AN EXAMPLE OF CLIFFORD ALGEBRAS CALCULATIONS WITH GiNaC

VLADIMIR V. KISIL

Abstract. This is an example of C++ code of Clifford algebra calculations with the GiNaC computer algebra system. This code makes both symbolic and numeric computations. It was used to produce illustrations for paper [14, 12]. Described features of GiNaC are already available at PyGiNaC and due to course should propagate into other software like GNU Octave and gTybalt which use GiNaC library as their back-end.

Contents

1. Introduction 1
2. Main procedure 3
3. Auxiliary matter
   3.1. Defines 4
   3.2. Variables 5
4. Symbolic Clifford Algebra Calculations
   4.1. Initialisation of Clifford Numbers 5
   4.2. Möbius Transformations 9
   4.3. Symbolic Calculations of the Vector Fields 10
5. Numeric Calculations with Clifford Algebras
   5.1. Numeric Calculations of Orbits and Transverses 12
   5.2. Building of Transverses 13
   5.3. Future-to-Past Transformations 14
   5.4. Single Node Calculation 15
   5.5. Cayley Transforms of Images 17
   5.6. Numeric Check of Formulae 18
6. How to Get the Code 21
Appendix A. Textual Output of the Program 21
Appendix B. A Sample of Graphics Generated by the Program 22
Appendix C. Index of Identifiers 23
References 24

1. Introduction

This example of Clifford algebras calculations uses GiNaC library [1], which includes a support for generic Clifford algebra starting from version 1.3.0. Both symbolic and numeric calculation are possible and can be blended with other functions of GiNaC. Described features of GiNaC are already available at PyGiNaC and due to course should propagate into other software like GNU Octave and gTybalt which use GiNaC library as their back-end.

We bind our C++-code with documentation using noweb within the literate programming concept [17]. Our program makes output of two types: some results are typed on screen for information only and the majority of calculated data are stored in files which are lately incorporate by MetaPost to produce PostScript graphics for the paper [14, 12]. Since this code can be treated as software we are pleased to acknowledge that it is subject to GNU General Public License [10].

GiNaC allows to use a generic Clifford algebra, i.e., $2^n$ dimensional algebra with generators $e_k$ satisfying the identities $e_i e_j + e_j e_i = B(i,j) + B(j,i)$ for some (metric) $B(i,j)$, which may be non-symmetric [8, 9] and contain symbolic entries. Such generators are created by the function

ex clifford_unit(const ex & mu, const ex & metr, unsigned char rl = 0, bool anticommuting = false);

where mu should be varidx class object indexing the generators, an index mu with a numeric value may be of type idx as well. Parameter metr defines the metric $B(i,j)$ and can be represented by a square matrix, tensormetric or indexed class object, optional parameter rl allows to distinguish different Clifford algebras (which will commute each other). The last optional parameter anticommuting defines if the anticommuting assumption (i.e. $e_i e_j + e_j e_i = 0$)
will be used for contraction of Clifford units. If the metric is supplied by a matrix object, then the value of anticommuting is calculated automatically and the supplied one will be ignored. One can overcome this by giving metric through matrix wrapped into an indexed object.

Note that the call CliffordUnit(mu, minkmetric()) creates something very close to DiracGamma(mu), although DiracGamma have more efficient simplification mechanism. The method Clifford::getMetric() returns metric defining this Clifford number. The method Clifford::isAnticommuting() returns the anticommuting property of a unit.

If the matrix \( B(i,j) \) is in fact symmetric you may prefer to create the Clifford algebra units with a call like that

```cpp
ex e = CliffordUnit(mu, Indexed(B, sym(m), i, j));
```

since this may yield some further automatic simplifications. Again, for a metric defined through a matrix such a symmetry is detected automatically.

Individual generators of a Clifford algebra can be accessed in several ways. For example

```cpp
{...
    varidx nu(symbol("nu"), 4);
    realsymbol s("s");
    ex M = diag_matrix(lst(1, -1, 0, s));
    ex e = CliffordUnit(nu, M);
    ex e0 = e.subs(nu == 0);
    ex e1 = e.subs(nu == 1);
    ex e2 = e.subs(nu == 2);
    ex e3 = e.subs(nu == 3);
    ...
}
```

will produce four generators of a Clifford algebra with properties \( e_0^2 = 1 \), \( e_1^2 = -1 \), \( e_2^2 = 0 \) and \( e_3^2 = s \).

A similar effect can be achieved from the function

```cpp
ex lst_to_clifford(const ex & v, const ex & mu, const ex & metr, unsigned char rl = 0,
                   bool anticommuting = false);
```

which converts a list or vector \( v = (v_0, v_1, \ldots, v_n) \) into the Clifford number \( v_0e_0 + v_1e_1 + \cdots + v_ne_n \) with \( e_k \) directly supplied in the second form of the procedure. In the first form the Clifford unit \( e_k \) is generated by CliffordUnit(mu, metr, rl, anticommuting). The previous code may be rewritten with help of lst_to_clifford() as follows

```cpp
{...
    varidx nu(symbol("nu"), 4);
    realsymbol s("s");
    ex M = diag_matrix(lst(1, -1, 0, s));
    ex e0 = lst_to_clifford(lst(1, 0, 0, 0), nu, M);
    ex e1 = lst_to_clifford(lst(0, 1, 0, 0), nu, M);
    ex e2 = lst_to_clifford(lst(0, 0, 1, 0), nu, M);
    ex e3 = lst_to_clifford(lst(0, 0, 0, 1), nu, M);
    ...
}
```

There is the inverse function

```cpp
lst clifford_to_lst(const ex & e, const ex & c, bool algebraic=true);
```

which took an expression \( e \) and tries to find such a list \( v = (v_0, v_1, \ldots, v_n) \) that \( e = v_0e_0 + v_1e_1 + \cdots + v_ne_n \) with respect to given Clifford units \( c \) and none of \( v_k \) contains the Clifford units \( c \) (of course, this may be impossible). This function can use an algebraic method (default) or a symbolic one. In algebraic method \( v_k \) are calculated as \( (e c_k + e_k c)/pow(e_k, 2) \). If \( pow(c_k, 2) \) is zero or is not numeric for some \( k \) then the method will be automatically changed to symbolic. The same effect is obtained by the assignment (algebraic=false) in the procedure call.

There are several functions for (anti-)automorphisms of Clifford algebras:

```cpp
ex clifford_prime(const ex & e)
inline ex clifford_star(const ex & e) { return e.conjugate(); }
inline ex clifford_bar(const ex & e) { return clifford_prime(e.conjugate()); }
```

The automorphism of a Clifford algebra clifford_prime() simply changes signs of all Clifford units in the expression. The reversion of a Clifford algebra clifford_star() coincides with conjugate() method and effectively reverses the order of Clifford units in any product. Finally the main anti-automorphism of a Clifford algebra clifford_bar() is the composition of two previous, i.e. makes the reversion and changes signs of all Clifford units in a product. Names for this functions corresponds to notations \( e' \), \( e^* \) and \( \bar{e} \) used in Clifford algebra textbooks [3][4][5].
The function
\texttt{ex\ clifford\_norm(const \texttt{ex} & \texttt{e});}
calculates the norm of Clifford number from the expression \(|e|^2 = e\bar{e}\). The inverse of a Clifford expression is returned by the function
\texttt{ex\ clifford\_inverse(const \texttt{ex} & \texttt{e});}
which calculates it as \(e^{-1} = e/|e|^2\). If \(|e| = 0\) then an exception is raised.

If a Clifford number happens to be a factor of \texttt{dirac\_ONE()} then we can convert it to a “real” (non-Clifford) expression by the function
\texttt{ex\ remove\_dirac\_ONE(const \texttt{ex} & \texttt{e});}
The function \texttt{canonicalize\_clifford()} works for a generic Clifford algebra in a similar way as for Dirac gammas.

The last provided function is
\texttt{ex\ clifford\_moebius\_map(const \texttt{ex} & \texttt{a}, \texttt{const \texttt{ex} & \texttt{b}, \texttt{const \texttt{ex} & \texttt{c}}, \texttt{const \texttt{ex} & \texttt{d}}, \texttt{const \texttt{ex} & \texttt{v}}, \texttt{\texttt{const \texttt{ex} & \texttt{G}}, \texttt{unsigned char} \texttt{rl} = 0, \texttt{\texttt{bool}} \texttt{anticommuting} = \texttt{false});}
\texttt{ex\ clifford\_moebius\_map(const \texttt{ex} & \texttt{M}, \texttt{const \texttt{ex} & \texttt{v}}, \texttt{\texttt{const \texttt{ex} & \texttt{G}}, \texttt{unsigned char} \texttt{rl} = 0, \texttt{\texttt{bool}} \texttt{anticommuting} = \texttt{false});}
It takes a list or vector \(v\) and makes the Möbius (conformal or linear-fractional) transformation \(v \mapsto (av + b)(cv + d)^{-1}\) defined by the matrix \(M = \begin{pmatrix} a & b \\
 c & d \end{pmatrix}\).

The matrix may be given in two different forms—as one entity or by its four elements. The last parameter \(G\) defines the metric of the surrounding (pseudo-)Euclidean space. This can be an indexed object, tensormetric, matrix or a Clifford unit, in the later case the optional parameters \(rl\) and \(anticommuting\) are ignored even if supplied. The returned value of this function is a list of components of the resulting vector.

Finally the function
\texttt{char\ clifford\_max\_label(const \texttt{ex} & \texttt{e}, \texttt{bool} \texttt{ignore\_ONE} = \texttt{false});}
can detect a presence of Clifford objects in the expression \(e\): if such objects are found it returns the maximal \texttt{representation\_label} of them, otherwise -1. The optional parameter \texttt{ignore\_ONE} indicates if \texttt{dirac\_ONE} objects should be ignored during the search.

\LaTeX output for Clifford units looks like \texttt{\clifford[1]{e}^\nu}, where 1 is the \texttt{representation\_label} and \texttt{\nu} is the index of the corresponding unit. This provides a flexible typesetting with a suitable definition of the \texttt{\clifford} command. For example, the definition
\texttt{\newcommand{\clifford}[1][]{}},
typesets all Clifford units identically, while the alternative definition
\texttt{\newcommand{\clifford}[2][]\{\ifcase #1 \or #2\or \tilde{#2} \or \breve{#2} \or \bar{#2} \fi\}},
prints units with \texttt{representation\_label}=0 as \(e\), with \texttt{representation\_label}=1 as \(\tilde{e}\) and with \texttt{representation\_label}=2 as \(\breve{e}\).

2. Main procedure

Here is the main procedure, which has a very straightforward structure. This and next initialisation section is pretty standard. The first usage of \texttt{GiNaC} for Clifford algebras is in Section 3.
Now we run a cycle over the three possible type of metric in two dimensional space (i.e. elliptic, parabolic and hyperbolic). For each space we initialise the corresponding Clifford units, symbolically calculate various types of M"obius transforms as well as vector fields for three subgroups of $SL_2(\mathbb{R})$.

Uses elliptic $\bigcirc$, hyperbolic $\bigcirc$, and metric $\bigcirc$.

Then we run a cycle for three subgroups of $SL_2(\mathbb{R})$ (i.e. $A$, $N$, $K$). For all possible combinations of those with metric from the surrounding cycle in the previous chunk we

1. build orbits of the subgroups and their transversal curves;
2. two types of the Cayley transform images of all above curves;
3. check some formulae in the paper;

We draw all pictures by substitution of numeric values into the symbolic results obtained in the above chunks.

Uses subgroup $\bigcirc$, subgroup $\bigcirc$, and subgroup $\bigcirc$.

Finally we draw eight frames which illustrates the continuous transformation of the future part of the light cone into the its past part [12, Figure 4].

3. Auxiliary matter

Some standard inclusions, but do not forget GiNaC library!

`<ginac/ginac.h>` // At least ver. 1.4.0!
`<cmath>`

using namespace std;
using namespace GiNaC;
3.1. Defines. Some constants are defined here for a better readability of the code.

\[
\text{\#define elliptic 0} \\
\text{\#define parabolic 1} \\
\text{\#define hyperbolic 2} \\
\text{\#define subgroup}_A 0 \\
\text{\#define subgroup}_K 2 \\
\text{\#define subgroup}_N 1 \\
\text{\#define grey 0.6}
\]

Defines:
- \text{elliptic}, used in chunks 4a, 10, and 18b.
- \text{grey}, used in chunk 12b.
- \text{hyperbolic}, used in chunks 4a, 6a, 10, 14–16, and 18d.
- \text{parabolic}, used in chunks 10b and 17–19.
- \text{subgroup}_A, used in chunks 4b, 11, 15c, 16c, and 20.
- \text{subgroup}_K, used in chunks 4b and 11–19.
- \text{subgroup}_N, used in chunk 16.

Some macro definitions which we use to make more compact code. They initialise variables, open and close curve description in the MetaPost file. Here is initialisation of a new curve.

\[
\text{\#define init\_coord(X) upos}[X] = 0; \\
\text{\#define close\_curve(X) fprintf(fileout}[X], "(a%+5.3f,b%+5.3f)*u withcolor %5.3f*%s; n", } \\
\text{\#define put\_draw(X) fprintf(fileout}[X], "ndraw ")}
\]

Defines:
- \text{init\_coord}, used in chunks 4 and 5.
- \text{close\_curve}, used in chunks 4, 5a, and 5c.

Uses \text{fileout}, and \text{upos}.

This part is used to close a curve output in cases the end is reached or curves passes the infinity.

\[
\text{\#define put\_point(X) if (inversion) fprintf(fileout}[X], "(a%+5.3f,b%+5.3f)*u...", upos}[X], \text{vpos}[X], \text{color\_grade, color\_name}[\text{subgroup}] } \\
\text{\#define renew\_curve(Y) close\_curve(Y); put\_draw(Y)}
\]

Defines:
- \text{put\_point}, used in chunk 5d.
- \text{renew\_curve}, used in chunk 6.

Uses \text{close\_curve}, \text{fileout}, \text{subgroup}, \text{color\_name}, \text{u}, and \text{upos}.

Here is the common part of code which is used for outputs a segment to files.
We should make a rough check that the curve is still in the bounded area, if it cross infinity then such line should be discontinued and started from a new. Our bound are few times bigger that the real picture, the excellent cutting within the desired limits is done by MetaPost itself with the \texttt{clip currentpicture to ...;} command.

Besides some outer margins we put different types of bound depending from the nature of objects: sometimes it is limited to the upper half plane, sometimes to hyperbolic unit disk. The necessity of such checks in the hyperbolic case is explained in \cite{12, § 2.5}.

\texttt{\#define if in limits(X) if ( (abs(u_res.to_double()) \leq ulim) \&\& (abs(v_res.to_double()) \leq vlim) \&\& (metric \neq hyperbolic) \lor inversion \lor (cayley \land \texttt{ex to numeric}((-pow(u_res,2)+pow(v_res,2)-1.001).is_positive()))) \{ \\ upos[X] = u_res.to_double(); \\ vpos[X] = v_res.to_double(); \\ ex Vect = dV[subgroup][X].subs(lst(x \equiv u_res, y \equiv v_res)); \\ udir[X] = \texttt{ex to numeric}((Vect.op(0)).to_double(); \\ vdir[X] = \texttt{ex to numeric}((Vect.op(1)).to_double(); \\ put_point(X); \\ } else { \\ renew_curve(X); \\ }}

Defines:
\texttt{if in limits}, used in chunks 15–17.

Uses \texttt{hyperbolic, metric, numeric, put point, renew curve, subgroup, ulim, and upos}.

Then a curve is going through infinity we catch the exception, close the corresponding \texttt{draw} statement of MetaPost and start a new one from the next point.

\texttt{\#define catch handle(X) cerr << "*** Got problem: " << p.what() << endl; \\ renew_curve(X)}

Defines:
\texttt{catch handle}, used in chunks 15–17.

Uses \texttt{renew curve}.

Extracting of numerical values out of Moebius transformations

\texttt{\#define get components u_res = \texttt{ex to numeric}(res.op(0).evalf()); \\ v_res = \texttt{ex to numeric}(res.op(1).evalf());}

Defines:
\texttt{get components}, used in chunks 15–17.

Uses \texttt{numeric}.
To make an accurate drawing we calculate the direction of a transverse line out of symbolic vector fields calculation done before.

\[
\text{\#define transverse\_dir}(X) \text{ if (!direct) \{ \}
\]
\[
trans\_uf = \text{ex\_to<numeric>}(\text{trans\_dir\_sub}[X].\text{op}(0).\text{subs(a\_node).evalf()}).\text{to\_double}();
\]
\[
trans\_uf = \text{ex\_to<numeric>}(\text{trans\_dir\_sub}[X].\text{op}(1).\text{subs(a\_node).evalf()}).\text{to\_double}();
\]
\[
\text{if (trans\_uf \equiv INFINITY) \{ trans\_uf = 1; trans\_vf = 0; \}} \}
\]
\[
\text{else if (trans\_uf \equiv -INFINITY) \{ trans\_uf = -1; trans\_vf = 0; \}} \}
\]
\[
\text{else if (trans\_vf \equiv INFINITY) \{ trans\_uf = 0; trans\_vf = 1; \}} \}
\]
\[
\text{else if (trans\_vf \equiv -INFINITY) \{ trans\_uf = 0; trans\_vf = -1; \}} \}
\]
\[
\text{else if (abs(trans\_uf) + abs(trans\_vf) > 100) \{ \}
\]
\[
\text{double } r = \text{sqrt}(\text{trans\_uf} \ast \text{trans\_uf} + \text{trans\_vf} \ast \text{trans\_vf});
\]
\[
\text{trans\_uf } \div = r; \text{ trans\_vf } \div = r; \}
\}
\]

Defines:

\text{transverse\_dir}, used in chunks 16 and 17.

Uses \text{numeric} 8d.

Some global variables which is convenient to use for the \text{openfile()} function below.

\[
\text{FILE} \ast \text{openfile}(\text{const char} \ast F) \{
\]
\[
\text{char } \text{filename}[]="\text{cayley-t-k-e.d}", \text{temp}[]="\text{cayley-t-%.1s-%.1s.d}";
\]
\[
\text{char } \ast \text{filename} = \text{filename}, \ast \text{Stempl} = \text{templ};
\]
\[
\text{strcat}()\text{strcpy}(\text{Stempl}, F), "-%.1s-%.1s.d")
\]
\[
\text{sprintf}()\text{fopen}(\text{Sfilename}, \text{Stempl}, \&\text{sgroup}[\text{subgroup}], \&\text{metric}\_name[\text{metric}]);
\]
\[
\text{return fopen}(\text{Sfilename}, \"w\")
\}
\]

Defines:

\text{openfile}, used in chunks 12 and 13d.

Uses \text{numeric} 8d, \text{metric} 7b, \text{sgroup} 7b, and \text{subgroup} 7b.
3.2. Variables. First we define variables from the standard C++ classes.

```c++
FILE *fileout[3]; /* files to pass results to \MetaPost */
static char *color_name[] = {"hyp", "par", "ell", "white"}, /* \ A, N, K subgroup colours */
    formula[] = {"\nDistance to center is:",
                 "\nDirectrice is:",
                 "\nDifference to foci is:"};
bool direct = true, /* Is it orbit or transverse? */
cayley = false, /* Is it the Cayley transform image? */
inversion = false; /* Is it future-past inversion? */
double u, v, upos[3], vpos[3], udir[3], vdir[3], vval = 0; /* coordinates of point, vector, etc */
color grade, focal f[2] = {0, 0},
trans uf =1, trans uf =0;
```

Defines:
- `color_name`, used in chunks 5c and 12b.
- `fileout`, used in chunks 5, 12, 13, and 15a.
- `u`, used in chunks 5, 12b, and 19c.
- `upos`, used in chunks 5 and 6a.
- `v`, used in chunks 8, 12, and 19c.

Then other variables of GiNaC types are defined as well. They are needed for numeric and symbolic calculations.

```c++
varidx nu(symbol("nu", "\nu"), 2),
    mu(symbol("mu", "\mu"), 2),
    psi(symbol("psi", "\psi"),2),
    xi(symbol("xi", "\xi"), 2);
realsymbol x("x"),
y("y"), t("t"), // for symbolic calculations
tr u("U"),
tr v("V"); // Vector of the transverse direction
lst a_node, sob[2], a_trans;
```

Uses subgroup 7b.

```c++
matrix M(2, 2), // The metric of the vector space
C(2, 2), CI(2, 2), ICI(2, 2), // Two versions of the Cayley transform
T(2, 2), TI(2, 2), // The map from first to second Cayley transform
Jacob[3][3] = {matrix(2,2)), // Jacobian of the Moebius transformation
trans_dir[3][3] = {matrix(2,1)); // Components of transverse direction
trans_dir_sub[3] = {matrix(2,1)}; // Components of transverse direction substituted by a subgroup
```

Uses metric 7d and subgroup 7b.

```c++
numeric u_res, v_res, // coordinates of the Moebius transform
up[3][2] = {0, 0, 0, 0, 0, 0}; // saved values of coordinates of the parabola
vp[3][2] = {0, 0, 0, 0, 0, 0};
const numeric half(1, 2);
```

Defines:
- `numeric`, used in chunks 5, 6, 18–20.

```c++
ex res, e0, eI,
    Moebius[3][5], dV[3][3],
    // indexed by [subgroup], [type](=direct, Cayley-operator, Cayley1-op, C-point, C1-p)
ddV[3], Curv[3], // indexed by [type](=direct, Cayley-operator, Cayley1-op)
focal, p,
    focalL, focalL, focalW; // parameters of parabola given by v = au^2 + bu + c
```

Uses subgroup 7f and v 8a.
Here is the set of constants which allows to fine tune MetaPost output depending from the type of subgroup and metric used. The quality of pictures will significantly depend from the number of points chosen for iterations: too much lines will mess up the picture, very few makes it incomplete or insensitive to the singular regions. These numbers should be different for different combinations of subgroup and metric.

\[
\begin{align*}
\text{(Pictures tuning)} & \equiv \\
\text{const int vilimits}[3][3] & = \{10, 20, 30, \ \text{// indexed by } [\text{subgroup}], [\text{metric}] \\
& \quad 10, 10, 19, \\
& \quad 10, 10, 10\}, \\
\text{fsteps}[3][3] & = \{15, 15, 20, \ \text{// indexed by } [\text{subgroup}], [\text{metric}] \\
& \quad 15, 10, 20, \\
& \quad 12, 15, 15\}; \\
\text{float ulim} & = 25, \ \text{vlim} = 25, \\
\text{flimits}[3][3] & = \{2.0, 2.0, 4.0, \ \text{// indexed by } [\text{subgroup}], [\text{metric}] \\
& \quad 10.0, 4.0, 4.0, \\
& \quad 0.5, 0.5, 0.5\}, \\
\text{vpoints}[3][10] & = \{0, 1.0 \div 8, 1.0 \div 4, 1.0 \div 2, 1.0, 2.0, 3.0, 6.0, 10.0, 20.0, \\
& \quad 0.1, 1.0 \div 8, 1.0 \div 4, 1.0 \div 2, 1.0, 2.0, 3.0, 5.0, 10.0, 100\};
\end{align*}
\]

Defines:
- \text{ulim}, used in chunks 9a and 44.
- \text{vilimits}, used in chunks 13–16.

Uses \text{metric} 7b and \text{subgroup} 7b.

Initialise the set of coordinates for a cycle

\[
\begin{align*}
\text{(Initialisation of coordinates)} & \equiv \\
\text{init\_coord}(0); \\
\text{init\_coord}(1); \\
\text{put\_draw}(2);
\end{align*}
\]

Uses \text{init\_coord} 13a and \text{put\_draw} 14.

4. Symbolic Clifford Algebra Calculations

This section finally starts to deal with Clifford algebras. We try to make all possible calculations symbolically delaying the numeric substitution to the latest stage. This produces a faster code as well.

4.1. Initialisation of Clifford Numbers. We initialise Clifford numbers first. Alternative ways to define \(e, e0\) and \(e1\) are indicated in comments.

\[
\begin{align*}
\text{(Initialise Clifford numbers)} & \equiv \\
\text{signum} & = \text{numeric}(\text{metric}-1); \ \text{// the value of } e_2^2 \\
M & = -1, 0, \\
& \quad 0, \text{signum}; \\
& \quad /e = \text{clifford\_unit}(\mu, \text{indexed}(M, \text{symmetric2}(), \text{xi}, \psi)); \\
& \quad e = \text{clifford\_unit}(\mu, M); \\
& \quad e0 = e.\text{subs}(\mu \equiv 0); \\
& \quad e1 = e.\text{subs}(\mu \equiv 1); \\
& \quad /e0 = \text{lst\_to\_clifford}(\text{lst}(1.0, 0), \mu, M); \\
& \quad /e1 = \text{lst\_to\_clifford}(\text{lst}(0, 1.0), \mu, M);
\end{align*}
\]

Uses \text{metric} 7b and \text{numeric} 8d.

Now we define matrices used for definition of the alternative Cayley transforms 14 and 12.

\[
\begin{align*}
\text{(Initialise Clifford numbers)} & \equiv \\
T & = \text{dirac\_ONE}(), e0, \ \text{// Transformation to the alternative Cayley map} \\
& \quad e0, \text{dirac\_ONE}(); \ \text{// is given by} \begin{pmatrix} 1 & e_1 \\ e_1 & 1 \end{pmatrix} \\
TI & = \text{dirac\_ONE}(), -e0, \ \text{// The inverse of } T \\
& \quad -e0, \text{dirac\_ONE}(); \ \text{// is given by} \begin{pmatrix} 1 & -e_1 \\ -e_1 & 1 \end{pmatrix}
\end{align*}
\]
A form of matrix for the Cayley transform depends from the type of metric.

\[
\langle \text{Initialise Clifford numbers} \rangle \equiv
\]

\[
\text{switch (metric) } \{
\text{case elliptic:}
\text{case hyperbolic:}
\]

\[
C = \text{dirac\_ONE}(), -e1, \quad \text{// First Cayley transform } \begin{pmatrix} 1 & -e_2 \\ \sigma e_2 & 1 \end{pmatrix}
\]

\[
signum*e1, \text{dirac\_ONE}();
\]

\[
CI = \text{dirac\_ONE}(), e1, \quad \text{// The inverse of } C \begin{pmatrix} 1 & e_2 \\ -\sigma e_2 & 1 \end{pmatrix}
\]

\[
-\text{signum}*e1, \text{dirac\_ONE}();
\]

\[
C1 = C.\text{mul}(T); \quad \text{// Second Cayley transform}
\]

\[
C1I = CI.\text{mul}(C1); \quad \text{// The inverse of } C1
\]

\[
\text{break;}
\]

Uses elliptic 5a, hyperbolic 5a, and metric 7b.

In the parabolic case there are two different (elliptic and hyperbolic) types of the Cayley transform [14, 12].

\[
\langle \text{Initialise Clifford numbers} \rangle \equiv
\]

\[
\text{case parabolic:}
\]

\[
C = \text{dirac\_ONE}(), -e1*half, \quad \text{// First (elliptic) Cayley transform for the parabolic case}
\]

\[
-e1*half, \text{dirac\_ONE}();
\]

\[
CI = \text{dirac\_ONE}(), e1*half, \quad \text{// The inverse of } C
\]

\[
e1*half, \text{dirac\_ONE}();
\]

\[
C1 = \text{dirac\_ONE}(), -e1*half, \quad \text{// Second (hyperbolic) Cayley transform for the parabolic case}
\]

\[
e1*half, \text{dirac\_ONE}();
\]

\[
C1I = CI*\text{mul}(C1); \quad \text{// The inverse of } C1
\]

\[
-\text{e1*half}, \text{dirac\_ONE}();
\]

\[
\text{break;}
\]

Uses elliptic 5a, hyperbolic 5a, and parabolic 5a.

4.2. Möbius Transformations. We calculate all Moebius transformations along the orbits as well as two their Cayley transforms only once in a symbolic way. Their usage will be made through the GiNaC substitution mechanism. First, we define matrices \( \text{Exp}_A, \text{Exp}_N, \text{Exp}_K \) [15, VI.1] related to the Iwasawa decomposition of \( SL_2(\mathbb{R}) \) [13 § III.1].

\[
\langle \text{Calculation of Moebius transformations} \rangle \equiv
\]

\[
\text{matrix } \text{Exp}_A(2, 2), \text{Exp}_N(2, 2), \text{Exp}_K(2, 2);
\]

\[
\text{Exp}_A = \exp(t) * \text{dirac\_ONE}(), 0, \quad \text{// Matrix } \begin{pmatrix} e^t & 0 \\ 0 & e^{-t} \end{pmatrix} = \exp \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
\]

\[
0, \exp(-t) * \text{dirac\_ONE}();
\]

\[
\text{Exp}_N = \text{dirac\_ONE}(), t * e0, \quad \text{// Matrix } \begin{pmatrix} 0 & t \\ 0 & 0 \end{pmatrix} = \exp \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}
\]

\[
0, \text{dirac\_ONE}();
\]

\[
\text{Exp}_K = \cos(t) * \text{dirac\_ONE}(), \sin(t) * e0, \quad \text{// Matrix } \begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix} = \exp \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}
\]

\[
\sin(t) * e0, \cos(t) * \text{dirac\_ONE}();
\]

\[
\text{ex Exp[3] = \{Exp}_A, \text{Exp}_N, \text{Exp}_K\};
\]

\[
\text{matrix } E(2, 2);
\]
Here we symbolically calculate the related Möbius transformations, as well as its two images under Cayley transform C and C1 for operators and finally two Cayley images for points.

We calculate symbolic expressions for the vector fields of subgroups = subgroup_A, subgroup_K and subgroup_K_K.

4.3. Symbolic Calculations of the Vector Fields. We calculate symbolic expressions for the vector fields of three subgroups A, N and K, which are stated in [12] Lemmas 2.1 and 2.2 and shown on [12, Figures 1 and 2]. The formula for a derived representation \( \rho(X) \) of a vector field \( X \) used here is [13, § VI.1]:

\[
\rho(X) = \frac{d}{dt}\rho(e^{tX}) \mid t=0.
\]

Results of calculations are directed to stdout and will be used for a better drawing of orbits, see 3.1.

We calculate Jacobian of the Möbius transformations in order to supply to MetaPost the tangents of the transverse lines.

```plaintext
try {
    cout << "Vect field \t Direct \t In Cayley \t In Cayley1" << endl;
    for (subgroup = subgroup_A; subgroup ≤ subgroup_K; subgroup++) {
        for (int i=0; i < 3; i++) {
            lst Moeb = ex_to_lst(Moebius[subgroup][cayley?cayley+2:0]);
            dV[subgroup][cayley] = Moebius[subgroup][cayley].diff(t).subs(t = 0);
            Jacob[subgroup][cayley] = matrix[2, 2, lst(Moeb.op(0).diff(x), Moeb.op(0).diff(y),
                                                  Moeb.op(1).diff(x), Moeb.op(1).diff(y)));
            // Transformation of a direction by a Jacobian
            for (int i=0; i < 2; i++)
                trans_dir[subgroup][cayley] = Jacob[subgroup][cayley].mul(matrix[2, 1, lst(tr_u, tr_v)));
        }
    }
}
```
We also calculate curvature of the orbits using the formula [5.1(20)]

\[
\kappa = \frac{|\ddot{x}y - \ddot{y}x|}{(x^2 + y^2)^{3/2}}
\]

5. Numeric Calculations with Clifford Algebras

Numeric calculations are done in two fashions:

1. Through a substitution of numeric values to symbols in some previous symbolic results, subsection 5.1 and 5.5;
2. Direct calculations with numeric \(\text{GiNaC}\) (8a, and 8d).

The first example of substitution approach is the drawing of vector fields. Three vector fields are drawn by arrows into a MetaPost file.

5.1. Numeric Calculations of Orbits and Transverses. For any of three possibility \(c_2^2 = -1, 0, 1\) and three possible subgroups \((A, N, K)\) orbits are constructed. First we output MetaPost files.
This chunk runs iterations over the different orbits, which are initiated by the point \( v_i \).

```c
13a
for (int vi = 0; vi < vilimits[subgroup][metric]; vi++) { // iterator over orbits
  color_grade = 1.2*vi/vilimits[subgroup][metric];
  if (subgroup == subgroup_K)
    cout \ll formula[metric] ;
    (Initialisation of coordinates 9b)
    (Nodes iterations 13b)
    (Close all curves 14b)
    (Check parabolas 19d)
  }
(Closing all files 13c)
``` 

Uses metric 7b, subgroup 7b, subgroup_K 6a, and vilimits 5a.

Each orbit is processed by iteration over the “time” parameter \( j \) on the orbit. For each node on an orbit an entry is put into appropriate MetaPost file, formulae from papers are numerically checked and all Cayley transforms are produced.

```c
13b
for (int j = -fsteps[subgroup][metric]; j <= fsteps[subgroup][metric]; j++) {
  float f = flimits[subgroup][metric]*j/fsteps[subgroup][metric]; // the angle of rotation
  (Generating one entry 15c)
  (Check formulas in the paper 18a)
  (Producing Cayley transform of the orbit 17b)
}
``` 

Uses metric 7b and subgroup 7b.

Closing all files when finishing drawing orbits or transverses

```c
13c
fclose(fileout[0]);
fclose(fileout[1]);
fclose(fileout[2]);
``` 

Uses fileout 8a.

5.2. Building of Transverses. Construction of transverses to the orbits follows the same structure as for orbits themselves with the changed order of iterations over time parameter and orbit origins. We again start from opening of the corresponding files.

```c
13d
direct = false;
fileout[0] = openfile("orbit-t");
fileout[1] = openfile("cayley-t"); // Cayley transform of transverses
fileout[2] = openfile("cayl-a-t"); // Alternative Cayley transform of transverses
color_grade = 1.2;
(Define transverse directions 16c)
``` 

Uses fileout 6a and openfile 7.
Thus chunk performs iterations over the transverse lines.

(Building transverses 13d) +≡

for (int j = -fsteps[subgroup][metric]; j ≤ fsteps[subgroup][metric]; j++) {
    float f = flimits[subgroup][metric] + j*fsteps[subgroup][metric]; // the angle of rotation
    (Initialisation of coordinates 9b)
    for (int vi = 0; vi < vilimits[subgroup][metric]; vi++) {
        // iterator over orbits
        vval = vpoints[metric][vi];
        (Generating one entry 15c)
        (Producing Cayley transform of the orbit 17b)
    }
    (Close all curves 14b)
}
(Closing all files 13c)

Uses metric 7b, subgroup 7b, and vilimits 9a.
All MetaPost draw statements should be closed at the end of run.

(Close all curves 14b) ≡

const int curves = 15, nodes = 40, frames = 8;
const float exp_scale = 1.3, node_scale = 4.0,
    rad[] = {1.0/5, 1.0/4, 1/3.5, 1/2.5, 1/2, 1/1.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5};
char name[] = "future-past-00.d";
char* S = name;
ulim = 8.5;
vlim = 8.5;

metric = hyperbolic;
subgroup = subgroup_K;
direct = true;
inversion = true; // to cut around unit circle.
(Initialise Clifford numbers 9c)
ex Fut = clifford_moebius_map(dirac_ONE(), -a * e1,
    a * e1, dirac_ONE(), lst(x, y), M);

Defines:
    curves, used in chunk 15a
    exp_scale, used in chunk 15b
    name, never used
    S, used in chunk 15a
Uses hyperbolic 5a, metric 7b, subgroup 7b, subgroup_K 9a, and ulim 9a.
Then we proceed with iteration over the frames.

\[ \langle \text{Build future-past transition} \rangle \equiv \]
\[
\text{for } (\text{int } j = 0; j < \text{frames}; j++) \{ // the power of transformation
\text{sprintf} (S, "future-past-%d.d", j);
\text{fileout}[0] = fopen (S, "w");
\text{for } (\text{int } k = 0; k < \text{curves}; k++) \{ // the number of curve
\text{color_grade} = k/\text{frames};
\text{init_coord}(0);
\} // the power of transformation
\text{close_curve}(0);
\}
\]

\[ \text{fclose} (\text{fileout}[0]); \]

\[ \text{cout} \ll \text{endl}; \]

\text{Uses close_curve, curves, fileout, init_coord, and S. }

A curve is created by rotation in hyperbolic metric and then a Möbius transformations corresponding to the frame number is applied.

\[ \langle \text{Iteration over a curve} \rangle \equiv \]
\[
\text{for } (\text{int } l = -\text{nodes}/2; l \leq \text{nodes}/2; l++) // the node number
\text{try} l // There is a chance of singularity!
\text{float} \text{angl} = ((j > 0) ? \text{exp}(\text{double}(j/\text{exp_scale}-3)) : 0);
\text{res} = \text{Fut}.\text{subs}(\text{lst}(a \equiv \text{angl}, x \equiv \text{rad}[k]*\text{cosh}(l/\text{node_scale}), y \equiv \text{rad}[k]*\text{sinh}(l/\text{node_scale}));
\text{get_components;}
\text{if in_limits}(0);
\} \text{catch} (\text{exception } & \text{p}) \{
\text{catch_handle}(0);
\}
\]

\[ \text{Uses catch, catch_handle, exp_scale, get_components, and if in_limits. } \]

5.4. Single Node Calculation. This is a common portion of code for building orbits and transverses. A single entry into \textit{MetaPost} file is calculated and written. Calculations depend for three possible values of the subgroup. For each of them we create a special list \texttt{a_node} of substitutions for the already symbolically calculated Möbius[].

For subgroup \texttt{A} the points are distributed evenly on the unit circle.

\[ \langle \text{Generating one entry} \rangle \equiv \]
\[
\text{switch} (\text{subgroup}) \{
\text{case subgroup}_A:
\text{vval} = 1.0*\text{vi}/(\text{vilimits}[_\text{subgroup}][\text{metric}]-1);
\text{if} (\text{metric} \equiv \text{hyperbolic}) // we need a double set of value for negatives as well
\text{vval} *= 2;
\text{a_node} = \text{lst}(t \equiv f, x \equiv \text{cos}(\text{Pi*vval}), y \equiv \text{sin}(\text{Pi*vval}));
\text{break};
\}
\]

\[ \text{Uses hyperbolic, metric, subgroup, and vilimits.}\]

For the subgroups \texttt{K} and \texttt{N} points are distributed evenly on the vertical axis.

\[ \langle \text{Generating one entry} \rangle \equiv \]
\[
\text{case subgroup}_K:
\text{vval} = \text{vpoinroes}[\text{metric}][\text{vi}];
\text{a_node} = \text{lst}(t \equiv (f * \text{Pi}), x \equiv 0, y \equiv \text{vval});
\text{break};
\]

\[ \text{Uses metric and subgroup}_K.\]
For the subgroup $N$ we need additional points.

\[ \langle \text{Generating one entry } \rangle + \equiv (13b 14a) \triangleright \]

\text{case subgroup } N:\]
\[ \text{if (metric } \equiv \text{ hyperbolic } \rangle \langle / we need a double set of value for negatives as well}
\vval = ( (vi - vlimits[subgroup][metric] \div 2) < 0 ) ? -1 : 1
* vpoints[metric][abs(vi - vlimits[subgroup][metric] \div 2)];
\]
\[ \text{else}
\vval = vpoints[metric][vi];
\]
\[ a\_node = \text{lst}(t \equiv f, x \equiv 0, y \equiv \vval);
\text{break;}
\]

\text{Uses hyperbolic 5a, metric 7b, subgroup 7b, and vilimits 9a.}

And now using values stored above in $a\_node$ we do the actual calculation through substitution and write the node into the MetaPost file.

\[ \langle \text{Generating one entry } \rangle + \equiv (13b 14a) \triangleright \]

\text{try}
\[ \text{res} = \text{Moebius}[subgroup][0].\text{subs}(a\_node);
\text{transverse}\_\text{dir}(0);
\text{get}\_\text{components};
\text{if}\_\text{in}\_\text{limits}(0);
\text{catch} (\text{exception } &p) \{
\text{catch}\_\text{handle}(0);
\}
\]

\text{Defines:}
\text{catch, used in chunks 11a, 15b, 19a, and 20.}
\text{Uses catch\_handle 6b, get\_components 6c, if\_in\_limits 6a, subgroup 7b, and transverse\_dir 7a.}

We define the tangents to the transverse lines here.

\[ \langle \text{Define transverse directions } \rangle \equiv (13d) \triangleright \]

\text{switch (subgroup) } \{
\text{case subgroup } A:\]
\[ a\_\text{trans} = \text{lst}(tr\_\alpha \equiv -y, tr\_\beta \equiv x);
\text{break;}
\text{case subgroup } N:\]
\text{case subgroup } K:\]
\[ a\_\text{trans} = \text{lst}(tr\_\alpha \equiv 0, tr\_\beta \equiv 1);
\text{break;}
\}

\text{Uses subgroup 7b, subgroup}_A 5a, subgroup_K 14a, and subgroup_K 7a.}

And here again comes evaluation through substitution.

\[ \langle \text{Define transverse directions } \rangle + \equiv (13d) \triangleright \]

\text{for (int cal=0; cal < 3; cal++)}
\[ \text{trans}\_\text{dir}\_\text{sub}[cal] = \text{matrix}(2, 1,
\text{lst}(\text{trans}\_\text{dir}[subgroup][cal].op(0).\text{subs}(a\_\text{trans}).\text{normal}()),
\text{trans}\_\text{dir}[subgroup][cal].op(1).\text{subs}(a\_\text{trans}).\text{normal}()));
\]

\text{Uses subgroup 7b.}
5.5. Cayley Transforms of Images. We will need a calculation parameters of parabola into the Cayley transform images. This checks the statement \cite[Lemma 2.17]{Kisil2008}. The calculation is done by linear equation solver \texttt{lsolve()} from \texttt{GiNaC}.

\begin{verbatim}
#define calc_par_focal(X) if (direct && (metric == parabolic) \\
    && (subgroup \neq subgroup_K)) { \\
    up[2][X] = u_res; \\
    vp[2][X] = v_res; \\
    if (j \equiv 1) { \\
        lst eqns vars; \\
        vars = a, b, c; \\
        eqns = \texttt{a*pow(up[0][X], 2) + b*up[0][X] + c} \equiv \texttt{vp[0][X]}, \\
            \texttt{a*pow(up[1][X], 2) + b*up[1][X] + c} \equiv \texttt{vp[1][X]}, \\
            \texttt{a*pow(up[2][X], 2) + b*up[2][X] + c} \equiv \texttt{vp[2][X];} \\
        soln[X] = \texttt{ex_to<lst>(lsolve(eqns, vars));} \\
    } \\
/* After calculation is made we store previous values for the next round. */ \\
    up[0][X] = up[1][X]; \\
    vp[0][X] = vp[1][X]; \\
    up[1][X] = up[2][X]; \\
    vp[1][X] = vp[2][X]; \\
} \\
\end{verbatim}

Defines:
\texttt{calc_par_focal}, used in chunk 17.

Uses \texttt{metric}, \texttt{parabolic}, \texttt{subgroup}, and \texttt{subgroup_K}.

We produce two versions of the Cayley transforms for each node of the orbit or transverses lines. This done by simple substitution of \texttt{a_node} into \texttt{Moebius[[3,4]]} symbolically calculated in subsection 4.2.

\begin{verbatim}
try { // There is a chance of singularity! \\
    res = Moebius[subgroup][3].subs(a_node); \\
    transverse_dir(1); \\
    get_components; \\
    if_in_limits(1); \\
    calc_par_focal(0); \\
} catch (exception &p) { \\
    catch_handle(1); \\
} \\
\end{verbatim}

Defines:
\texttt{catch}, used in chunks 11a, 15b, 19a, and 20.

Uses \texttt{calc_par_focal}, \texttt{catch_handle}, \texttt{get_components}, \texttt{if_in_limits}, \texttt{subgroup}, and \texttt{transverse_dir}.

For second type of the Cayley transforms we perform an extra run similar to the above.

\begin{verbatim}
try { \\
    res = Moebius[subgroup][4].subs(a_node); \\
    transverse_dir(2); \\
    get_components; \\
    if_in_limits(2); \\
    calc_par_focal(1); \\
} catch (exception &p) { \\
    catch_handle(2); \\
} \\
\end{verbatim}

\texttt{cayley = false;}

Defines:
\texttt{catch}, used in chunks 11a, 15b, 19a, and 20.

Uses \texttt{calc_par_focal}, \texttt{catch_handle}, \texttt{get_components}, \texttt{if_in_limits}, \texttt{subgroup}, and \texttt{transverse_dir}.
5.6. Numeric Check of Formulae. Here is a numeric check of few formulas in the paper about radius and focal length of sections. We calculate focal properties for three types of orbits (circles, parabolas and hyperbolas) of the subgroup \( K \).

\( \langle \text{Check formulas in the paper} \rangle^a \equiv \langle \text{Check formulas in the paper} \rangle^b \equiv \langle \text{Check formulas in the paper} \rangle^c \equiv \langle \text{Check formulas in the paper} \rangle^d \equiv \langle \text{Check formulas in the paper} \rangle^e \equiv \langle \text{Check formulas in the paper} \rangle^f \equiv \langle \text{Check formulas in the paper} \rangle^g \equiv \langle \text{Check formulas in the paper} \rangle^h \equiv \langle \text{Check formulas in the paper} \rangle^i \equiv \langle \text{Check formulas in the paper} \rangle^j \equiv \langle \text{Check formulas in the paper} \rangle^k \equiv \langle \text{Check formulas in the paper} \rangle^l \)

Here is a numeric check of few formulas in the paper about radius and focal length of sections. We calculate focal properties for three types of orbits (circles, parabolas and hyperbolas) of the subgroup \( K \).

\[
\text{if } ((j \neq \text{fsteps[subgroup[metric]]) \land (j \neq \text{fsteps[subgroup[metric]) // End points are weird!}}
\]

\[
\text{(subgroup \( \equiv \text{subgroup[metric]} \) \land (vval \neq 0)) } \text{only for that values}
\]

\[
\text{try } \{
\text{switch (metric) } \{
\text{depends from the type of metric}
\}
\}
\]

The values are calculated for each node on the orbit as follows, see [12, Lemma 2.2]. For elliptic orbits of subgroup \( K \): a circle with the centre at \((0, \frac{(v + v^{-1})}{2})\) and the radius \((v - v^{-1})/2\).

\[
\text{case elliptic:}
\]

\[
focal_f[1] = \text{ex_to<numeric>>(pow(u_res*u_res + pow(v_res-(vval+1/vval)÷2, 2), 0.5)).to_double();}
\]

\[
\text{break;}
\]

Uses metric 7b, subgroup 7b, and subgroup \( K \).

For parabolic orbits of subgroup \( K \): a parabola with the focus at \((0, \frac{(v + v^{-1})}{2})\) and focal length \(v^{-1}/2\).

\[
\text{case parabolic:}
\]

\[
focal_f[1] = \text{ex_to<numeric>>(pow(u_res*u_res + pow(v_res-(vval+1/vval÷4), 2), 0.5)-v_res).to_double();}
\]

\[
\text{break;}
\]

Uses numeric 8d and parabolic 5a.

For hyperbolic orbits of subgroup \( K \): a hyperbola with the upper focus located at \((0, f)\) with:

\[
f = \begin{cases} 
    p - \frac{p^2}{2} - 1, & \text{for } 0 < v < 1; \text{ and} \\
    p + \frac{p^2}{2} - 1, & \text{for } v \geq 1.
\end{cases}
\]

and has the focal distance between focuses \(2p\).

\[
\text{case hyperbolic:}
\]

\[
p = (vval*\text{vval+1})\div\text{vval+pow(2,0.5)};
\]

\[
focal = ((\text{vval}<1) \text{ ? p -pow(p*p÷2-1,0.5) : p+pow(p*p÷2-1,0.5))};
\]

\[
focal_f[1] = \text{ex_to<numeric>>(pow(u_res*u_res + pow(v_res-focal, 2), 0.5)}
\]

\[
-\text{pow(u_res*u_res + pow(v_res-focal+2*p, 2), 0.5)).to_double();}
\]

\[
\text{break;}
\]

Uses hyperbolic 5a and numeric 8d.
If the obtained value is reasonably close to the previous one then = sign is printed to stdout, otherwise the new value is printed. This produce lines similar to the following:

**Distance to center is:** 3.938======================

Remark 5.1. Note that all check are passed smoothly (see Appendix A), however in the hyperbolic case there is “V” shape of switch from positive values to negative and back (with the same absolute value) like this:

**Difference to foci is:** 2.000====== -2.000============== 2.000======

This demonstrates the non-invariance of the upper half plane in the hyperbolic case as explained in [12, § 2.5].

Uses catch 12a 16b 17b 17c.

We last check we make is about some properties of Cayley transform in parabolic case. All parameters of parabolic orbits were calculated in Subsection 5.5, now we check properties listed in [12, Lemma 2.17].

Here is expressions for focal length \( f_{\text{ocal}l} \) and vertex \((f_{\text{ocal}u}, f_{\text{ocal}v})\) of a parabola given by its equation \( v = au^2 + bu + c \).

\[
\begin{align*}
\text{focal}_l &= 1/(4a); /\!\!/ \text{ focal length} \\
\text{focal}_u &= b/(2a); /\!\!/ \text{ u comp of focus} \\
\text{focal}_v &= c-\text{pow}(b/(2a), 2); /\!\!/ \text{ v comp of focus}
\end{align*}
\]

Properties of parabolas are printed to stdout in the form:

**Parab** (A/ 7/ 0.368); **vert**=( 1.140, -2.299); **l**= 0.2500; second **vert**=(-1.140, -2.299); **l**=-0.2500

The two vertexes correspond to two Cayley transformations \( P_e \) defined by \( C \) and \( P_h \) defined by \( C_1 \), see [12, § 2.6].

Uses metric 7b, output_focal 19b, parabolic 5a, sgroup 7b, subgroup 7b, and subgroup_K 5a.

Uses catch 12a 16b 17b 17c.

The last check we make is about some properties of Cayley transform in parabolic case. All parameters of parabolic orbits were calculated in Subsection 5.5, now we check properties listed in [12, Lemma 2.17].
In the case of `subgroup_A` an additional line

**Check vertices:** -1 and -1

is printed. It confirms that vertexes of the orbits under the Cayley transform do belong to the parabolas \( v = \pm v^2 - 1 \), as stated in [12, 2.17].

```cpp
if (subgroup == subgroup_A)
    cout << "\nCheck vertices: "
    << ex_to<numeric>(focal_v.subs(soln[0]) + pow(focal_u.subs(soln[0]), 2).evalf()).to_double() << " and "
    << ex_to<numeric>(focal_v.subs(soln[1]) + pow(focal_u.subs(soln[1]), 2).evalf()).to_double();
```

Uses catch 12a 16b 17b 17c, numeric 8d, sgrou[7b, subgroup 7b, and subgroup_A 5a.}
6. How to Get the Code

(1) Get the LaTeX source of this paper [13] from the arXiv.org.

(2) Run the source through \LaTeX. Three new files (noweb, C++ and MetaPost sources) will be created in the current directory.

(3) Use it on your own risk under the GNU General Public License [14].

Appendix A. Textual Output of the Program

Here is the complete textual output of the program.

Metric is: e.

\begin{verbatim}
Vect field Direct In Cayley In Cayley1
\text{dA is: } 2x,2y; -2y*x,1-y^2+x^2; -1-y^2*x^2,2*y*x
\text{dN is: } 1,0; 1/2-y+1/2*y^2-1/2*x^2,-y*x+x; -y*x-y,1/2+x-1/2*y^2+1/2*x^2
\text{dK is: } 1-y^2*x^2,2*y*x; -2*y,2*x; -2*y,2*x
\text{Curvature of K-orbits on the v-axis: } (1+y^4-2*y^2)^{-1.5}*(-2*y^5-2*y+4*y^3)
\end{verbatim}

Distance to center is:

\begin{verbatim}
Distance to center is: 3.938
Distance to center is: 1.875
Distance to center is: 0.750
Distance to center is: 0.000
Distance to center is: 0.750
Distance to center is: 1.333
Distance to center is: 2.400
Distance to center is: 3.938
Distance to center is: 7.969
\end{verbatim}

Metric is: p.

\begin{verbatim}
Vect field Direct In Cayley In Cayley1
\text{dA is: } 2x,2y; 2*x,2+2*y+2*x^2; 2*x,2+2*y-2*x^2
\text{dN is: } 1,0; 1,2*x; 1,-2*x
\text{dK is: } 1+x^2,2*y*x; 1+x^2,2*y*x+4*x; 1+x^2,2*y*x
\text{Curvature of K-orbits on the v-axis: } -2*y
\end{verbatim}

Parab (A/ 0/ 0.000); vert=( 0.000, -1.000); l= 0.2500; second vert=( 0.000, -1.000); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 1/ 0.053); vert=( 0.083, -1.007); l= 0.2500; second vert=(-0.083, -1.007); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 2/ 0.105); vert=( 0.172, -1.029); l= 0.2500; second vert=(-0.172, -1.029); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 3/ 0.158); vert=( 0.271, -1.073); l= 0.2500; second vert=(-0.271, -1.073); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 4/ 0.211); vert=( 0.389, -1.151); l= 0.2500; second vert=(-0.389, -1.151); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 5/ 0.263); vert=( 0.543, -1.295); l= 0.2500; second vert=(-0.543, -1.295); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 6/ 0.316); vert=( 0.765, -1.586); l= 0.2500; second vert=( 0.765, -1.586); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 7/ 0.368); vert=( 1.140, -2.299); l= 0.2500; second vert=( 1.140, -2.299); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 8/ 0.421); vert=( 1.974, -4.898); l= 0.2500; second vert=( 1.974, -4.898); l=-0.2500
Check vertices: -1 and -1
Parab (A/ 9/ 0.474); vert=( 6.034, -37.410); l= 0.2500; second vert=( 6.034, -37.410); l=-0.2500
Check vertices: -1 and -1
Parab (A/10/ 0.526); vert=(-6.034, -37.410); l= 0.2500; second vert=(-6.034, -37.410); l=-0.2500
Check vertices: -1 and -1
Parab (A/11/ 0.579); vert=(-1.974, -4.898); l= 0.2500; second vert=(-1.974, -4.898); l=-0.2500
\end{verbatim}
Check vertices: -1 and -1
Parab (A/12/ 0.632); vert=(-1.140, -2.299); l= 0.2500; second vert=( 1.140, -2.299); l=-0.2500
Check vertices: -1 and -1
Parab (A/13/ 0.684); vert=(-0.765, -1.586); l= 0.2500; second vert=( 0.765, -1.586); l=-0.2500
Check vertices: -1 and -1
Parab (A/14/ 0.737); vert=(-0.543, -1.295); l= 0.2500; second vert=( 0.543, -1.295); l=-0.2500
Check vertices: -1 and -1
Parab (A/15/ 0.789); vert=(-0.389, -1.151); l= 0.2500; second vert=( 0.389, -1.151); l=-0.2500
Check vertices: -1 and -1
Parab (A/16/ 0.842); vert=(-0.271, -1.073); l= 0.2500; second vert=( 0.271, -1.073); l=-0.2500
Check vertices: -1 and -1
Parab (A/17/ 0.895); vert=(-0.172, -1.029); l= 0.2500; second vert=( 0.172, -1.029); l=-0.2500
Check vertices: -1 and -1
Parab (A/18/ 0.947); vert=(-0.083, -1.007); l= 0.2500; second vert=( 0.083, -1.007); l=-0.2500
Check vertices: -1 and -1
Parab (A/19/ 1.000); vert=(-0.000, -1.000); l= 0.2500; second vert=( 0.000, -1.000); l=-0.2500
Check vertices: -1 and -1
Parab (N/ 0/ 0.000); vert=( 0.000, -1.000); l= 0.2500; second vert=( 0.000, -1.000); l=-0.2500
Parab (N/ 1/ 0.125); vert=( 0.000, -0.875); l= 0.2500; second vert=( 0.000, -0.875); l=-0.2500
Parab (N/ 2/ 0.250); vert=( 0.000, -0.750); l= 0.2500; second vert=( 0.000, -0.750); l=-0.2500
Parab (N/ 3/ 0.500); vert=( 0.000, -0.500); l= 0.2500; second vert=( 0.000, -0.500); l=-0.2500
Parab (N/ 4/ 1.000); vert=( 0.000, 0.000); l= 0.2500; second vert=( 0.000, 0.000); l=-0.2500
Parab (N/ 5/ 2.000); vert=( 0.000, 1.000); l= 0.2500; second vert=( 0.000, 1.000); l=-0.2500
Parab (N/ 6/ 3.000); vert=( 0.000, 2.000); l= 0.2500; second vert=( 0.000, 2.000); l=-0.2500
Parab (N/ 7/ 6.000); vert=( 0.000, 5.000); l= 0.2500; second vert=( 0.000, 5.000); l=-0.2500
Parab (N/ 8/ 10.000); vert=( 0.000, 9.000); l= 0.2500; second vert=( 0.000, 9.000); l=-0.2500
Parab (N/ 9/ 20.000); vert=( 0.000, 19.000); l= 0.2500; second vert=( 0.000, 19.000); l=-0.2500

Directrice is:

Directrice is: 1.875
Directrice is: 0.750
Directrice is: 0.000
Directrice is: -0.750
Directrice is: -1.875
Directrice is: -2.917
Directrice is: -5.958
Directrice is: -9.975
Directrice is: -19.987

Metric is: h.

Vect field Direct In Cayley In Cayley1
dA is: 2*x,2*y; -2*y*x,1-y^2-x^2; 2*y,2*x
dN is: 1,0; 1/2-y+1/2*y^2+1/2*x^2,y*x-x; 1/2-1/16*(16*y-16*x)*x+1/2*y^2-1/2*x^2,1/2-1/16*(16*y-16*x)*y+1/2*x^2,2*y*x
dK is: 1+y^2+x^2,2*y*x; 1+y^2+x^2,2*y*x; 1+y^2+x^2,2*y*x

Curvature of K-orbits on the v-axis: (-2*y^5-2*y-4*y^3)*(1+y^4+2*y^2)^(-1.5)

Difference to foci is:
Difference to foci is: 8.125
Difference to foci is: 4.250
Difference to foci is: 2.500
Difference to foci is: 2.000
Difference to foci is: 2.500
Difference to foci is: 3.333
Difference to foci is: 5.200
Difference to foci is: 10.100
Difference to foci is: 100.010

Appendix B. A Sample of Graphics Generated by the Program

A sample of graphics produced by the program and post-processed with MetaPost is shown on Figure 1. Some more examples can be found in [12].
Figure 1. The elliptic, parabolic and hyperbolic unit disks.

Appendix C. Index of Identifiers

calc_par_focal: 17a, 17b, 17c

catch: 11a, 12a, 13, 14, 15, 16a, 17a, 17b, 17c

catch_handle: 12b, 13, 14, 15, 16a, 17a, 17b, 17c

close_curve: 5d, 6d, 14b, 15a

color_name: 5c, 8a, 12b

curves: 14d, 15a

eLLIPtic: 4a, 5a, 10a, 10b, 18b

exP_scale: 14d, 15b
fileout: 5b, 5c, 5d, 8a, 12b, 12c, 13b, 13c, 15a
get_components: 6a, 15b, 16b, 17b, 17c
grey: 5a, 12b
hyperbolic: 1a, 5a, 6a, 10b, 14c, 15a, 16b, 18d
if_in_limits: 6a, 15b, 16b, 17b, 17c
init Coord: 9b, 15a
main: 3
metric: 4a, 6a, 7b, 7c, 8a, 8c, 9a, 9c, 10a, 13a, 13b, 14a, 14c, 15a, 16c, 16d, 17a, 18a, 19a, 19b
numeric: 4b, 5a, 11a, 11b, 12a, 13a, 13b, 14a, 14c, 15a, 16c, 16d, 16e, 16f, 16g, 16h, 17a, 17b, 17c
open file: 7c, 12b, 12c, 13d
output focal: 19b, 19d
parabolic: 5a, 10b, 17a, 18c, 19c
put draw: 5c, 5d, 9b
put point: 5d, 6a
renew curve: 5d, 6a, 6b
S: 14c, 15a
sgroup: 7b, 7c, 11b, 19d, 20
sub group A: 4b, 5a, 11a, 11b, 15a, 16a, 20
sub group K: 4b, 5a, 11a, 11b, 12a, 13a, 13b, 14a, 14c, 15a, 16c, 16d, 16e, 16f, 17a, 17b, 18a, 19d
sub group N: 5a, 16a, 16b, 17a, 17b, 18a, 19d
transverse dir: 7a, 16b, 16c, 17a, 17c, 18a, 19d
u: 5a, 5d, 5e, 12b, 19a
ulim: 5a, 14c
upos: 5b, 5c, 5e, 6a, 8a
v: 8a, 8b, 8e, 12a, 12b, 19c
vilimits: 9a, 15a, 15b, 15c, 16a, 16b

References

[1] Christian Bauer, Alexander Frink, Richard Kreckel, and Jens Vollinga. GiNaC is Not a CAS. [http://www.ginac.de/]
[2] F. Brackx, Richard Delanghe, and F. Sommen. Clifford Analysis, volume 76 of Research Notes in Mathematics. Pitman (Advanced Publishing Program), Boston, MA, 1982.
[3] Jonathan Brandmeyer. PyGiNaC—a Python interface to the C++ symbolic math library GiNaC. [http://sourceforge.net/projects/pyginac/]
[4] Jan Chops. An introduction to Dirac operators on manifolds, volume 24 of Progress in Mathematical Physics. Birkhäuser Boston Inc., Boston, MA, 2002.
[5] R. Delanghe, F. Sommen, and V. Soucek. Clifford Algebra and Spinor-Valued Functions, volume 53 of Mathematics and its Applications. Kluwer Academic Publishers Group, Dordrecht, 1992. A function theory for the Dirac operator, Related REDUCE software by F. Brackx and D. Constales, With 1 IBM-PC floppy disk (3.5 inch).
[6] B. A. Dubrovin, A. T. Fomenko, and S. P. Novikov. Modern geometry—methods and applications. Part I, volume 93 of Graduate Texts in Mathematics. Springer-Verlag, New York, second edition, 1992. The geometry of surfaces, transformation groups, and fields, Translated from the Russian by Robert G. Burns.
[7] John W. Eaton et al. GNU Octave—high-level language, primarily intended for numerical computations. [http://www.octave.org/]
[8] Bertfried Fauser. Clifford algebras and spinor-valued functions—1: Geometry in the sense of F. Klein. 2005. E-print: arXiv:math.CT/0410044. (To appear).
[9] Bertfried Fauser and Ralf Blumenthal. On the decomposition of Clifford algebras of arbitrary bilinear form. In: Ablamowicz, Rafał (ed.) et al., Clifford Algebras and their Applications in Mathematical Physics, Proceedings of the 5th conference, Ixtapa-Zihuatanejo, Mexico, June 27-July 4, 1999. Volume I: Algebra and physics. Boston, MA: Birkhäuser. Prog. Phys. 18, 341-366. 2000. Zbl # 0955.15018.
[10] Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. GNU General Public License, second edition, 1991. [http://www.gnu.org/licenses/gpl.html/]
[11] John D. Hobby. MetaPost: A MetaFont like system with postscript output. [http://www.tug.org/metapost.html]
[12] Vladimir V. Kisil. Elliptic, parabolic and hyperbolic analytic function theory—1: Geometry in the sense of F. Klein. 2005. E-print: arXiv:math.CV/0410044. (To appear).
[13] Vladimir V. Kisil. An example of Clifford algebras calculations with GiNaC. Adv. in Appl. Clifford Algebras, 15(1), 1-26, 1996.
[14] Vladimir V. Kisil and Debapriya Biswas. Elliptic, parabolic and hyperbolic analytic function theory—0: Geometry of domains. In: Complex Analysis and Free Boundary Flows, volume 1 of Trans. Inst. Math. of the NAS of Ukraine, pages 100–118, 2004. E-print: arXiv:math.CT/0410044.
[15] Serge Lang. SL2(R), volume 105 of Graduate Texts in Mathematics. Springer-Verlag, New York, 1985. Reprint of the 1975 edition.
[16] Norman Ramsey. Noweb—a simple, extensible tool for literate programming. [http://www.eecs.harvard.edu/~nr/noweb/]
[17] David B. Thompson. The literate programming FAQ. [http://shelob.com/](http://shelob.com/).
[18] S. Weinzierl and R. Marani. gTybalt - a free computer algebra system. [http://www.fis.unipr.it/~stefanw/gtybalt.html]
School of Mathematics, University of Leeds, Leeds LS2 9JT, UK

E-mail address: kisilv@maths.leeds.ac.uk

URL: http://maths.leeds.ac.uk/~kisilv/