The Clifford algebra construction is well defined.*

Proof: $V(E)$ is well defined. Moreover, the vector space extension is also well defined. The compatibility Axioms imply the inclusion $V(E, \mathbb{K}) \subset \mathbb{G}(E, \mathbb{K})$.

Therefore, we only need to check the additional Algebra axioms. The Reduction Axiom implies the Closure Axiom trivially. The Anti-commutativity Axiom implies the Closure Axiom trivially. Further, by restriction to ordered pairs the Anti-commutativity Axiom does not apply when the Reduction Axiom applies. Therefore, these two axioms are logically independent. Therefore, restricted algebra construction over $V(E)$ is well defined.

The algebra extension is well-defined since the existence of $C\ell_n$ implies the existence of $C\ell_{n-1}$ by the Closure Axiom. By reduction, for $n = 1$ the three possible cases for the sign of the element represent the double, complex or dual numbers respectively. Therefore, by induction $C\ell_n$ exists. ■

Definition 8 (Canonical algebra). *Define the canonical ordering as the nested lexicographical order ρ, such that $i < j \Longrightarrow e_i \prec e_j$ and extend it over $P(E)$ as*

$$e_1 \prec e_2 \prec \underbrace{e_1 e_2}_{e_{12}} \prec \ldots \prec \underbrace{e_1 \ldots e_n}_{e_N}.$$

In addition assume that the first p elements square to 1, then next q elements square to -1 and the last r elements square to 0. Then the algebra $C\ell_{p,q,r} \equiv \{E, \rho, \mathbb{K}\}$ is the canonical Clifford algebra.

Using this definition it is very easy to demonstrate the equivalence between different algebras.

Theorem 9 (Algebra equivalence). *Two algebras with the same numbers of p, q and r orths are order-isomorphic.*

Proof: If the elements are ordered identically then the statement is trivial. Let's assume that the elements are ordered differently. But then there is a permutation putting them into canonical order. Therefore, the algebras are isomorphic. Therefore, we can identify an order $\prec'$ with the second permutation. ■

Corollary 10 (S-Law of inertia). *For two isomorphic algebras $C\ell_1$ and $C\ell_2$ ($C\ell_1 \cong C\ell_2$) there is an invertible map*

$$\imath : S_1 \mapsto S_2.$$

Conversely, if there is an invertible map, such that

$$\imath : S_1 \mapsto S_2$$

then $C\ell_1 \cong C\ell_2$.

Proof: The forward statement follows from Theorem 9. Converse case: If the dimensions of S_1 and S_2 are equal and the numbers of p, q and r orths are equal then there is a permutation

$$\imath : S_1 \mapsto S_2.$$

However we note that this is the same permutation which maps

$$\iota : \mathbf{E}_1 \mapsto \mathbf{E}_2$$

and hence by Theorem 9 $C\ell_1 \cong C\ell_2$. ■

Corollary 11. *The canonical algebra $C\ell_{p,q,r}$ defines an equivalence class.*

Definition 12 (Quadratic form of the algebra). *We can define the quadratic form of the algebra as the scalar part of the square of an element $v \in C\ell_n$*

$$Q(v) := \langle vv \rangle_0 : C\ell_n \mapsto \mathbb{K}.$$

In such way we can benefit from the statement of the universal property of the Clifford algebra [11].

3.1. Indicial Representation

Definition 13. *For the sets A and B we define the symmetric difference as the operation*

$$A \setminus B := \{x \, ; \, x \notin A \cap B\}.$$

Definition 14. *Define the argument map* arg *acting on symbol compositions as*

$$\arg : \; f(g) \equiv f \circ g \mapsto g.$$

Assert $\arg f = \emptyset$.

Theorem 15 (Inidical representation). *For generators $e_s, e_t \in C\ell_{p,q,r}$ the following diagram commutes*

$$\begin{array}{ccc} e_s & \underset{\iota_e}{\overset{\arg}{\rightleftarrows}} & \{s\} \\ \downarrow{\scriptstyle e_t} & & \downarrow{\scriptstyle \setminus\{t\}} \\ e_s e_t \equiv e_{st} & \underset{\iota_e}{\overset{\arg}{\rightleftarrows}} & \{s,t\} \end{array}$$

Proof: The right-left ι action follows from the construction of $C\ell_{p,q,r}$. The left-right argument action is trivial. We observe that $\arg f = \emptyset$. Trivially, $\{s\} \setminus \emptyset = \emptyset \setminus \{s\} = \{s\}$. Let's suppose that $s = t$. We notice that $\{s\} \setminus \{t\} = \{t\} \setminus \{s\} = \emptyset$. Let us suppose that $s \neq t$. We notice that $\{s\} \setminus \{t\} = \{s,t\}$ and $\{t\} \setminus \{s\} = \{t,s\}$. ■

4. Simplification of Products in Monomials

The axioms define a canonical simplified representation of a multivector. The main lemma is demonstrated in [9] and is repeated here for convenience

Lemma 16 (Permutation equivalence). *Let $B = e_{k_1} \dots e_{k_i}$ be an arbitrary Clifford multinomial, where the i generators are not necessarily different. Then*

$$B = s P_\rho \{e_{k_1} \dots e_{k_i}\}$$

where $s = \pm 1$ is the sign of permutation of B and $P_\rho \{e_{k_1} \dots e_{k_i}\} \mapsto e_{k_\alpha} \dots e_{k_\omega}$ is the product permutation according to the canonical ordering.

Proof: The proof follows directly from the anti-commutativity of Clifford multiplication for any two generator elements A-C, observing that the sign of a permutation of S can be defined from its decomposition into the product of transpositions as $\operatorname{sgn}(B) = (-1)^m$, where m is the number of transpositions in the decomposition. ■

The corresponding Maxima implementation is given in Listing 2.

Further we can define a *simplified form* or *blade form* according to the action of the Reduction Axiom.

Theorem 17 (Simplified monomial form). *Let $e_{k_1} \dots e_{k_i}$ be an arbitrary Clifford multinomial, where the i generators in $C\ell_{p,q,r}$ are not necessarily different. Let $\setminus$ denote the symmetric set difference operation and s_P is the sign of the permutation $P_\rho \{e_{k_1} \dots e_{k_i}\}$. We say that simplification induces an equivalence relation*

$$\Xi : E \times \dots \times E \mapsto C\ell_{p,q}$$

such that

$$\begin{aligned} e_{k_1} \dots e_{k_i} &= s_P \, \Xi \left(e_{k_\alpha} \dots e_{k_\omega}\right) = s_P \, e_{k_\alpha} \Xi \left(e_{k_\beta} \dots e_{k_\omega}\right) \\ &= s_P \, \Xi \left(e_{k_\alpha} e_{k_\beta}\right) \Xi \left(e_{k_\gamma} \dots e_{k_\omega}\right). \end{aligned}$$

Let $2p \leq i$ generators square as $e_j^2 = 1$. Then its simplified form is

$$e_{k_1} \dots e_{k_i} = s_P \, e_M, \qquad M = \{k_1 \setminus k_2 \setminus \dots k_{i-2p}\}$$

with all k-indices different. Let $2q \leq i$ generators square as $e_j^2 = -1$. Then its simplified form is

$$e_{k_1} \dots e_{k_i} = (-1)^q s_P \, e_M, \qquad M = \left\{k_1 \setminus k_2 \setminus \dots k_{i-2q}\right\}$$

with all k-indices different. Let at lest 2 *generators square as* $e_j^2 = 0$. *Then its simplified form is*

$$e_{k_1} \dots e_{k_i} = 0.$$

Proof: The product is transformed according to Lemma 16. Then we use Theorem 15 to compute the indices. By equation (R) the elements of equal indices are removed from the final list. This induces a factor of $(-1)^q$ in the final result. Hence, for $2q$ elements of index a $\{k_a\} \setminus \{k_a\} = \emptyset$. Further $\emptyset \setminus \{k_i\} = \{k_i\} \setminus \emptyset = \{k_i\}$. The parentheses can be skipped from the final notation to improve readability and the result follows. The associativity of Ξ follows from the associativity of the Clifford product. ■

Listing 2. Clifford product simplification in `clifford`.

```
dotsimpc(ab):=block([ba, c:1, v, w:1, q, r, l, sop],
    ....
    sop:inop(ab),
    if mapatom(ab) or freeof(``.'', ab) or
        sop='nil or sop=``^'' or
            sop=``^^'' then return(ab),
    if sop=``+'' then
        map(dotsimpc, ab)
    else if sop=``*'' then (
        [r,l]: oppart(ab, lambda([u], freeof(``.'', u))),
        if _debug=true then display(sop, r, l),
        r:subst(nil=1, r),
        l:subst(``.'',``*'',l),
        r*dotsimpc(l)
    ) else (
        ba:copy(ab),
        v:inargs(ba),
        if _debug=true then display(sop, v),
        w:sublist( v, lambda ([z], not freeof(asymbol,z) and
            mapatom(z))),
        w:permsign(w),
        if w#0 then (
            v:sort(v),
            for q in v do c:c.q,
            if _debug=true then  display(w, v),
            w*c
        ) else ab
    )
);
```

The constitutive equations define a canonical representation of a multivector expression, which allows for automatic simplification.

Defining simplified forms allows for an implementation of an efficient Clifford product simplification algorithm given in Listing 2.

From this discussion it is evident that in principle the Clifford product simplification can be executed in $\mathbb{O}(N\log(N))$ time since the parity of permutation can be computed along with the sorting step, for example using the merge-sort algorithm. This is a substantial speed-up compared to calculations using matrix representations, which run in exponential time as $\mathbb{O}(4^{\dim})$.

5. Properties of Clifford Algebras in View of Maxima Implementation

Further we can define a *simplified form* according to the action of equation (R). By means of this distinction it is convenient to define *blade* objects.

Definition 18. *A blade of grade k is a product of k basis elements in simplified form.*

Conventionally, 0-blades are scalars, 1-blades are vectors etc. A general multivector M can be decomposed by the grade projection operators $\langle\ \rangle_k$ into a direct sum of different sub-spaces

$$M = \sum_{k=0}^{n} \langle M\rangle_k . \tag{2}$$

Proposition 19 (Maximal element). *The algebra $C\ell_{p,q,r}$ has a maximal element $I = e_1 \dots e_n$ called pseudoscalar.*

Proof: Suppose that $r > 0$. Then if a product contains more than one nilpotent generators of the same index the simplified form is 0 by Lemma 16. If a product contains exactly one nilpotent generator per index the maximal element is the product of all generators by Theorem 17

$$I = e_1 \dots e_n, \qquad n = p+q+r.$$

Suppose that $r = 0$. Then by Theorem 17 the maximal element is the product of all generators

$$I = e_1 \dots e_n, \qquad n = p+q.$$

This element is referred to as the *pseudoscalar* of the algebra. ■

5.1. Main Involutions

There are three important involutions which change signs of blades – the reflection $\hat{A}$, the order reversion $A^{\sim}$ and the Clifford conjugation $A^{\dagger}$. The Clifford conjugation is the composition of reversion and reflection. The sign mutations for the different involutions are shown in Table 1.

Table 1. Sign mutation table for Clifford algebras $C\ell_{p,q,r}$.

	k mod 4			
	0	1	2	3
$\hat{A}$	+	−	+	−
$A^{\sim}$	+	+	−	−
$A^{\dagger} = \hat{A}^{\sim}$	+	−	−	+

Definition 20. *Define the blade index map for $e_a, e_b \in C\ell_{p,q,r}$ with action*

$$j : \lambda e_a e_b \mapsto \{a,b\}.$$

Definition 21. *Define the algebraical dual element of e_s as the result of the duality operation*

$$\star e_s := e_{s \setminus I}.$$

Using this definition we have trivially that the duality transformation is an automorphsim

$$\star(\star e_s) = e_s.$$

Therefore, this operation can identified with the orthogonal complement. The Clifford algebra dual is related to the so-defined algebraic dual by the formula

$$e_s^* = e_s\, I^{-1} = \sigma_I\, s_{P(U)} \star e_s, \qquad U = j(e_s) \cup j(I). \tag{3}$$

5.2. Scalar, Inner and Outer Products

The Clifford product of **vectors** decomposes into a sum of *inner* and *outer* products according to

$$a\,b = a \cdot b + a \wedge b = \langle ab \rangle_0 + \langle ab \rangle_2 . \tag{4}$$

Hestenes further identifies the outer product of vectors with the Grassmann's outer product [7, Ch. 1] and defines the outer product by extension for blades (not for scalars!) as

$$a \wedge B := \frac{1}{2}\left(aB + (-1)^r Ba\right) = \frac{1}{2}\left(aB + \hat{B}a\right) \tag{5}$$

$$a \cdot B := \frac{1}{2}\left(aB - (-1)^r Ba\right) = \frac{1}{2}\left(aB - \hat{B}a\right) \tag{6}$$

where r is the grade of B and a is a vector. In the general case for a grade k blade A_k and grade l blade A_l

$$A_k \wedge B_l = \begin{cases} \langle A_k B_l \rangle_{k+l}, & k+l \leq n \\ 0, & k+l > n. \end{cases}$$

The outer product can be defined also in a different way without demanding grade decomposition. Tn this definition the type of the arguments is important.

Definition 22. *For a scalar α and a blade or scalar B*

$$\alpha \wedge B \mapsto \alpha B.$$

For blades A, B

$$A \wedge B := \begin{cases} AB - \langle AB \rangle_0, & j(A) \cap j(B) = \emptyset \\ 0, & j(A) \cap j(B) \neq \emptyset. \end{cases}$$

Then the product is extended to the entire algebra by linearity of its arguments.

The inner product, on the other hand, can be extended to blades in several different ways depending on the sign of $k-l$ resulting in *left*, *right* or *symmetric contractions*. For the latter we have

$$A_k \rfloor B_l := \begin{cases} \langle A_k B_l \rangle_{k-l}, & k \leq l \\ 0, & k > l \end{cases} \tag{7}$$

$$A_k \lfloor B_l := \begin{cases} \langle A_k B_l \rangle_{k-l}, & k \geq l \\ 0, & k < l \end{cases} \tag{8}$$

$$A_k \cdot B_l := \langle A_k B_l \rangle_{|k-l|}. \tag{9}$$

In addition, the *scalar product* is defined as

$$A_k * B_l := \langle A_k B_l \rangle_0$$

giving rise to the symmetric contraction decomposition [6]

$$A_k \cdot B_l = A_k \rfloor B_l + A_k \lfloor B_l - A_k * B_l.$$

Using these definitions it can be verified that for a scalar α and a vector v the decomposition does not hold

$$\alpha \cdot v + \alpha \wedge v = 2\alpha v.$$

Since such a decomposition for low-grade objects is useful in applications Hestenes demands for example that

$$\alpha \cdot v = 0, \qquad \alpha \cdot \alpha = 0.$$

Another choice is to abandon the complete identification of the wedge product with the Grassmann's outer product and assign

$$\alpha \wedge v = 0, \qquad \alpha \wedge \alpha = 0.$$

All of these choices can be easily implemented in `clifford` using global variable switches.

Contractions can be defined as well by duality without resorting to grade decomposition. For example in $C\ell_{p,0,0}$

$$A \rfloor B = (A \wedge B^*)^*, \quad A \lfloor B = (A^* \wedge B)^*, \quad A \cdot B = (A \wedge B^* + A^* \wedge B)^* - A * B. \quad (10)$$

For different signatures these relationships are valid up to sign. However, in principle these correspondences can be made exact.

6. Features of the Clifford Package

At present the `clifford` package implements more extensive functionality related to Clifford algebras [15]. The code is distributed under GNU Lesser General Public License. A permanent repository of the presented version of the package is available through the Zenodo repository [14]. A development version is hosted at GitHub (`http://dprodanov.github.io/clifford/`). The package includes also a large number of unit tests for the core functionality and offers several interactive demonstrations. The intention is to have `clifford` serving as core engine of any type of Clifford algebra computations possible in Maxima. Since there are multiple possible directions of development every new functionality set (for example geometric calculus or visualizations) is spun off from the core `clifford` engine in a specialized package. To fully support this strategy, it

is endorsed with minimalistic design approach, which surprisingly coincides with the Clifford algebra construction offered by [9], about which the author was not aware until recently. The package relies extensively on the Maxima simplification functionality, and its features are fully integrated into the Maxima simplifier. The `clifford` package defines multiple rules for pre- and post-simplification of Clifford products, outer products, scalar products, inverses and powers of Clifford vectors. Using this functionality, any combination of products can be simplified into the canonical representation. The main features of the package are summarized in Table 2.

Table 2. Main functions in the `clifford` package.

function name	functionality
simplification	
`cliffsimpall` (expr)	full simplification of expressions
`dotsimp1` (ab)	canonic reordering of dot products
`dotsimpc` (ab)	simplification of dot products
`dotinvsimp` (ab)	simplification of inverses
`powsimp` (ab)	simplification of exponents
involutions	
`dotreverse` (ab)	Clifford reverse of product
`cinvolve` (expr)	Clifford involution of expression
`dotconjugate` (expr)	Clifford conjugate of expression
grade functions	
`grade` (expr)	grade decomposition of expression
`scalarpart` (expr)	$< expr >_0$
`vectorpart` (expr)	$< expr >_1$
`grpart` (expr,k)	$< expr >_k$
`mvectorpart` (expr)	$< expr >_{2+}$
`bdecompose` (expr)	blade decomposition of expression

Example 23 (Quaternions). *The quaternion algebra $C\ell_{0,2}$ can be initialized by issuing the command*

```
clifford(e,0,2);
```

The function `mtable1` *computes and simplifies the geometric products of a list of elements and returns the multiplication matrix.*

```
mtable1([1, e[1],e[2], e[1] . e[2]]);
```

$$\begin{pmatrix} 1 & e_1 & e_2 & e_1.e_2 \\ e_1 & -1 & e_1.e_2 & -e_2 \\ e_2 & -e_1.e_2 & -1 & e_1 \\ e_1.e_2 & e_2 & -e_1 & -1 \end{pmatrix}$$

Here we give some examples in $\mathbb{G}^3$

Example 24 (Outer product). *Outer product evaluation*

```
e[1]&e[2] & e[3];
```

$$e_1 e_2 e_3$$

```
(1+e[1])&(1+e[1]);
```

$$1+2e_1$$

```
(1+e[1])&(1-e[1]);
```

$$1.$$

In a similar way the outer product table can be computed

```
mtable2o();
```

$$\begin{pmatrix} 1 & e_1 & e_2 & e_3 & e_1\cdot e_2 & e_1\cdot e_3 & e_2\cdot e_3 & e_1\cdot e_2\cdot e_3 \\ e_1 & 0 & e_1\cdot e_2 & e_1\cdot e_3 & 0 & 0 & e_1\cdot e_2\cdot e_3 & 0 \\ e_2 & -e_1\cdot e_2 & 0 & e_2\cdot e_3 & 0 & -e_1\cdot e_2\cdot e_3 & 0 & 0 \\ e_3 & -e_1\cdot e_3 & -e_2\cdot e_3 & 0 & e_1\cdot e_2\cdot e_3 & 0 & 0 & 0 \\ e_1\cdot e_2 & 0 & 0 & e_1\cdot e_2\cdot e_3 & 0 & 0 & 0 & 0 \\ e_1\cdot e_3 & 0 & -e_1\cdot e_2\cdot e_3 & 0 & 0 & 0 & 0 & 0 \\ e_2\cdot e_3 & e_1\cdot e_2\cdot e_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ e_1\cdot e_2\cdot e_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example 25 (Associativity of outer product). *We create three vectors with scalar components*

```
a1:cvect(a),b1:cvect(b), c1:cvect(c);
```

$$a_1 e_1 + a_2 e_2 + a_3 e_3$$

And then test for the associativity of the vector product

```
(a1 & b1 )& c1,factor;
```

$$(-c_1 b_2 a_3 + b_1 c_2 a_3 + c_1 a_2 b_3 - a_1 c_2 b_3 - b_1 a_2 c_3 + a_1 b_2 c_3)\,(e_1 \cdot e_2 \cdot e_3)$$

```
a1 & (b1 & c1),factor
```

$$(-c_1 b_2 a_3 + b_1 c_2 a_3 + c_1 a_2 b_3 - a_1 c_2 b_3 - b_1 a_2 c_3 + a_1 b_2 c_3)\,(e_1 \cdot e_2 \cdot e_3)\,.$$

The modulus of the triple product can be identified as the determinant of the coordinate matrix as will be shown further. Moreover, we can verify an the mixed product identity from vector algebra

$$c \cdot (a \times b) = \det \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

```
c1| (-%iv. (a1 & b1)),  dotsimpc;
```

$$-c_1 b_2 a_3 + b_1 c_2 a_3 + c_1 a_2 b_3 - a_1 c_2 b_3 - b_1 a_2 c_3 + a_1 b_2 c_3$$

```
M1: matrix(
[a[1],a[2],a[3]],
[b[1],b[2],b[3]],
[c[1],c[2],c[3]]
)$
D2:determinant(M1);
```

$$(-c_1 b_2 + b_1 c_2)\,a_3 - a_2\,(-c_1 b_3 + b_1 c_3) + a_1\,(-c_2 b_3 + b_2 c_3)\,.$$

Example 26 (Inner products). *Inner product evaluation*

```
((1+e[1])/2)|( (1+e[1])/2),factor;
```

$$\frac{1+e_1}{2}.$$

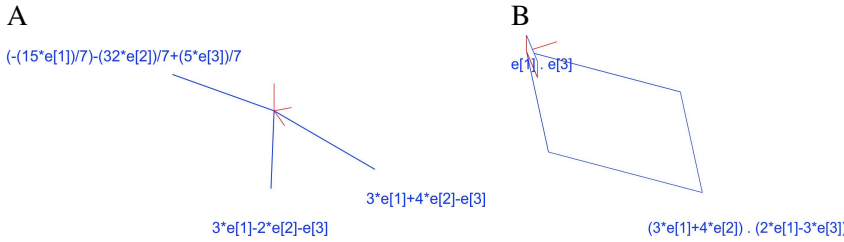
In `clifford` *the left and right contractions and the symmetrical product are supported by the command line switches* `lc` *for left contraction,* `rc` *for right contraction and* `sym` *for the symmetric product. The example below calculates left contraction in* $\mathbb{G}^3$

```
inprotype:lc;
mtable2i();
```

$$\begin{pmatrix} 1 & e_1 & e_2 & e_3 & e_1.e_2 & e_1.e_3 & e_2.e_3 & e_1.e_2.e_3 \\ 0 & 1 & 0 & 0 & e_2 & e_3 & 0 & e_2.e_3 \\ 0 & 0 & 1 & 0 & -e_1 & 0 & e_3 & -e_1.e_3 \\ 0 & 0 & 0 & 1 & 0 & -e_1 & -e_2 & e_1.e_2 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -e_3 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & e_2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -e_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}.$$

6.1. Visualization of Blades

The `clidraw` package offers elementary functionality for visualization of multivector operations. Projections of multivectors on two and three dimensional subspaces can be drawn using the underlying Maxima graphical functionality. Reflection of the vector $a = 3e_1 + 4e_2 - e_3$ by $b = 3e_1 - 2e_2 - e_3$ is plotted in Fig. 2A. Bi-vectors spanned by the vectors $\{e_1, e_2\}$ and $\{3e_1 + 4e_2, 2e_1 - 3e_3\}$ are plotted in Fig. 2B.



A – plotting of vector operations; B – plotting of bi-vectors.

Figure 2. Vector and bi-vector display in the `clidraw` package.

7. Generalized Derivatives in Clifford Algebras

7.1. Multi-Vector Derivatives

Vector derivative of a function is well-defined in the case a Clifford algebra is non-degenerate ($r = 0$), which will be assumed further on.

Definition 27 (Reciprocal frames). *Consider the multivector $r = x_k e_k$, in the subspace spanned by $e_{i_1} \wedge \ldots e_{i_m}$. The frame e_k can be extracted by partial differentiation with respect to coordinates*

$$e_k = \frac{\partial r}{\partial x_k}.$$

Then for every fame we define the reciprocal frame as $e^k = e_k^{-1}$ so that

$$e_k * e^j = \delta_k^j.$$

Consider the m-vector v spanned by the subspace $v \wedge e_1 \wedge \ldots e_m = 0$. Then we can compute the projections using the reciprocal frames

$$v_j = e^J * v, \;\; e_J \wedge e_1 \wedge \ldots e_m = 0.$$

Definition 28 (Directional derivative). *Define the directional derivative as the projection along the unit norm (multi)vector v*

$$(v * \nabla) f(x) \equiv {}_v\nabla f(x) := \sum_j v_j \frac{\partial f}{\partial x_j}. \tag{11}$$

Moreover, it is clear that

$${}_v\nabla f(x) = \sum_J v * \underbrace{\frac{\partial x_j}{\partial r}}_{e^J} \underbrace{\frac{\partial f}{\partial x_j}}_{\partial_J f(x)} = \sum_J \left(e^J * v\right) \partial_J f(x).$$

Definition 29 (Geometric derivative). *The vector derivative ∇_r of the multi-vector valued function $f(x)$ is defined as*

$$\nabla_r f(x) := \frac{df}{dr} = \sum_J \underbrace{\frac{\partial x_j}{\partial r}}_{e^J} \frac{\partial f}{\partial x_j} = e^J \partial_J f(x) \tag{12}$$

so that

$$\partial_J f(x) = (e_J * \nabla_r) f(x) = \nabla_{r * e_J} f(x).$$

The definition can be readily extended to multi-vectors. The vector (but not in general the multivector) derivative can be decomposed into inner and exterior components

$$\nabla_r F = \nabla_r \cdot F + \nabla_r \wedge F$$

in a way similar to the decomposition of the geometric product.

Another useful decomposition can be based on the projection against a blade

$$\nabla_r F = {}_v\nabla F + {}_{\perp v}\nabla F$$

so that the expressions

$$v\,({}_v\nabla)\,F = v\cdot\nabla_r F,\;\; v\,({}_{\perp v}\nabla)\,F = v\wedge\nabla_r F$$

give components along the directions of a **vector** v.

A local fractional variant of the derivative called *fractional velocity* [5,12] can also be constructed. We define directional limits

$$\partial^{\beta}_{\pm J} f(x) = \pm\lim_{\varepsilon\to 0}\frac{f(x_J\pm\varepsilon)-f(x)}{\varepsilon^{\beta}}$$

for the exponent $0<\beta\leq 1$. Then for functions of bounded variation [12]

$$\partial^{\beta}_{\pm k} f(x) = \lim_{\varepsilon\to 0}\frac{1}{\beta}\varepsilon^{1-\beta}\partial_k f(x_k\pm\varepsilon).$$

We start from a projection definition [17]

Definition 30. *For a unit vector v*

$$v*\nabla^{\beta}_{\pm} f(x) := \sum_J \left(v*e^J\right)\partial^{\beta}_{\pm J} f(x) \tag{13}$$

for the exponent $0<\beta\leq 1$.

Definition 31. *Then we proceed in defining*

$$\nabla^{\beta}_{\pm} f(x) := \sum_J e^J\partial^{\beta}_{\pm J} f(x) \tag{14}$$

so that

$$\nabla^{\beta}_{\pm} f(x) = \frac{1}{\beta}\lim_{\varepsilon\to 0}\sum_J e^J\varepsilon^{1-\beta}\partial_J f(x_J\pm\varepsilon).$$

8. Geometric Calculus Functionality in Maxima

Maxima supports symbolic differentiation and integration in the real and complex domains. This functionality can be extended also to Clifford numbers. The `cliffordan` package, which is based on `clifford` implements symbolical differentiation based on the vector derivative. The main building block is the total derivative function shown in Listing 3.

The main functions of the packages are listed in Table 3.

Listing 3. Clifford-valued total differentiation of an expression in `cliffordan`.

```
/*
Clifford-valued
total differentiation; */

ctotdiff(f,x):=block( [ret:0, lv],
    if mapatom(x) then
        ret: diff( f, x )
    else (
        lv: sublist( listofvars(x), lambda ([z], freeof(asymbol,z
            ))),
        for u in lv do
            ret: ret + cinv(diff(x, u)). subst(``.'', ``*'', diff
                ( f, u ))
    ),
ret
);
```

Listing 4. Clifford-valued directional differentiation of an expression in `cliffordan`.

```
/*
Clifford-valued
directional derivative
*/
 cdirdiff(f, v, x):=block( [ret:0, lv, uu:0, qq],
    if mapatom(x) then
        ret: diff( f, x )
    else (
        lv: sublist( listofvars(x), lambda ([z], freeof(asymbol,
            z))),
        for u in lv do (
        qq: cinv(diff(x, u)),
        uu: scprod(qq , v),
        ret: ret + uu * subst(``.'', ``*'', diff( f, u ))
    )
    ),
    factorby(dotsimpc(ret), %elements)
);
```

8.1. Potential Problems in $C\ell_{3,0}$

We shall give a presentation of the potential problem in the geometric algebra $\mathbb{G}^3 = C\ell_{3,0}$. In electrostatic or magnetostatic setting the Green's function of the system

$$\nabla_r G = \delta(r)$$

Table 3. Main functions in `cliffordan`.

name	functionality
`ctotdiff`(f, x)	total derivative w.r.t. multivector x
`ctotintdiff`(f, x)	inner total derivative w.r.t. x
`ctotextdiff`(f, x)	outer total derivative w.r.t. x
`vectdiff`(f, ee, k)	vector derivative of order k w.r.t. basis vector list *ee*
`mvectdiff`(f, x, k)	multivector derivative of order k w.r.t. multivector x
`parmvectdiff`(f, x, k)	partial multivector derivative of order k w.r.t. multivector x
`convderiv`(f, t, xx, [vs])	convective derivative w.r.t. multivector x
`coordsubst`(x, eqs)	substitutes coordinates in multivector x w.r.t. new variables in the list *eqs*
`clivolel`(x, eqs)	computes volume element of $Span\{x\}$ w.r.t. new variables in the list *eqs*

where $\delta(x)$ is the Dirac's delta function is

$$G(x,y,z) = \frac{e_1 x + e_2 y + e_3 z}{4\pi \sqrt{(x^2+y^2+z^2)^3}}. \tag{15}$$

Direct calculation issuing the commands

```
xx:e[1]*x+e[2]*y+e[3]*z$
G:xx/sqrt(-cnorm(xx))^3$
mvectdiff(GG,xx)
```

evaluates to 0 for $x \neq 0, y \neq 0, z \neq 0$. In the last calculation the factor is skipped for simplicity. $G(x,y,z)$ can be derived from the following scalar potential

$$V(x,u,z) = -\frac{C}{\sqrt{x^2+y^2+z^2}} \tag{16}$$

where C is an arbitrary constant matching the initial or boundary conditions

```
mvectdiff(-1/sqrt(x^2+y^2+z^2),xx);
```

yielding

$$\frac{e_1 x + e_2 y + e_3 z}{(x^2+y^2+z^2)^{\frac{3}{2}}}.$$

8.2. Coordinate Transformations

Geometric algebra implementation in `clifford` allows for transparent coordinate substitutions.

Example 32. *In the following example we verify the properties of the Green's function in cylindrical coordinates*

```
GG_c:coordsubst(G, cyl_eq),factor;
```

$$\frac{e_1\rho\cos\phi + e_2\rho\sin\phi + e_3 z}{(\rho^2+z^2)^{\frac{3}{2}}}$$

```
rc:coordsubst(r, cyl_eq);
```

$$(e_1\cos\phi + e_2\sin\phi)\,\rho + e_3 z$$

```
V:coordsubst(-1/sqrt(-cnorm(r)),cyl_eq);
```

$$-\frac{1}{\sqrt{\rho^2+z^2}}$$

```
rc:coordsubst(r, cyl_eq);
```

$$(e_1\cos\phi + e_2\sin\phi)\,\rho + e_3 z$$

```
mvectdiff(V,rc);
```

yielding

$$\frac{e_1\rho\cos\phi + e_2\rho\sin\phi + e_3 z}{(\rho^2+z^2)^{\frac{3}{2}}}$$

as expected.

```
mvectdiff(GG_c,rc);
```

which yields 0 *as expected.*

8.3. Homogeneous d'Alembert Equation in `Clifford`

Clifford algebras offer a convenient way of combining objects of different grades in the form of inhomogeneous sums. For example, the sum of a scalar and a 3-vector in $\mathbb{G}^3$ (called a *paravector*) is a well-defined inhomogeneous object that can be used in calculations. The Euler-Lagrange field equations corresponding to the scalar Lagrangian density $\mathscr{L}(q,\partial_x q)$ involving a field u and its derivatives $\partial_x u$ with respect to the coordinates x are derived in [8] using the fundamental theorem of Geometric Calculus. Here we give another derivation based on the usual tensor notation. We start from the system of n Euler-Lagrange equations in fields u_a and their partial derivatives $u_{a;\nu}$ (using Einstein's summation convention)

$$\frac{\partial \mathscr{L}}{\partial u_a} - \frac{\partial}{\partial x^\nu}\frac{\partial \mathscr{L}}{\partial u_{a;\nu}} = 0.$$

We introduce auxiliary notation $u = e_a u_a$ and

$$p = \nabla_x u,\ p_\nu = (e_\nu * \nabla_x)\, u$$

so that $p_\nu = e_a u_{a;\nu}$ in the (extended) basis. Then the following identity (no summation by ν) holds

$$\frac{\partial}{\partial p^\nu}\mathscr{L} = \left(\frac{dp_\nu}{du_{a;\nu}}\right)^{-1}\frac{\partial \mathscr{L}}{\partial u_{a;\nu}} = e^a \frac{\partial \mathscr{L}}{\partial u_{a;\nu}} = \nabla_{p_\nu}\mathscr{L}.$$

We proceed by multiplying the input equations by e^a basis vectors and add them up to obtain

$$\frac{\partial \mathscr{L}}{\partial u_a}e_a - e^\nu \frac{\partial}{\partial x^\nu}\, e_\nu e^a \frac{\partial \mathscr{L}}{\partial u_{a;\nu}} = 0$$

where we recognize further

$$\frac{\partial \mathscr{L}}{\partial u_a}e_a - e^\nu \frac{\partial}{\partial x^\nu}\, e_\nu \nabla_{p_\nu}\mathscr{L} = \nabla_u \mathscr{L} - (\nabla_x * \nabla_p)\mathscr{L}.$$

So that finally

$$\nabla_u \mathscr{L} - (\nabla_x * \nabla_p)\mathscr{L} = 0.$$

Example 33. *The wave equation was derived in 1747 by Jean-Baptiste le Rond d'Alembert in the analysis of the problem of vibrating strings. The following application derives this equation using the Euler-Lagrange framework. Here the derivation procedure is replicated using the* `clifford` *package. Let A be the paravector potential given by*

$$A = A_t + e_1 A_x + e_2 A_y + e_3 A_z \tag{17}$$

which can be constructed by the command

```
AA: celem(A,[t,x,y,z])$
```

Then the geometric derivative object is given by applying geometric derivative using a paravector $x = t - r$

$$F = \nabla_{t-r} A \tag{18}$$

given by the command

```
F:mvectdiff(AA,t-r)$
```

resulting in a mixture of scalar, vector and bi-vector components

$$F = \langle F \rangle_0 + \langle F \rangle_1 + \langle F \rangle_2 . \tag{19}$$

The wave equation for the paravector potential can be derived from a purely quadratic Lagrangian composed from the components of the geometric derivative of F

$$\mathscr{L}_a = \frac{1}{2} \langle F^2 \rangle_0 \tag{20}$$

```
L:lambda([x],1/2*scalarpart(cliffsimpall(x.x)))(F);
```

$$\begin{aligned}(&{A_{t_t}}^2+{A_{t_x}}^2+{A_{t_y}}^2+{A_{t_z}}^2-2A_{t_x}A_{x_t}+{A_{x_t}}^2-2A_{t_t}A_{x_x}+{A_{x_x}}^2-{A_{x_y}}^2-{A_{x_z}}^2-2A_{t_y}A_{y_t}\\ &+{A_{y_t}}^2+2A_{x_y}A_{y_x}-{A_{y_x}}^2-2A_{t_t}A_{y_y}+2A_{x_x}A_{y_y}+{A_{y_y}}^2-{A_{y_z}}^2-2A_{t_z}A_{z_t}+{A_{z_t}}^2+2A_{x_z}A_{z_x}\\ &-{A_{z_x}}^2+2A_{y_z}A_{z_y}-{A_{z_y}}^2-2A_{t_t}A_{z_z}+2A_{x_x}A_{z_z}+2A_{y_y}A_{z_z}+{A_{z_z}}^2)/2\end{aligned}$$

where we can recognize the identity

$$\mathscr{L}_a = \frac{1}{2} \left(\langle F \rangle_0^2 + \langle F \rangle_1^2 + \langle F \rangle_2^2 \right) \tag{21}$$

```
S:scalarpart(F)$
V:vectorpart(F)$
Q:grpart(F,2)$
L-1/2*(S.S+V.V+Q.Q),cliffsimpall;
```

yielding 0.

Finally, applying the functional derivative, that is the Euler-Lagrange functional yields

```
dA:mvectdiff(AA,r);
EuLagEq2(L, t+r,[AA,dA]);
```

$$\left(-A_{ttt}+A_{txx}+A_{tyy}+A_{tzz}\right)+e_1\left(-A_{xtt}+A_{xxx}+A_{xyy}+A_{xzz}\right)$$
$$+e_2\left(-A_{y_{tt}}+A_{y_{xx}}+A_{y_{yy}}+A_{y_{zz}}\right)+e_3\left(-A_{ztt}+A_{zxx}+A_{zyy}+A_{zzz}\right)$$

which can be recognized as the D'Alembertian for the components of the paravector potential

$$\nabla_{t+r}\nabla_{t-r}A=0. \tag{22}$$

In the last derivation based on paravectors we used implicitly the space-time split used by Hestenes. The `Maxima` *expression can be decomposed in a matrix form as*

```
(%i51)  bdecompose(%);
```

$$[[[1],\left(-A_{ttt}+A_{txx}+A_{tyy}+A_{tzz}\right)],$$
$$[[e_1,e_2,e_3],\begin{pmatrix}-A_{xtt}+A_{xxx}+A_{xyy}+A_{xzz}\\-A_{y_{tt}}+A_{y_{xx}}+A_{y_{yy}}+A_{y_{zz}}\\-A_{ztt}+A_{zxx}+A_{zyy}+A_{zzz}\end{pmatrix}],$$
$$[[0],(0)],[[0],(0)]]$$

if one wishes to solve for individual components.

9. Outlook

This paper demonstrates applications of Clifford algebra in several areas of applied sciences, notably: visualizations of geometric objects, coordinate transformations, derivation of the Green function of the Poisson's equation and variational problems. In summary, Clifford algebra packages of Maxima can be considered as sufficiently mature for use as research tools. Further development of the Clifford algebra tools will be directed towards more extensive linear algebra functionality and fractional calculus.

Appendix: Example Code listings

Listing 5. Inner and outer products example.

```
/* Initialization */
load('clifford);
clifford(e,0,2);
mtable1([1, e[1],e[2], e[1] . e[2]]);
clifford(e, 3);
e[1] &e[2] &e[3];
(1+e[1])&(1+e[1]);
(1+e[1])&(1-e[1]);
mtable2o();
inprotype:lc;
((1+e[1])/2)|( (1+e[1])/2),factor;
mtable2i();
```

Listing 6. Associatvity of outer products example.

```
/* Initialization */
load('clifford);
clifford(e, 3);
a1:cvect(a),b1:cvect(b), c1:cvect(c);
b1:cvect(b);
c1:cvect(c);
(a1 & b1 )& c1,factor;
a1 & (b1 & c1),factor;
L1:a1 | (b1 & c1);
L2:(a1|b1).c1 - (a1 | c1) .b1,expand;
L1-L2;
D1:c1| (-%iv. (a1 & b1)),dotsimpc;

M1: matrix(
[a[1],a[2],a[3]],
[b[1],b[2],b[3]],
[c[1],c[2],c[3]]
);
D2:determinant(M1);

equal(D1,D2),pred;
```

Listing 7. Potential theory example.

```
/* Initialization */
load('clifford);
load ('cliffordan);
clifford(e,3);

r:cvect([x,y,z]);
G:r/sqrt(-cnorm(r))^3;
mvectdiff(G,r);
mvectdiff(-1/sqrt(-cnorm(r)),r);
mvectdiff(-1/sqrt(-cnorm(r)),r,2);


cyl_eq:[x=rho*cos(phi), y=rho*sin(phi)];
declare( [rho, phi], scalar);
GG_c:coordsubst(G, cyl_eq),factor;
rc:coordsubst(r, cyl_eq);
mvectdiff(GG_c,rc);
V:coordsubst(-1/sqrt(-cnorm(r)),cyl_eq);
mvectdiff(V,rc);
```

Listing 8. Lagrangian example.

```
/* Initialization */
load('clifford);
load ('cliffordan);
clifford(e,3);

derivabbrev:true;
AA: celem(A,[t,x,y,z]);
dependsv(A,[t,x,y,z]);
r:cvect([x,y,z]);
F:mvectdiff(AA,t-r);
L:lambda([x],1/2*scalarpart(cliffsimpall(x.x)))(F);
S:scalarpart(F);
V:vectorpart(F);
Q:grpart(F,2);
L-1/2*(S.S+V.V+Q.Q),cliffsimpall;
dA:mvectdiff(AA,r);
EuLagEq2(L, t+r,[AA,dA]);
bdecompose(%);
```

Acknowledgments

The work is partially supported by a grant from Research Fund – Flanders (FWO), contract numbers G.0C75.13N, VS.097.16N.

References

[1] Abłamowicz R. and Fauser B., *Mathematics of CLIFFORD - A Maple Package for Clifford and Graßmann Algebras*, Advances in Applied Clifford Algebras **15** (2002) 157–181.

[2] Abłamowicz R. and Fauser B., *Clifford and Graßmann Hopf Algebras via the BIGEBRA Package for Maple*, Computer Physics Communications **170** (2005) 115–130.

[3] Abłamowicz R. and Fauser B., *Using Periodicity Theorems for Computations in Higher Dimensional Clifford Algebras*, Advances in Applied Clifford Algebras **24** (2014) 569–587.

[4] Aragon-Camarasa G., Aragon-Gonzalez G., Aragon J. and Rodriguez-Andrade M., *Clifford Algebra with Mathematica*, In: Recent Advances in Applied Mathematics, I. Rudas (Ed), Proceedings of AMATH '15 WSEAS Press, Budapest 2015, pp. 64–73.

[5] Cherbit G., *Local Dimension, Momentum and Trajectories*, In: Fractals, Non-integral Dimensions and Applications. G. Cherbit (Ed), John Wiley & Sons, Paris 1991, pp. 231– 238.

[6] Dorst L., *The Inner Products of Geometric Algebra*, In: Applications of Geometric Algebra in Computer Science and Engineering, Birkhäuser, Boston 2002, pp. 35–46.

[7] Hestenes D., *Space-Time Algebra*, 2nd Edn, Birkhäuser, Basel 2015.

[8] Lasenby A., Doran C. and Gull S., *A Multivector Derivative Approach to Lagrangian Field Theory*, Foundations of Physics **23** (1993) 1295–1327.

[9] Macdonald A., *An Elementary Construction of the Geometric Algebra*, Advances in Applied Clifford Algebras **12** (2002) 1 – 6.

[10] Maxima Project, http://maxima.sourceforge.net/docs/manual/maxima.html `Maxima 5.35.1 Manual` (2014).

[11] Porteus I., *Clifford Algebras and the Classical Groups*, 2nd Edn, Cambridge University Press, Cambridge 2000.

[12] Prodanov D., *Fractional Variation of Hölderian Functions*, Fract. Calc. Appl. Anal. **18** (2015) 580 – 602.

[13] Prodanov D., *Clifford: A Light-Weight Package for Performing Geometric and Clifford Algebra Calculations*, http://dx.doi.org/10.5281/zenodo.61261 (2016).

[14] Prodanov D., *Some Applications of Fractional Velocities*, Fract. Calc. Appl. Anal. **19** (2016) 173 – 187.

[15] Prodanov D. and Toth V., *Sparse Representations of Clifford and Tensor Algebras in Maxima*, Advances in Applied Clifford Algebras (2016) 1–23.

[16] Sangwine S. and Hitzer E., *Clifford Multivector Toolbox* (*for MATLAB*), Advances in Applied Clifford Algebras (2016) 1–20.

[17] Wang X., *Fractional Geometric Calculus*: *Toward a Unified Mathematical Language for Physics and Engineering*, In: Proceedings of The Fifth Symposium on Fractional Differentiation and its Applications (FDA12), Hohai University, Nanjing 14–17 May 2012.

Dimiter Prodanov
Department of Environment, Health and Safety
Neuroscience Research Flanders
IMEC, Leuven, BELGIUM
E-mail address: dimiter.prodanov@imec.be