Sparse Representations of Clifford and Tensor Algebras in Maxima

2 authors:

Dimiter Prodanov
IMEC International

58 PUBLICATIONS   **343** CITATIONS

SEE PROFILE

V. T. Toth
N/A

48 PUBLICATIONS   **702** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project

Multiscale segmentation of histological images View project

SPARSE REPRESENTATIONS OF CLIFFORD AND TENSOR ALGEBRAS IN MAXIMA

DIMITER PRODANOV[1] AND VIKTOR T. TOTH [2]

ABSTRACT. Clifford algebras have broad applications in science and engineering. The use of Clifford algebras can be further promoted in these fields by availability of computational tools that automate tedious routine calculations. We offer an extensive demonstration of the applications of Clifford algebras in electromagnetism using the geometric algebra $\mathbb{G}^3 \equiv C\ell_{3,0}$ as a computational model in the Maxima computer algebra system. We compare the geometric algebra-based approach with conventional symbolic tensor calculations supported by Maxima, based on the `itensor` package. The Clifford algebra functionality of Maxima is distributed as two new packages called `clifford` - for basic simplification of Clifford products, outer products, scalar products and inverses; and `cliffordan` - for applications of geometric calculus.

[1] Department of Environment, Health and Safety, Neuroscience Research Flanders, IMEC vzw, Leuven, Belgium

[2] Center for Research on Integrated Sensors Platforms Carleton University Ottawa, Ontario, Canada

1. INTRODUCTION

While Computer Algebra Systems (CAS) cannot replace mathematical intuition, they can facilitate routine calculations and teaching. This paper focuses on applications of two packages implementing abstract algebras in the popular open-source *Maxima* CAS: the `itensor` package implementing indicial tensor manipulation and a new package for Clifford algebra called `clifford`, along with its companion package `cliffordan`. While an extensive overview of the Maxima system is not our objective, we offer some details that demonstrate the major advantages of Maxima over purely numerical systems, such as Matlab (Mathworks, Natick, MA, USA).

1.1. **System level functionality.** Maxima is derived from one of the first ever computer algebra systems, MACSYMA (Project MAC's SYmbolic MAnipulation System), developed by the Mathlab group of the MIT Laboratory

for Computer Science during the years 1969-1972. Maxima is written entirely in Lisp and is distributed under open source license[1]. Maxima has its own programming language, which is particularly well-suited for handling symbolical mathematical expressions or mixed numerical-symbolical expressions. In addition, the system supports 64-bit precision floating point and arbitrary precision arithmetics. 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 system also offers the possibility of running batch unit tests.

Maxima supports several primitive data types [11]: *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. 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. Maxima distinguishes between two forms of applications of operators – forms which are `noun`s and forms which are `verb`s. 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 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.

Listing 1: Dot products and exponent simplification rules in `clifford`.

```
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)),
tellsimpafter(aa[kk]^nn, powsimp(aa[kk]^nn)),
tellsimpafter(aa[kk]^^nn, powsimp(aa[kk]^^nn))
);
```

1.2. **Expression representation and transformation in Maxima.** The manner in which Maxima represents expressions, function calls and index expressions using the Lisp language is particularly relevant for the `itensor` and `clifford` 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 *lambda* construct and its value is the value of the last assigned member. This is a design

---

[1]Maxima is distributed under GNU General Public License and developed and supported by a group of volunteers
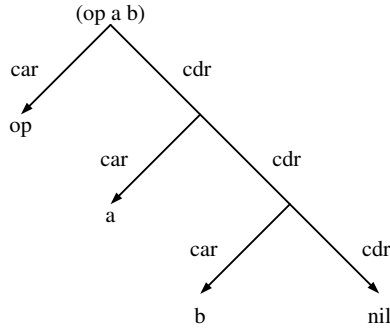
FIGURE 1. Expression representation in Lisp. A general expression can be represented as a list with the first element being the operator `op` and the rest of the elements representing operator arguments. For instance, the expression `op(a,b)` will be represented by the list `(op a b)`, which is the ordered pair of the atom `op`, and another list, `(a b)`, which, in turn, is represented by another ordered pair.

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. Or, to be more precise, the idea of an ordered pair, the first element of which is the head (or `car`), the second element the tail (or `cdr`) of the 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 transformed to another expression.

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 (see Fig. 2). The simplifier subroutine operates by descending the tree of an expression until it gets to atoms, and then simplifies the smallest pieces and backs out.

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. 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`.

This rule-based approach to computer algebra is particularly suitable to implement "sparse" representations of algebraic objects. For instance, a tensor object may be defined only in terms of its indices and the transformation rules that it obeys, without ever assigning values to tensor components. This
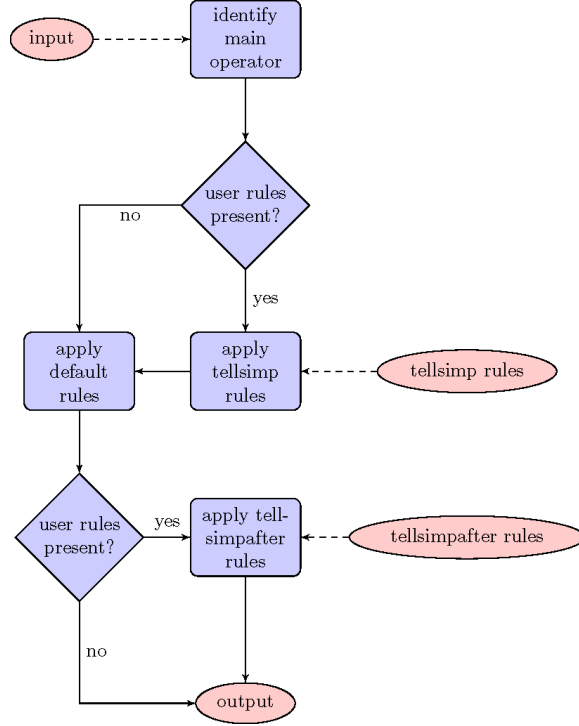
FIGURE 2. Expression simplification in Maxima.

is how tensors are implemented in the `itensor` package of Maxima. Similarly, a multivector can be defined symbolically as a linear combination of basis vectors, without ever assigning those basis vectors any specific values or representing them by matrices. This is the approach followed by the built-in `atensor` package and the `clifford` package that is described in the present paper.

2. TENSOR ALGEBRA REPRESENTATIONS IN MAXIMA

Tensors are abstract objects that describe multilinear relations between geometric objects. A coordinate-independent way to define tensors is by multilinear maps. In this approach, a tensor T of type (p, q) is defined as a map that is linear in each of its arguments, by the equation

$$T : \underbrace{V^\star \otimes \ldots V^\star}_{p} \otimes \underbrace{V \otimes \ldots V}_{q} \to \mathbb{K}, \qquad (2.1)$$

where $\mathbb{K}$ is a field, V is a (finite-dimensional) vector space over $\mathbb{K}$, while $V^\star$ is the corresponding dual space of co-vectors.

Maxima has two major packages for tensor manipulation. The `ctensor` package implements component representation of tensors, with functionality that is especially tailored for applications in general relativity. The `itensor` package treats tensors as opaque objects, manipulated via their indices. As

such, `itensor` is especially suited for computations where general covariance is maintained, including general relativity. The package has built-in support for the metric and curvature tensors, for covariant differentiation, for the Kronecker delta, and for utilizing the symmetry properties of tensors for algebraic simplification. All this is accomplished without the need to define tensor components.

The `itensor` package also has facilities for tensor calculus, including functional differentiation with respect to tensor quantities. These facilities make it possible to use `itensor` to investigate field theories that are formulated using a Lagrangian density functional. By way of a demonstrative example, let us consider the Lagrangian density for the Maxwell field (hereafter we assume that indices run from 1 through the number of dimensions D, and we use the Einstein summation convention, $x_\mu y^\mu = \sum_{\mu=1}^{D} x_\mu y^\mu$):

$$\mathcal{L}_{\text{EM}} = -\frac{1}{4} F_{\kappa\lambda} F^{\kappa\lambda} + j_\kappa A^\kappa, \tag{2.2}$$

where $F_{\kappa\lambda} = \partial_\kappa A_\lambda - \partial_\lambda A_\kappa$ is the electromagnetic field tensor corresponding to the 4-potential A_μ.

The Euler-Lagrange equation that corresponds to this Lagrangian is written as

$$\frac{\partial(\sqrt{-g}\mathcal{L}_{\text{EM}})}{\partial A_\mu} - \nabla_\nu \frac{\partial(\sqrt{-g}\mathcal{L}_{\text{EM}})}{\partial(\nabla_\nu A_\mu)} = 0, \tag{2.3}$$

where ∇_ν is the covariant derivative with respect to the coordinate x^μ.

For the sake of simplicity of discussion, we restrict the presentation to rectilinear coordinates of special relativity, such that the square root of metric determinant $\sqrt{-g}$ can be assumed to be constant and removed from consideration, and covariant derivatives can be replaced with ordinary partial derivatives with respect to the coordinates, i.e., $\partial_\mu = \partial/\partial x^\mu$. Under these circumstances, the Euler-Lagrange equation becomes

$$\frac{\partial\mathcal{L}_{\text{EM}}}{\partial A_\mu} - \partial_\nu \frac{\partial\mathcal{L}_{\text{EM}}}{\partial(\partial_\nu A_\mu)} = 0. \tag{2.4}$$

The first term in this equation is trivial: $\partial\mathcal{L}_{\text{EM}}/\partial A_\mu = j^\mu$, as follows directly from the definition of $\mathcal{L}_{\text{EM}}$. As to the second term, we first note that

$$F_{\kappa\lambda} F^{\kappa\lambda} = F_{\kappa\lambda} g^{\kappa\alpha} g^{\lambda\beta} F_{\alpha\beta},$$

and that

$$\frac{\partial F_{\kappa\lambda}}{\partial(\partial_\nu A_\mu)} = \delta_\kappa^\nu \delta_\lambda^\mu - \delta_\lambda^\nu \delta_\kappa^\mu,$$

thus

$$\frac{\partial}{\partial(\partial_\nu A_\mu)} F_{\kappa\lambda} F^{\kappa\lambda} = (\delta_\kappa^\nu \delta_\lambda^\mu - \delta_\lambda^\nu \delta_\kappa^\mu) F^{\kappa\lambda} + F^{\alpha\beta}(\delta_\alpha^\nu \delta_\beta^\mu - \delta_\beta^\nu \delta_\alpha^\mu) = 4F^{\nu\mu} = -4F^{\mu\nu}. \tag{2.5}$$

Therefore

$$\partial_\nu \frac{\partial \mathcal{L}_{\text{EM}}}{\partial(\partial_\nu A_\mu)} = -\frac{1}{4}\partial_\nu \frac{\partial}{\partial(\partial_\nu A_\mu)} F_{\kappa\lambda} F^{\kappa\lambda} = \partial_\nu F^{\mu\nu} = F^{\mu\nu}_{,\nu}, \tag{2.6}$$

where we used the common shorthand notation $\partial_\mu A = A_{,\mu}$ for ordinary partial coordinate derivatives. Finally, we can now put everything together and obtain the explicit form of the Euler-Lagrange equation, which amounts to the definition of the 4-current:

$$j^\mu + F^{\mu\nu}_{,\nu} = 0. \tag{2.7}$$

Example. In the following Maxima code snippets, input expressions in Maxima are sequentially labeled by `%iN`, while the output may be labeled by `%oN`, where N is a number, starting with 1, that is incremented by 1 for each new input line. Commands are grouped together between parentheses. Multiple commands can be entered by separating them with a comma. The semicolon or the dollar sign terminate a command line; the latter can be used if the suppression of output is desired (e.g., to avoid cluttering the terminal display with very large intermediate expressions.) For clarity of presentation the output labels are suppressed and the expressions are colored. Expressions are only minimally edited for improvement of readability.

To carry out the derivation (2.3)–(2.7) using `itensor` in Maxima, after loading the package and configuring the metric (see Listing 2 in the Appendix), we define the tensor $F_{\mu\nu}$ in terms of its components and construct $\mathcal{L}_{\text{EM}}$[2]:

```
(%i3) components(F([m,n],[]),A([n],[],m)-A([m],[],n))$
(%i4) L:ishow(-1/4*F([k,l])*F([a,b],[])*g([],[k,a])*g([],[l,b])
      +j([k],[])*A([l],[])*g([],[k,l]))$
```

$$g^{kl} j_k A_l - \frac{g^{ka} g^{lb} (A_{b,a} - A_{a,b}) (A_{l,k} - A_{k,l})}{4}.$$

We can now discard the component definition of $F_{\mu\nu}$ as it has served its purpose, and we would like the final result to appear in terms of $F_{\mu\nu}$ for simplicity. Instead, we define the symmetry properties of $F_{\mu\nu}$:

```
(%i5) remcomps(F)$
(%i6) decsym(F,0,2,[],[anti(all)])$
```

Next, we compute one of the terms of the Euler-Lagrange equation, and then instruct Maxima to re-express this result in terms of $F_{\mu\nu}$:

```
(%i7) ishow(diff(L,A([m],[],n)))$
```

$$-\frac{g^{ka} g^{lb} (A_{b,a} - A_{a,b}) (\delta^n_k \delta^m_l - \delta^m_k \delta^n_l)}{4} - \frac{g^{ka} g^{lb} (\delta^n_a \delta^m_b - \delta^m_a \delta^n_b) (A_{l,k} - A_{k,l})}{4}$$

```
(%i8) ishow(subst(F([a,b],[])+A([a],[],b),A([b],[],a),%))$
```

---

[2]The `ishow` command is a built-in "pretty print" feature of `itensor` to display tensor quantities

$$-\frac{g^{ka}\, g^{lb}\, F_{ab}\, (\delta^n_k\, \delta^m_l - \delta^m_k\, \delta^n_l)}{4} - \frac{g^{ka}\, g^{lb}\, (\delta^n_a\, \delta^m_b - \delta^m_a\, \delta^n_b)\, (A_{l,k} - A_{k,l})}{4}$$

(%i9) ishow(subst(F([k,l],[])+A([k],[],l),A([l],[],k),%))$

$$-\frac{g^{ka}\, g^{lb}\, F_{ab}\, (\delta^n_k\, \delta^m_l - \delta^m_k\, \delta^n_l)}{4} - \frac{g^{ka}\, g^{lb}\, (\delta^n_a\, \delta^m_b - \delta^m_a\, \delta^n_b)\, F_{kl}}{4}$$

(%i10) ishow(canform(rename(contract(expand(%)))))$

$$-F^{mn}.$$

Finally, we complete the Euler-Lagrange equation by adding the remaining term:

(%i11) ishow(contract(diff(L,A([m],[])))-idiff(%,n)=0)$

$$F^{mn}_{,n} + j^m = 0.$$

Even this relatively simple example demonstrates how `itensor` can serve as an effective tool for the field theorist. Similar derivations can be carried out in curved spacetime, making full use of the metric and covariant differentiation, as well as properly accounting for the volume element $\sqrt{-g}$.

The indicial tensor manipulation capability of `itensor` is directly linked to the component tensor manipulation functionality `ctensor`. A conversion function can translate indexed `itensor` expressions into expression blocks for `ctensor`. Thus, it is possible to use the indicial capabilities of `itensor`, as in the example above, to derive a set of field equations, which can then be solved by postulating a specific coordinate system and metric using `ctensor`.

3. Properties of Clifford algebras in view of sparse implementation strategies

Clifford algebras provide natural generalizations of complex numbers, dual numbers and double numbers. We will give a brief exposition on the properties of Clifford algebras over the reals following Dorst [6].

A Clifford algebra is an associative algebra which is generated by a vector space V spanned by the orthonormal basis $\{e_1 \ldots e_n\}$, over a field $\mathbb{K}$ of characteristic different from 2. The algebra is equipped with an anticommutative product, which is referred to as the *Clifford product*, and a scalar unit denoted conventionally by 1. The fundamental relation satisfied by the orthonormal basis is given by

$$e_i e_j = -e_j e_i, \quad i \neq j. \tag{3.1}$$

The algebra also has a zero element, denoted conventionally by 0.

The square of a vector v in the Clifford algebra $C\ell(V, Q)$ is defined by

$$v\, v := Q(v)\, 1, \quad \forall v \in V, \tag{3.2}$$

where $Q(v)$ denotes a quadratic form. There are different conventions for the sign of the form [12]. In our implementation, `clifford`, both conventions are

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

	$k \bmod 4$			
	0	1	2	3
$A^{\sim}$	+	+	−	−
$A^{\star}$	+	−	−	+

allowed, using a sign switch of the pseudo-norm function `psnorm`. The latter function depends on the elementary product simplification rules defined by the signature of the algebra (Fig. 1).

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. 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. For a given Clifford algebra $C\ell_{p,q,r}(\mathbb{R})$ over the real numbers the quadratic form is, therefore, given by

$$Q(v) := v_1^2 + \ldots + v_p^2 - v_{p+1}^2 - \ldots - v_{p+q}^2. \tag{3.3}$$

For the elements for which $Q(v) \neq 0$ the inverse element can be reduced into the canonical form

$$v^{-1} = \frac{v}{Q(v)}. \tag{3.4}$$

A multivector A of the Clifford algebra is an element of the 2^n-dimensional vector space $P(V)$ spanned by the power set $\{1, e_1, ..., e_n, e_1 e_2, e_1 e_3, ..., e_1 e_2 ... e_n\}$.

The inner (dot) product and the outer, or exterior (wedge) product of two elements a and b are defined by

$$a \cdot b := \frac{a\,b + b\,a}{2}, \tag{3.5}$$

$$a \wedge b := \frac{a\,b - b\,a}{2}. \tag{3.6}$$

A *blade* of grade k is an object having the same number of basis vectors per term. Conventionally, 0-blades are scalars, 1-blades are vectors etc. A general multivector A can be decomposed by the *grade projection operators* $\langle \, \rangle_k$ into a sum of different blades:

$$A = \sum_{k=0}^{n} \langle A \rangle_k. \tag{3.7}$$

There are two principal involutions – the Clifford conjugation $A^{\star}$ and the reversion $A^{\sim}$ having the sign mutations shown in Table 1.

3.1. **Multi-vector derivatives.** The vector derivative of a function is well-defined when the Clifford algebra is non-degenerate $(r \neq 0)$, which will be assumed further on.

Definition 3.1. Consider a vector x given in terms of the basis $\{e_1, \ldots, e_n\}$ as $x = \sum_{i=1}^{n} x^k e_k$. For the basis vectors e_k we can define the reciprocal frame $e^k = e_k^{-1}$. We now define the vector derivative [5] as

$$\nabla_x f = \sum_k e^k \frac{\partial}{\partial x^k} f(x) = e^k \partial_k f(x). \tag{3.8}$$

The vector derivative can be decomposed into inner (directional) and exterior components

$$\nabla_x F = \nabla_x \cdot F + \nabla_x \wedge F \tag{3.9}$$

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

Remark 3.2. The vector derivative can be readily generalized onto general blades if we allow for the arguments to traverse the bases of a general multivector v.

$$\nabla_v f(x) = \sum_{v_i \in P(V)} (v_i)^{-1} \lim_{\epsilon \to 0} \frac{f(x + \epsilon\, v_i) - f(x)}{\epsilon}, \tag{3.10}$$

where $\epsilon > 0$ is a scalar vanishing in limit and the limit procedure is understood in norm. This definition can be literally implemented in Maxima if proper simplification rules are given for the multivector inverse. Such an approach was undertaken in `clifford`.

4. The package `clifford`

In Maxima, the Clifford product is represented by the non-commutative operator symbol $(\cdot)$. At present there are two packages that support Clifford algebras in Maxima. The package `atensor` partially implements generalized (tensor) algebras [11, 13]. The `clifford` package implements more extensive functionality related to Clifford algebras[3]. 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 put into a canonical representation. The main features of the package are summarized in Table 2.

4.1. **Command line simplification in `clifford`.** In the `clifford` package, the inner product is represented by the operator symbol "|" and the outer (exterior, or wedge) product by the operator symbol "&". The package defines several geometric product simplification rules which are integrated in the built-in Maxima simplifier (see also Listing 1). For example the sum of the inner and outer products of two elements can be immediately combined into the Clifford product:

```
(%i14) a | b + a & b;
```

---

[3]The code is distributed under GNU Lesser General Public License from GitHub (http://dprodanov.github.io/clifford/).

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

function name	functionality
`dotsimpc` (ab)	simplification of dot products
`dotinvsimp` (ab)	simplification of inverses
`powsimp` (ab)	simplification of exponents
`cliffsimpall` (expr)	full simplification of expressions
`cinv` (ab)	Clifford inverse of element
`grade` (expr)	grade decomposition of expressions
`creverse` (ab)	Clifford reverse of product
`cinvolve` (expr)	Clifford involution of expressions
`cconjugate` (expr)	Clifford conjugate of expressions
`cnorm`(expr)	squared norm of expression

$$a \cdot b.$$

In a similar manner, the Jacobi identity can be demonstrated to hold for the outer product:

```
(%i15) a & b & c + b & c & a + c & a & b;
```

$$0.$$

The `clifford` package assigns built-in simplification rules for products of blades (the first three declarations in Listing 1) and powers of blades (the last two declarations).

Upon initialization, the user has to declare the symbol that will represent the orthonormal basis, and the number of basis vectors with positive, negative and degenerate norms. For example, the *quaternion algebra* can be defined conveniently in `clifford` by issuing the command

```
(%i21) clifford(e,0,2);
```

Once the algebra has been initialized, the quaternion multiplication table can be computed by command

```
(%i22) mtable2();
```

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

Maxima supports a regime of command line simplification, which can be overloaded by custom functionality. The `clifford` package defines several simplifying functions, which can be passed as command line options. In the same algebra $C\ell_{0,2}$, we have:

- Simplification of inverses:
  ```
  (%i31) 1/(1+e[1]),cliffsimpall;
  ```

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

- Outer product evaluation:

  ```
  (%i32) e[1]&e[2];
  ```

$$e_1.e_2.$$

  ```
  (%i33) (1+e[1])&(1+e[1]);
  ```

$$0.$$

- Inner product evaluation:

  ```
  (%i34) (1+e[1])|(1+e[1]);
  ```

$$2\,e_1.$$

- Multi-vector inverses

    A general inverse of a quaternion can be computed. We define an input expression in the form $cc = a + b\,e_1 + c\,e_2 + d\,e_1 \cdot e_2$ and compute the output as the last part of the command block:

  ```
  (%i35) block(declare([a,b,c,d],scalar),
         cc:a+b*e[1]+c*e[2]+d*e[1].e[2],dd:cinv(cc));
  ```

$$dd = \frac{a - e_1\,b - e_2\,c - (e_1 \cdot e_2)\,d}{a^2 + b^2 + c^2 + d^2}.$$

    The result can be verified to yield unity after multiplication with the input, expansion and a rational simplification step.

  ```
  (%i36) ev(dd.cc,expand,ratsimp)
  ```

$$1.$$

5. Geometric calculus functionality in clifford

The clifford package also implements symbolical differentiation based on the vector derivative. We shall give a presentation of the Maxwell equations in the geometric algebra $\mathbb{G}^3 = C\ell_{3,0}$. The exposition follows closely the presentation of Chappell et al. [4]. In this presentation we assume $c = \mu_0 = \epsilon_0 = 1$.

Maxwell's equations can be represented in a convenient way using the notation of Clifford/geometric algebra by a single equation :

$$\nabla_{t+r} F \equiv \left(\nabla_r + \frac{\partial}{\partial t}\right) F = J\,, \tag{5.1}$$

where F is the field object (corresponding to the electromagnetic field tensor) and J is the generalized (4-dimensional) current

$$J = \rho - (J_x\,e_1 + J_y\,e_2 + J_z\,e_3)\,, \tag{5.2}$$

using the component representation. The electromagnetic field object is represented as sum of a vector and a bivector as

$$F = E + i\,B, \tag{5.3}$$

where i is the pseudoscalar of the algebra. This field object has the required transformation properties under reflection [4]. In the conventional Heaviside-Gibbs notation, the fields are represented by the vectors $\mathbf{E}$ and $\mathbf{B}$, which change sign under reflection of the coordinate system. On the other hand, a reflection of the coordinate system the electric field is inverted whereas the magnetic field is invariant. Hence, an ambiguity is introduced with the conventional Heaviside vector notation, in that, two quantities with different geometric properties are both represented using the same type of mathematical object.

5.1. **Electrostatic and Magnetostatic problems in** `clifford`.

Example. In an electrostatic or magnetostatic setting the Green's function of the system

$$\nabla_r G = \delta\,(r) \tag{5.4}$$

is

$$G\,(x, y, z) = \frac{e_1\,x + e_2\,y + e_3\,z}{4\pi\,\sqrt{(x^2 + y^2 + z^2)^3}}. \tag{5.5}$$

Direct calculation issuing the commands

```
(%i71) X:e[1]*x+e[2]*y+e[3]*z$
(%i72) G:X/sqrt(-cnorm(X))^3$
(%i73) mvectdiff(G,X)$
```

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

$$V(x, u, z) = -\frac{C}{\sqrt{x^2 + y^2 + z^2}}, \tag{5.6}$$

where C is an arbitrary constant matching the initial or boundary conditions. Applying the multivector derivative to $V(x, u, z)$ yields

```
(%i74) mvectdiff(-1/sqrt(x^2+y^2+z^2),X);
```

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

which can be recognized as a scalar multiple of the Green's function. If we take for example $C = i$ and apply the vector derivative to the potential, $\nabla_r i V$, we get

```
(%i75) mvectdiff(-%iv/sqrt(x^2+y^2+z^2),X);
```

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

which is also a solution. Therefore, the most general expression for the Green's function is

$$G = C_1 \frac{e_1\, x + e_2\, y + e_3\, z}{(x^2 + y^2 + z^2)^{\frac{3}{2}}} + C_2 \frac{(e_2 \cdot e_3)\ x - (e_1 \cdot e_3)\ y + (e_1 \cdot e_2)\ z}{(x^2 + y^2 + z^2)^{\frac{3}{2}}}, \quad (5.7)$$

which can be represented in the form

$$G = \langle G \rangle_1 + \langle G \rangle_2 = E + iB, \tag{5.8}$$

corresponding to an electromagnetic field object.

5.2. Electromagnetic field in `clifford`.

Example. The multivector **F** representing the electromagnetic field is defined in the end of the following command sequence, in which the variables **EE** and **BB** respectively represent the electric and the magnetic vector fields:

```
(%i42) EE:cvect(E, [x,y,z]);
```

$$e_1\, E_x + e_2\, E_y + e_3\, E_z$$

```
(%i43) BB:cvect(B, [x,y,z]);
```

$$e_1\, B_x + e_2\, B_y + e_3\, B_z$$

```
(%i44) F:cliffsimpall(EE + %iv.BB);
```

$$(e_2 \cdot e_3)\ B_x + e_1\, E_x - (e_1 \cdot e_3)\ B_y + e_2\, E_y + (e_1 \cdot e_2)\ B_z + e_3\, E_z\,.$$

The variable `%iv` in `clifford` denotes the pseudoscalar element of $\mathbb{G}^3$. Simplification revealed that **F** is the sum of a vector and a bivector.

The multivector derivative of **F** can be computed further as follows by the following commands:

```
(%i46) r:e[1]*x+e[2]*y+e[3]*z$
(%i47) DF:mvectdiff(FF1, r)$
(%i48) G:grade(DF)$
```

where in the last step, we apply grade decomposition, which reveals a scalar sector:

```
(%i49) G[1];
```

$$\frac{\partial E_x}{\partial\, x} + \frac{\partial E_y}{\partial\, y} + \frac{\partial E_z}{\partial\, z}.$$

A vector in the Euclidean sector can also be identified there as

```
(%i50) G[2];
```

$$e_1\ \left(\frac{\partial B_y}{\partial\, z} - \frac{\partial B_z}{\partial\, y}\right) + e_2\ \left(\frac{\partial B_z}{\partial\, x} - \frac{\partial B_x}{\partial\, z}\right) + e_3\ \left(\frac{\partial B_x}{\partial\, y} - \frac{\partial B_y}{\partial\, x}\right),$$

having the following antisymmetric multiplication table

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

The bivector segment in the field derivative can be identified as

`(%i51) G[3];`

$$(e_1 \cdot e_2) \left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \right) + (e_1 \cdot e_3) \left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} \right) + (e_2 \cdot e_3) \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \right).$$

This segment is isomorphic to the quaternionic algebra, hence it can be referred to as *quaternionic*.

`(%i52) mtable1([e[1] . e[2],e[2] . e[3], e[1] . e[3]]);`

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

Finally, the *pseudoscalar* sector can be identified by

`(%i53) G[4];`

$$(e_1 \cdot e_2 \cdot e_3) \left(\frac{\partial B_x}{\partial x} + \frac{\partial B_y}{\partial y} + \frac{\partial B_z}{\partial z} \right),$$

with a multiplication table

`(%i54) mtable1([%iv]);`

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

In accordance with Gauss's law for magnetism, no magnetic monopoles exist, which implies that this segment is identically zero.

Due to the difficulties of representing Maxwell's equations using quaternions, Heaviside rejected Hamilton's algebraic system and developed a system of vector notations using the scalar (dot) and cross products, which is the conventional system used today in engineering and physics. On the other hand, representation of Maxwell's equations in $\mathbb{G}^3$ reveals the quaternionic symmetries in the vector derivative of the electromagnetic field [9, 14], which are hidden in the conventional vector and tensor notations (e.g, Eq. (2.7)).

5.3. **Homogeneous d'Alembert equation in `clifford`.** Clifford algebras offer a convenient way to combine 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 Lagrangian density $\mathcal{L}(q, \partial_x q)$ involving a field q and its derivatives $\partial_x q$ with respect to the coordinates x are derived in [10], which we replicate in the following example using the `clifford` package.

Example. Let $q = A$ be the paravector potential given by

$$A = A_t + e_1 A_x + e_2 A_y + e_3 A_z, \tag{5.9}$$

which can be constructed by the command

`(%i84) AA: celem(A,[t,x,y,z])$`

Then the geometric derivative object is given by

$$F = \nabla_{t-r} A, \tag{5.10}$$

which is implemented as

`(%i85) F:mvectdiff(AA,t-r)$`

resulting in scalar, vector and bivector parts:

$$F = \langle F \rangle_0 + \langle F \rangle_1 + \langle F \rangle_2 . \tag{5.11}$$

The d'Alembert equation for the paravector potential can be derived from a purely quadratic Lagrangian composed from the components of the geometric derivative

$$\mathcal{L}_a = \frac{1}{2} \langle F^2 \rangle_0 \tag{5.12}$$

`(%i86)  L:lambda([x],1/2*scalarpart(cliffsimpall(x.x)))(F);`

$$((A_{tt})^2 + (A_{tx})^2 + (A_{ty})^2 + (A_{tz})^2 - 2(A_{tx})(A_{xt}) + (A_{xt})^2 - 2(A_{tt})(A_{xx})$$
$$+ (A_{xx})^2 - (A_{xy})^2 - (A_{xz})^2 - 2(A_{ty})(A_{yt}) + (A_{yt})^2$$
$$+ 2(A_{xy})(A_{yx}) - (A_{yx})^2 - 2(A_{tt})(A_{yy})$$
$$+ 2(A_{xx})(A_{yy}) + (A_{yy})^2 - (A_{yz})^2 - 2(A_{tz})(A_{zt}) + (A_{zt})^2$$
$$+ 2(A_{xz})(A_{zx}) - (A_{zx})^2 + 2(A_{yz})(A_{zy}) - (A_{zy})^2 - 2(A_{tt})(A_{zz})$$
$$+ 2(A_{xx})(A_{zz}) + 2(A_{yy})(A_{zz}) + (A_{zz})^2)/2 ,$$

where we can recognize the identity

$$\mathcal{L}_a = \frac{1}{2} \left(\langle F \rangle_0^2 + \langle F \rangle_1^2 + \langle F \rangle_2^2 \right) \tag{5.13}$$

`(%i87)  S:scalarpart(F)$`
`(%i88)  V:vectorpart(F)$`
`(%i89)  Q:grpart(F,2)$`
`(%i90)  L-1/2*(S.S+V.V+Q.Q),cliffsimpall;`

$$0.$$

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

`(%i50) EuLagEq2(L, t+r,[AA,dA]);`

$$(-A_{ttt} + A_{txx} + A_{tyy} + A_{tzz}) + e_1 (-A_{xtt} + A_{xxx} + A_{xyy} + A_{xzz})$$
$$+ e_2 \left(-A_{ytt} + A_{yxx} + A_{yyy} + A_{yzz} \right) + e_3 (-A_{ztt} + A_{zxx} + A_{zyy} + A_{zzz}),$$

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

$$\nabla_{t+r} \nabla_{t-r} A = 0. \tag{5.14}$$

The Maxima expression can be decomposed in a matrix form as

`(%i51)  bdecompose(%);`

$$[[[1], \left(-A_{ttt} + A_{txx} + A_{tyy} + A_{tzz}\right)],$$

$$[[e_1, e_2, e_3], \begin{pmatrix} -A_{xtt} + A_{xxx} + A_{xyy} + A_{xzz} \\ -A_{ytt} + A_{yxx} + A_{yyy} + A_{yzz} \\ -A_{ztt} + A_{zxx} + A_{zyy} + A_{zzz} \end{pmatrix}],$$

$$[[0], (0)], [[0], (0)]]]$$

if one wishes to solve for individual components.

5.4. **Euler-Lagrange treatment of electromagnetism using `clifford`.**

Example. Finally, we demonstrate the capabilities of the Clifford package by deriving basic identities and the field equations of electromagnetism from Lagrangian functionals. We begin with loading and setting up the package (see Listing 3 for details), establishing the appropriate Clifford algebra, declaring dependencies and defining the coordinates. Next, we define an 4-vector field A.

To demonstrate the gauge invariance of electromagnetism, we add to this field the 4-gradient of an arbitrary twice differentiable scalar field f, which shall vanish from all the field equations. We then define the electromagnetic field tensor F, which is the exterior derivative of A. We compute the vector derivative of A, which shall be used later when computing the Euler-Lagrange equation:

`(%i10)  a:celem(A,[t,x,y,z])+mvectdiff(f,t+r);`

$$f_t + e_1\,f_x + e_2\,f_y + e_3\,f_z + A_t + e_1\,A_x + e_2\,A_y + e_3\,A_z$$

`(%i11)  dA:mvectdiff(a,t-r);`

$$f_{tt} - f_{xx} - f_{yy} - f_{zz} + A_{tt} + e_1\left(-A_{tx} + A_{xt}\right) - A_{xx} + e_1 \cdot e_2 A_{xy}$$
$$+ e_1 \cdot e_3 A_{xz} + e_2\left(-A_{ty} + A_{yt}\right) - e_1 \cdot e_2\,A_{y_x} - A_{yy} + e_2 \cdot e_3 A_{y_z}$$
$$+ e_3\left(-A_{tz} + A_{zt}\right) - e_1 \cdot e_3\,A_{zx} - e_2 \cdot e_3 A_{zy} - A_{zz}$$

`(%i12)  F:lambda([x],x-scalarpart(x)-grpart(x,4))(mvectdiff(a,t-r));`

$$A_{tt} + e_1\left(-A_{tx} + A_{xt}\right) - A_{xx} + e_2\left(-A_{ty} + A_{y_t}\right)$$
$$- (e_1.e_2)\left(-A_{xy} + A_{y_x}\right) - A_{yy} + e_3\left(-A_{tz} + A_{zt}\right)$$
$$- (e_1.e_3)\left(-A_{xz} + A_{zx}\right) - (e_2.e_3)\left(-A_{y_z} + A_{zy}\right) - A_{zz}.$$

We can now construct a standard EM Lagrangian, $\mathcal{L}_{EM} = -\frac{1}{2}\left\langle F\,F^{\star}\right\rangle_0$:

`(%i13)  L:lambda([x],1/2*scalarpart(cliffsimpall(-x.cconjugate(x))))(F);`

$$\begin{aligned}
&((A_{tx})^2 + (A_{ty})^2 + (A_{tz})^2 - 2\,(A_{tx})\,(A_{xt}) + (A_{xt})^2 - (A_{xy})^2 - (A_{xz})^2 \\
&- 2\,(A_{ty})\,(A_{y_t}) + (A_{y_t})^2 + 2\,(A_{xy})\,(A_{y_x}) - (A_{y_x})^2 - (A_{y_z})^2 \\
&- 2\,(A_{tz})\,(A_{zt}) + (A_{zt})^2 + 2\,(A_{xz})\,(A_{zx}) - (A_{zx})^2 \\
&+ 2\,(A_{y_z})\,(A_{zy}) - (A_{zy})^2)/\,2.
\end{aligned}$$

Next, using the vector part of A, we construct the electric and magnetic vector fields E and B, as well as the current density j and charge density q:

```
(%i14) b:vectorpart(a);
```

$$e_1\,f_x + e_2\,f_y + e_3\,f_z + e_1\,A_x + e_2\,A_y + e_3\,A_z$$

```
(%i15) B:factorby(cliffsimpall(-%iv.vvectdiff(b,r)),
       makelist(asymbol[i],i,1,ndim));
```

$$e_3\left(-A_{xy} + A_{y_x}\right) - e_2\left(-A_{xz} + A_{zx}\right) + e_1\left(-A_{y_z} + A_{zy}\right)$$

```
(%i16) E:factorby(diff(b,t)-mvectdiff(scalarpart(a),r),
       makelist(asymbol[i],i,1,ndim));
```

$$e_1\left(-A_{tx} + A_{xt}\right) + e_2\left(-A_{ty} + A_{y_t}\right) + e_3\left(-A_{tz} + A_{zt}\right)$$

```
(%i17) j:factorby(cliffsimpall(diff(E,t)-vvectdiff(%iv.B,r)),
       makelist(asymbol[i],i,1,ndim));
```

$$\begin{aligned}
&e_1\left(-A_{ttx} + A_{xtt} - A_{xyy} - A_{xzz} + A_{y_{xy}} + A_{zxz}\right) \\
&+ e_2\left(-A_{tty} + A_{xxy} + A_{y_{tt}} - A_{y_{xx}} - A_{y_{zz}} + A_{zyz}\right) \\
&+ e_3\left(-A_{ttz} + A_{xxz} + A_{y_{yz}} + A_{ztt} - A_{zxx} - A_{zyy}\right)
\end{aligned}$$

```
(%i18) q:svectdiff(E,r);
```

$$-A_{txx} - A_{tyy} - A_{tzz} + A_{xtx} + A_{y_{ty}} + A_{ztz}.$$

Finally, we can obtain the field equations. First, we verify the conservation of charge:

```
(%i19) svectdiff(q+j,t-r);
```

$$0.$$

Next comes Faraday's law:

```
(%i20) cliffsimpall(-%iv.vvectdiff(E,r)-diff(B,t));
```

$$0.$$

Next, Gauss's law for magnetism:

```
(%i21) svectdiff(B,r);
```

$$0.$$

We now check that the postulated Lagrangian is, in fact, identical to the conventional $\frac{1}{2}(E^2 - B^2)$ form:

```
(%i22) 1/2*(E.E-B.B)-L,expand;
```

$$0.$$

The Euler-Lagrange equation is identically satisfied:

```
(%i23) EL:EuLagEq2(L,t+r,[a,dA]);
```

$$0.$$

```
(%i24) bdecompose(EL);
```

$$[[[1], \left(A_{txx} + A_{tyy} + A_{tzz} - A_{xtx} - A_{yty} - A_{ztz}\right)],$$

$$[[e_1, e_2, e_3], \begin{pmatrix} A_{ttx} - A_{xtt} + A_{xyy} + A_{xzz} - A_{yxy} - A_{zxz} \\ A_{tty} - A_{xxy} - A_{ytt} + A_{yxx} + A_{yzz} - A_{zyz} \\ A_{ttz} - A_{xxz} - A_{yyz} - A_{ztt} + A_{zxx} + A_{zyy} \end{pmatrix}],$$

$$[[0], \left(0\right)], [[0], \left(0\right)]]$$

```
(%i25) EL+q+j;
```

$$0.$$

Finally, we modify the Lagrangian, introducing an external current J_0. Note that gauge invariance is lost in the presence of the external current, and the Lagrangian must be constructed accordingly:

```
(%i26) dependsv(J,[t,x,y,z])$
(%i27) A0:celem(A,[t,x,y,z])$
(%i28) J0:celem(J,[t,x,y,z])$
(%i29) EL:EuLagEq2(L+scalarpart(A0.J0),t+r,[a,dA])$
(%i30) bdecompose(EL);
```

$$[[[1], \left(J_t + A_{txx} + A_{tyy} + A_{tzz} - A_{xtx} - A_{yty} - A_{ztz}\right)],$$

$$[[e_1, e_2, e_3], \begin{pmatrix} A_{ttx} + J_x - A_{xtt} + A_{xyy} + A_{xzz} - A_{yxy} - A_{zxz} \\ A_{tty} - A_{xxy} + J_y - A_{ytt} + A_{yxx} + A_{yzz} - A_{zyz} \\ A_{ttz} - A_{xxz} - A_{yyz} + J_z - A_{ztt} + A_{zxx} + A_{zyy} \end{pmatrix}],$$

$$[[0], \left(0\right)], [[0], \left(0\right)]]$$

```
(%i31) EL+q+j-J0,expand;
```

$$0.$$

This demonstration shows the power of geometric algebra methods when applied to a field theory of physics, and the utility of the `clifford` and `cliffordan` packages to the theoretical physicist.

6. Discussion

The package `itensor` and the related package `ctensor` allow for advanced symbolical calculation in field theory and general relativity. Both packages provide complementary functionality that can be used in translating problems from geometric to tensor calculus and vice versa.

The elements of a Clifford algebra over an n-dimensional space live in the 2^n-dimensional space of the power-set $P(V)$. Naive implementations of the algebraic structures, therefore, quickly lose performance [8]. Other implementations can use isomorphic matrices over the reals or the complex numbers, which also have a great amount of redundancy [7]. In contrast, sparse representations, as inherently present in Maxima and Lisp, allow for efficient representation of the elements of the algebra. Another advantage is the possibility to perform complex symbolic calculations, for example the full Clifford simplification following (multi)vector differentiation.

The `clifford` package allows for implementation of functions following closely the mathematical notation used in Clifford and geometric algebra applications. This allows for implementation of new algorithms following closely their mathematical representation. Future developments will also incorporate solutions to variational problems, such as derivation of Euler-Lagrange equations from field Lagrangians.

6.1. **Comparison with other packages.** Maxima is a general purpose computer algebra system with capabilities that are similar to the two mainstream commercial CAS: Maple and Mathematica. Unsurprisingly, third-party contributions that implement Clifford, Grassmann and geometric algebra capabilities exist for both products.

For Maple, we note the work the package `CLIFFORD` [1], in development since the late 1990's. The `CLIFFORD` package was motivated by work on octonions. The functionality is limited to 9 dimensions. Another package, `BIGEBRA` [2], builds on `CLIFFORD`, with the aim to explore Hopf algebras and provide a useful tool for "experimental mathematics".

For Mathematica, the `clifford.m` package [3] introduces the concept of Clifford and Grassmann algebras, multivectors, and the geometric product. The blades are represented by tuples of numbers. Some graphical capabilities are also provided. The authors express their hope that their package, like our Maxima solution, will find its utility in the hands of physicists.

Our Maxima package is also designed to be a useful tool in the hands of applied mathematicians and physicists. Our package emphasizes simplification, including the ability to treat multivectors as sparse objects (i.e., perform algebraic simplification on expressions containing previously not defined symbols representing multivector objects; see Sec. 4.1.)

Authors' contributions. DP is the author of the `clifford` and `cliffordan` packages. VTT is the author of the Maxima version of the `atensor` package, as well as principal maintainer of `ctensor` and `itensor` for nearly 15 years.

The packages `atensor`, `ctensor` and `itensor` are distributed with Maxima. The `clifford` and `cliffordan` packages are freely available for download from GitHub[4]. The code of the latter packages is distributed under GNU Lesser General Public License (LGPL).

Acknowledgments

References

[1] R. Ablamowicz and B. Fauser. Mathematics of CLIFFORD - A Maple package for Clifford and Grassmann algebras. *ArXiv Mathematical Physics e-prints*, December 2002.

[2] R. Abłamowicz and B. Fauser. Clifford and Graßmann Hopf algebras via the BIGE-BRA package for Maple. *Computer Physics Communications*, 170:115–130, August 2005.

[3] G. Aragon-Camarasa, G. Aragon-Gonzalez, J. L. Aragon, and M. A. Rodriguez-Andrade. Clifford Algebra with Mathematica. In I. J. Rudas, editor, *Recent Advances in Applied Mathematics*, volume 56 of *Mathematics and Computers in Science and Engineering Series*, pages 64–73, Budapest, 12–14 Dec 2015. Proceedings of AMATH '15, WSEAS Press.

[4] J.M. Chappell, S.P. Drake, C.L. Seidel, L.J. Gunn, A. Iqbal, A. Allison, and D. Abbott. Geometric algebra for electrical and electronic engineers. *Proceedings of the IEEE*, 102(9):1340–1363, 2014.

[5] C. Doran and A. Lasenby. *Geometric Algebra for Physicists.* Cambridge University Press, 2003.

[6] L. Dorst. Honing geometric algebra for its use in the computer sciences. In G. Sommer, editor, *Geomteric Computing with Clifford Algebras.* Springer, 2001.

[7] L. Dorst, D. Fontijne, and S. Mann. *Geometric Algebra for Computer Science. An Object-oriented Approach to Geometry.* Elsevier, Amsterdam, 2007.

[8] D. Fontijne. *Efficient Implementation of Geometric Algebra.* PhD thesis, University of Amsterdam, 2007.

[9] A. Gsponer and J.-P. Hurni. The physical heritage of Sir W. R. Hamilton. Presented at the Conference: "The Mathematical Heritage of Sir William Rowan Hamilton" commemorating the sesquicentennial of the invention of quaternions. Trinity College, Dublin, 17th – 20th August, 1993. *ArXiv*, math-ph(0201058), January 2002.

[10] A. Lasenby, C. Doran, and S. Gull. A multivector derivative approach to lagrangian field theory. *Foundations of Physics*, 23(10):1295–1327, 1993.

[11] Maxima Project, http://maxima.sourceforge.net/docs/manual/maxima.html. *Maxima 5.35.1 Manual*, Dec 2014.

[12] I. Porteus. *Clifford Algebras and the Classical Groups.* Cambridge University Press, Cambridge, 2 edition, 2000.

[13] V. Toth. Tensor manipulation in GPL Maxima. *ArXiv*, cs(0503073), 2007.

[14] K.J. van Vlaenderen and A. Waser. Generalisation of classical electrodynamics to admit a scalar field and longitudinal waves. *Hadronic Journal*, pages 609 – 628, 2001.

Appendix: Example Code listings

---

[4]`http://dprodanov.github.io/clifford/`

Listing 2: Tensor calculus.

```
load(itensor);
imetric(g);
components(F([m,n],[]),A([n],[],m)-A([m],[],n));
L:ishow(-1/4*F([k,l])*F([a,b],[])*g([],[k,a])*g([],[l,b])
+j([k],[])*A([l],[])*g([],[k,l]))$
remcomps(F);
decsym(F,0,2,[],[anti(all)]);
ishow(diff(L,A([m],[],n)))$
ishow(subst(F([a,b],[])+A([a],[],b),A([b],[],a),%))$
ishow(subst(F([k,l],[])+A([k],[],l),A([l],[],k),%))$
ishow(canform(rename(contract(expand(%)))))$
ishow(contract(diff(L,A([m],[])))-idiff(%,n)=0)$
```

Listing 3: Geometric calculus.

```
/* Initialization */
load("clifford.mac")$
load("cliffordan.mac")$
derivabbrev:true$
%divsimp:true$
clifford(e,3)$
declare([t,x,y,z,f], scalar)$
dependsv(A,[t,x,y,z])$
depends(f,[x,y,z,t])$
r:cvect([x,y,z]);
/* The 4-potential with gauge freedom */
a:celem(A,[t,x,y,z])+mvectdiff(f,t+r);
dA:mvectdiff(a,t-r);
/* The electromagnetic field */
F:lambda([x],x-scalarpart(x)-grpart(x,4))(mvectdiff(a,t-r));
/* The Maxwell Lagrangian */
L:lambda([x],1/2*scalarpart(cliffsimpall(-x.cconjugate(x))))(F);
/* spatial field variable */
b:vectorpart(a);
/* The magnetic field */
B:factorby(cliffsimpall(-%iv.vvectdiff(b,r)),makelist(asymbol[i],i,1,ndim));
/* The electric field */
E:factorby(diff(b,t)-mvectdiff(scalarpart(a),r),makelist(asymbol[i],i,1,ndim));
/* The current density and charge density, which are conserved */
j:factorby(cliffsimpall(diff(E,t)-vvectdiff(%iv.B,r)),makelist(asymbol[i],i,1,ndim));
q:svectdiff(E,r);
svectdiff(q+j,t-r);
/* Maxwell's equations */
cliffsimpall(-%iv.vvectdiff(E,r)-diff(B,t));
svectdiff(B,r);
/* Verifying the Lagrangian */
1/2*(E.E-B.B)-L,expand;
/* The Euler-Lagrange equation */
EL:EuLagEq2(L,t+r,[a,dA]);
bdecompose(EL);
EL+(q+j);
/* Introducing an external current */
dependsv(J,[t,x,y,z]);
A0:celem(A,[t,x,y,z]);
J0:celem(J,[t,x,y,z]);
EL:EuLagEq2(L+scalarpart(A0.J0),t+r,[a,dA]);
bdecompose(EL);
EL+q+j-J0,expand;
```

DEPARTMENT OF ENVIRONMENT, HEALTH AND SAFETY, NEUROSCIENCE RESEARCH FLANDERS, IMEC vzw, LEUVEN, BELGIUM
E-mail address: dimiterpp@gmail.com;   dimiter.prodanov@imec.be

CENTER FOR RESEARCH ON INTEGRATED SENSORS PLATFORMS CARLETON UNIVERSITY OTTAWA, ONTARIO, CANADA
E-mail address: vttoth@vttoth.com