CoReLG

Computing with real Lie Algebras

Version 1.20

December 2014

Heiko Dietrich
Paolo Faccin
Willem de Graaf

Heiko Dietrich Email: heiko.dietrich@monash.edu
Homepage: http://users.monash.edu.au/~heikod/
Address: School of Mathematical Sciences
Monash University
Wellington Road 1
VIC 3800, Melbourne, Australia

Paolo Faccin Email: paolofaccin86@gmail.com
Address: Dipartimento di Matematica
Via Sommarive 14
I-38050 Povo (Trento), Italy

Willem de Graaf Email: degraaf@science.unitn.it
Homepage: http://www.science.unitn.it/~degraaf/
Address: Dipartimento di Matematica
Via Sommarive 14
I-38050 Povo (Trento), Italy

Abstract

This package provides functions for computing with various aspects of the theory of real simple Lie algebras.

Copyright

Acknowledgements

The research leading to this package has received funding from the European Union's Seventh Framework Program FP7/2007-2013 under grant agreement no 271712.

Contents

Chapter 1

Introduction

CoReLG (Computing with Real Lie Groups) is a GAP package for computing with (semi-)simple real Lie algebras. Various capabilities of the package have to do with the action of the adjoint group of a real Lie algebra (such as the nilpotent orbits, and non-conjugate Cartan subalgebras). CoReLG is also the acronym of the EU funded Marie Curie project carried out by the first author of the package at the University of Trento.

The simple real Lie algebras have been classified, and this classification is the main theoretical tool that we use, as it determines the objects that we work with. In Section 1.1 we give a brief account of this classification. We refer to the standard works in the literature (e.g., [Kna02]) for an in-depth discussion. The algorithms of this package are described in [DG13] and [DFG13].

We remark that the package still is under development, and its functionality is continuously extended. The package SLA, [Gra12], is required.

1.1 The simple real Lie algebras

Let $\mathfrak{g}^c$ denote a complex simple Lie algebra. Then there are two types of simple real Lie algebras associated to $\mathfrak{g}^c$: the *realification* of $\mathfrak{g}^c$ (this means that $\mathfrak{g}^c$ is viewed as an algebra over $\mathbb{R}$, of dimension $2\dim\mathfrak{g}^c$), and the *real forms* $\mathfrak{g}$ of $\mathfrak{g}^c$ (this means that $\mathfrak{g}\otimes_{\mathbb{R}}\mathbb{C}$ is isomorphic to $\mathfrak{g}^c$). It is straightforward to construct the realification of $\mathfrak{g}^c$; so in the rest of this section we concentrate on the real forms of $\mathfrak{g}^c$.

A Lie algebra is said to be *compact* if its Killing form is negative definite. The complex Lie algebra $\mathfrak{g}^c$ has a unique (up to isomorphism) compact real form $\mathfrak{u}$. In the sequel we fix the compact form $\mathfrak{u}$. Then $\mathfrak{g}^c = \mathfrak{u} + \imath\mathfrak{u}$, where $\imath$ is the complex unit; so we get an antilinear map $\tau : \mathfrak{g}^c \to \mathfrak{g}^c$ by $\tau(x+\imath y) = x - \imath y$, where $x, y \in \mathfrak{u}$. This is called the *conjugation* of $\mathfrak{g}^c$ with respect to $\mathfrak{u}$.

Now let θ be an automorphism of $\mathfrak{g}^c$ of order 2, commuting with τ. Then θ stabilises $\mathfrak{u}$, so the latter is the direct sum of the ±1-eigenspaces of θ, say $\mathfrak{u} = \mathfrak{u}_1 \oplus \mathfrak{u}_{-1}$. Set $\mathfrak{k} = \mathfrak{u}_1$ and $\mathfrak{p} = i\mathfrak{u}_{-1}$. Then $\mathfrak{g} = \mathfrak{g}(\theta) = \mathfrak{k} \oplus \mathfrak{p}$ is a real form of $\mathfrak{g}^c$. Regarding this construction we remark the following:

- $\mathfrak{g} = \mathfrak{k} \oplus \mathfrak{p}$ is called a *Cartan decomposition*. It is unique up to inner automorphisms of $\mathfrak{g}$.
- The map θ is a *Cartan involution*; it is the identity on $\mathfrak{k}$ and acts as multiplication by -1 on $\mathfrak{p}$)
- $\mathfrak{k}$ is compact, and it is a maximal compact subalgebra of $\mathfrak{g}$.
- Two real forms are isomorphic if and only if the corresponding Cartan involutions are conjugate in the automorphism group of $\mathfrak{g}^c$.

- The automorphism θ is described by two pieces of data: a list of signs $(s_1,\dots,s_r)$ of length equal to the rank r of $\mathfrak{g}$, and a permutation π of $1,\dots,r$, leaving the list of signs invariant. Let $\alpha_1,\dots,\alpha_r$ denote the simple roots of $\mathfrak{g}^c$ with corresponding canonical generators x_i,y_i,h_i. Then $\theta(x_i)=s_i x_{\pi(i)}$, $\theta(y_i)=s_i y_{\pi(i)}$, $\theta(h_i)=h_{\pi(i)}$.

1.2 Cartan subalgebras and more

Let $\mathfrak{g}$ be a real form of the complex Lie algebra $\mathfrak{g}^c$, with Cartan decomposition $\mathfrak{g}=\mathfrak{k}\oplus\mathfrak{p}$. A Cartan subalgebra $\mathfrak{h}$ of $\mathfrak{g}$ is *standard* (with respect to this Cartan decomposition) if $\mathfrak{h}=(\mathfrak{h}\cap\mathfrak{k})\oplus(\mathfrak{h}\cap\mathfrak{p})$, or, equivalently, when $\mathfrak{h}$ is stable under the Cartan involution θ.

It is a fact that every Cartan subalgebra of $\mathfrak{g}$ is conjugate by an inner automorphism to a standard one ([Kna02], Proposition 6.59). Moreover, there is a finite number of non-conjugate (by inner automorphisms) Cartan subalgebras of $\mathfrak{g}$ ([Kna02], Proposition 6.64). A standard Cartan subalgebra $\mathfrak{h}$ is said to be *maximally compact* if the dimension of $\mathfrak{h}\cap\mathfrak{k}$ is maximal (among all standard Cartan subalgebras). It is called *maximally non-compact* if the dimension of $\mathfrak{h}\cap\mathfrak{p}$ is maximal. We have that all maximally compact Cartan subalgebras are conjugate via the inner automorphism group. The same holds for all maximally non-compact Cartan subalgebras ([Kna02], Proposition 6.61).

A subspace of $\mathfrak{p}$ is said to be a *Cartan subspace* if it consists of commuting elements. If $\mathfrak{h}$ is a maximally non-compact standard Cartan subalgebra, then $\mathfrak{c}=\mathfrak{h}\cap\mathfrak{p}$ is a Cartan subspace. The other Cartan subalgebras (i.e., representatives of the conjugacy classes of the Cartan subalgebras under the inner automorphism group) can be constructed such that their intersection with $\mathfrak{p}$ is contained in $\mathfrak{c}$.

Every standard Cartan subalgebra $\mathfrak{h}$ of $\mathfrak{g}$ yields a corresponding root system Φ of $\mathfrak{g}^c$. Let $\alpha\in\Phi$, then a short argument shows that $\alpha\circ\theta$ (where $\alpha\circ\theta(h)=\alpha(\theta(h))$ for $h\in\mathfrak{h}$) is also a root (i.e., lies in Φ). This way we get an automorphism of order 2 of the root system Φ.

Now let $\mathfrak{h}$ be a maximally compact standard Cartan subalgebra of $\mathfrak{g}$, with root system Φ. Then it can be shown that there is a basis of simple roots $\Delta\subset\Phi$ which is θ-stable. Write $\Delta=\{\alpha_1,\dots,\alpha_r\}$, and let x_i,y_i,h_i be a corresponding set of canonical generators. Then there is a sequence of signs $(s_1,\dots,s_r)$ and a permutation π of $1,\dots,r$ such that $\theta(x_i)=s_i x_{\pi(i)}$. Now we encode this information in the Dynkin diagram of Φ. If $s_i=-1$ then we paint the node corresponding to α_i black. Also, if $\pi(i)=j\neq i$ then the nodes corresponding to α_i, α_j are connected by an arrow. The resulting diagram is called a *Vogan diagram* of $\mathfrak{g}$. It determines the real form $\mathfrak{g}$ up to isomorphism. The signs s_i are not uniquely determined. However, it is possible to make a "canonical" choice for the signs so that the Vogan diagram is uniquely determined.

Now let $\mathfrak{h}$ be a maximally non-compact standard Cartan subalgebra of $\mathfrak{g}$, with root system Φ. Then, in general, there is no basis of simple roots which is stable under θ. However we can still define a diagram, in the following way. Let $\mathfrak{c}=\mathfrak{h}\cap\mathfrak{p}$ be the Cartan subspace contained in $\mathfrak{h}$. Let $\Phi_c=\{\alpha\in\Phi\mid\alpha\circ\theta=\alpha\}=\{\alpha\in\Phi\mid\alpha(\mathfrak{c})=0\}$ be the set of *compact roots* . Then there is a choice of positive roots Φ^+ such that $\alpha\circ\theta\in\Phi^-$ for all *non-compact* positive roots $\alpha\in\Phi^+$. Let Δ denote the basis of simple roots corresponding to Φ^+. A theorem due to Satake says that there is a bijection $\tau:\Delta\to\Delta$ such that $\tau(\alpha)=\alpha$ if $\alpha\in\Phi_c$, and for non-compact $\alpha\in\Delta$ we have $\alpha\circ\theta=-\tau(\alpha)-\sum_{\gamma\in\Delta_c}c_{\alpha,\gamma}\gamma$, where $\Delta_c=\Delta\cap\Phi_c$ and the $c_{\alpha,\gamma}$ are non-negative integers. Now we take the Dynkin diagram corresponding to Δ, where the nodes corresponding to the compact roots are painted black, and the nodes corresponding to a pair $\alpha,\tau(\alpha)$, if they are unequal, are joined by arrows. The resulting diagram is called the *Satake diagram* of $\mathfrak{g}$. It determines the real form $\mathfrak{g}$ up to isomorphism.

1.3 Nilpotent orbits

By G^c, G we denote the adjoint groups of $\mathfrak{g}^c$ and $\mathfrak{g}$ respectively. The nilpotent G^c-orbits in $\mathfrak{g}^c$ have been classified by so-called weighted Dynkin diagrams. A nilpotent G^c-orbit in $\mathfrak{g}^c$ may have no intersection with the real form $\mathfrak{g}$. On the other hand, when it does have an intersection, then this may split into several G-orbits.

Let e be an element of a nilpotent G-orbit in $\mathfrak{g}$. By the Jacobson-Morozov theorem, e lies in an $\mathfrak{sl}_2$-triple (e,h,f); here this means that $[h,e]=2e$, $[h,f]=-2f$, and $[e,f]=h$. The triple is called a *real Cayley triple* if $\theta(e)=-f$, $\theta(f)=-e$ and $\theta(h)=-h$, where θ is the Cartan involution of $\mathfrak{g}$. Every nilpotent orbit has a representative lying in a real Cayley triple.

1.4 On base fields

To define a Lie algebra by a multiplication table over the reals, it usually suffices to take a subfield of the real field as base field. However, the algorithms contained in this package very often need a Chevalley basis of the Lie algebra at hand, which is defined only over the complex field. Computations with such a Chevalley basis take place behind the scenes, and the result is again defined over the reals. However, the computations would not be possible if the Lie algebra is just defined over (a subfield of) the reals. For this reason, we require that the base field contains the imaginary unit `E(4)`.

Furthermore, in many algorithms it is necessary to take square roots of elements of the base field. So the ideal base field would contain the imaginary unit, as well as being closed under taking square roots. However, such a field is difficult to construct and to work with on a computer. For this reason we have provided the field `SqrtField` containing the square roots of all rational numbers. Mathematically, this is the field $\mathbb{Q}^{\surd}(\imath)$ with $\mathbb{Q}^{\surd}=\mathbb{Q}(\{\sqrt{p}\mid p \text{ a prime}\})$ and $\imath=\sqrt{-1}\in\mathbb{C}$. Clearly, $\mathbb{Q}^{\surd}(\imath)$ is an infinite extension of the rationals $\mathbb{Q}$, and every f in $\mathbb{Q}^{\surd}(\imath)$ can be uniquely written as $f=\sum_{j=1}^{m} r_i\sqrt{k_j}$ for Gaussian rationals $r_i\in\mathbb{Q}(\imath)$ and pairwise distinct squarefree positive integers $k_1,\ldots,k_m$. Thus, f can be described efficiently by its coefficient vector $[[r_1,k_1],\ldots,[r_j,k_j]]$. We comment on our implementation of $\mathbb{Q}^{\surd}(\imath)$ in Chapter 2.

Although it is possible to try most functions of the package using the base field `CF(4)`, for example, it is likely that many computations will result in an error, because of the lack of square roots in that field. Many more computations are possible over `SqrtField`, but also in that case, of course, a computation may result in an error because we cannot construct a particular square root. Also, computations over `SqrtField` tend to be significantly slower than over, say, `CF(4)`; see the next example. But that is a price we have to pay (at least, in order to be able to do some computations).

Example

```
gap> L:=RealFormById("E",8,2);
<Lie algebra of dimension 248 over SqrtField>
gap> allCSA := CartanSubalgebrasOfRealForm(L);;time;
67224
gap> L:=RealFormById("E",8,2,CF(4));
<Lie algebra of dimension 248 over GaussianRationals>
gap> allCSA := CartanSubalgebrasOfRealForm(L);;time;
7301
# We remark that both computations are exactly the same;
# the difference in timing is caused by the fact that
# arithmetic over SqrtField is slower.
```

Chapter 2

The field *SqrtField*

2.1 Definition of the field

The field $\mathbb{Q}^{\surd}(\imath)$ with $\mathbb{Q}^{\surd} = \mathbb{Q}(\{\sqrt{p} \mid p \text{ a prime}\})$ and $\imath = \sqrt{-1} \in \mathbb{C}$ is realised as *SqrtField*. A few functions print some information on what they are doing to the info class *InfoSqrtField*; this can be turned off by setting *SetInfoLevel(InfoSqrtField, 0);*.

2.1.1 SqrtFieldIsGaussRat

▷ SqrtFieldIsGaussRat(*q*) (function)

Here *q* is an element of *SqrtField*; this function returns *true* if and only if *q* is the product of *One(SqrtField)* and a Gaussian rational.

Example

```
gap> F := SqrtField;
SqrtField
gap> IsField( F ); LeftActingDomain( F ); Size( F ); Characteristic( F );
true
GaussianRationals
infinity
0
gap> one := One( F );
1
gap> 2 in F; 2*one in F; 2*E(4)*one in F;
false
true
true
gap> a := 2/3*E(4)*one;;
gap> a in SqrtField; a in GaussianRationals; SqrtFieldIsGaussRat( a );
true
false
true
```

2.2 Construction of elements

Every f in *SqrtField* can be uniquely written as $f = \sum_{j=1}^{m} r_i\sqrt{k_j}$ for Gaussian rationals $r_i \in \mathbb{Q}(\imath)$ and pairwise distinct squarefree positive integers $k_1,\ldots,k_m$. Thus, f can be described efficiently by its coefficient vector $[[r_1,k_1],\ldots,[r_j,k_j]]$.

2.2.1 Sqroot

▷ `Sqroot(q)` (function)

Here `q` is a rational number and *Sqroot(q)* is the element $\sqrt{q}$ as an element of *SqrtField*. If $q = (-1)^\varepsilon a/b$ with coprime integers $a,b \geq 0$ and $\varepsilon \in \{0,1\}$, then *Sqroot(q)* is represented as the element *E(4)^ε*b*Sqroot(ab)* of *SqrtField*.

2.2.2 CoefficientsOfSqrtFieldElt

▷ `CoefficientsOfSqrtFieldElt(f)` (function)

If *f* is an element in *SqrtField*, then *CoefficientsOfSqrtFieldElt(f)* returns its coefficient vector $[[r_1,k_1],\ldots,[r_m,k_m]]$ as described above, that is, $r_1,\ldots,r_m \in \mathbb{Q}(\imath)$ and $k_1,\ldots,k_m$ are pairwise distinct positive squarefree integers such that $f = \sum_{j=1}^{m} r_j\sqrt{k_j}$.

2.2.3 SqrtFieldEltByCoefficients

▷ `SqrtFieldEltByCoefficients(l)` (function)

If *l* is a list $[[r_1,k_1],\ldots,[r_m,k_m]]$ with Gaussian rationals r_j and rationals k_j, then *SqrtFieldEltByCoeffiients(l)* returns the element $\sum_{j=1}^{m} r_j\sqrt{k_j}$ as an element of *SqrtField*. Note that here $k_1,\ldots,k_m$ need not to be positive, squarefree, or pairwise distinct.

Example

```
gap> Sqroot(-(2*3*4)/(11*13)); Sqroot(245/15); Sqroot(16/9);
2/143*E(4)*Sqroot(858)
7/3*Sqroot(3)
4/3
gap> a := 2+Sqroot(7)+Sqroot(99);
2 + Sqroot(7) + 3*Sqroot(11)
gap> CoefficientsOfSqrtFieldElt(a);
[ [ 2, 1 ], [ 1, 7 ], [ 3, 11 ] ]
gap> SqrtFieldEltByCoefficients([[2,9],[1,7],[E(4),13]]);
6 + Sqroot(7) + E(4)*Sqroot(13)
```

2.2.4 SqrtFieldEltToCyclotomic

▷ `SqrtFieldEltToCyclotomic(f)` (function)

If *f* lies in *SqrtField* with coefficient vector $[[r_1,k_1],\ldots,[r_m,k_m]]$, then *SqrtFieldEltToCyclotomic(f)* returns $\sum_{j=1}^{m} r_j\sqrt{k_j}$ lying in a suitable cyclotomic field *CF(n)*. The degree n can easily become too large, hence this function should be used with caution.

2.2.5 SqrtFieldEltByCyclotomic

▷ SqrtFieldEltByCyclotomic(c) (function)

If *c* is an element of $\mathbb{Q}^{\sqrt{}}(\iota)$ represented as an element of a cyclotomic field *CF(n)*, then *SqrtFieldEltByCyclotomic(c)* returns the corresponding element in *SqrtField*. Our algorithm for doing this is described in [DG13].

Example

```
gap> SqrtFieldEltToCyclotomic( Sqroot(2) );
E(8)-E(8)^3
gap> SqrtFieldEltToCyclotomic( Sqroot(2)+E(4)*Sqroot(7) );
E(56)^5+E(56)^8+E(56)^13-E(56)^15+E(56)^16-E(56)^23-E(56)^24+E(56)^29-E(56)^31+
E(56)^32+E(56)^37-E(56)^39-E(56)^40+E(56)^45-E(56)^47-E(56)^48+E(56)^53-E(56)^55
gap> SqrtFieldEltByCyclotomic( E(8)-E(8)^3 );
Sqroot(2)
gap> SqrtFieldEltByCyclotomic( 3*E(4)*Sqrt(11)-2/4*Sqrt(-13/7) );
3*E(4)*Sqroot(11) + (-1/14*E(4))*Sqroot(91)
```

2.3 Basic operations

All basic field operations are available. The inverse of an element f in *SqrtField* as follows: We first compute the minimal polynomial $p(X)$ of f over $\mathbb{Q}(\iota)$, that is, a non-trivial linear combination $0 = p(f) = a_0 + a_1 f + \ldots a_{i-1} f^{i-1} + f^i$. Then $f^{-1} = -(a_1 + a_2 f + \ldots + a_{i-1} f^{i-2} + f^{i-1})/a_0$. Although the inverse of f can be computed with linear algebra methods only, the degree of the minimal polynomial of f can become rather large. For example, if $f = \sum_{j=1}^{m} r_i \sqrt{k_j}$ for rational r_i and pairwise distinct positive squarefree integers $k_1, \ldots, k_m$, then f is a primitive element of the number field $\mathbb{Q}(\sqrt{k_1}, \ldots, \sqrt{k_m})$, see for example Lemma A.5 in [DG13]. For larger degree, the progress of the computation of the inverse is printed via the InfoClass *InfoSqrtField*. We remark that the method *Random* simply returns a sum of a few terms $a\sqrt{b}$ where a, b are random rationals constructed with *Random(Rationals)*.

Example

```
gap> a := Sqroot( 2 ) + 3 * Sqroot( 3/7 ); b := Sqroot( 21 ) - Sqroot( 2 );
Sqroot(2) + 3/7*Sqroot(21)
(-1)*Sqroot(2) + Sqroot(21)
gap> a + b; a * b; a - b;
10/7*Sqroot(21)
7 + 4/7*Sqroot(42)
2*Sqroot(2) + (-4/7)*Sqroot(21)
gap> c := ( a - b )^-2;
91/8 + 7/4*Sqroot(42)
gap> a := Sum( List( [2,3,5,7], x -> Sqroot( x ) ) );
Sqroot(2) + Sqroot(3) + Sqroot(5) + Sqroot(7)
gap> b := a^-1; a*b;
37/43*Sqroot(2) + (-29/43)*Sqroot(3) + (-133/215)*Sqroot(5) +
27/43*Sqroot(7) + 62/215*Sqroot(30) + (-10/43)*Sqroot(42) + (-34/215)*Sqroot(70)
+ 22/215*Sqroot(105)
1
gap> ComplexConjugate(Sqroot(17)+Sqroot(-7));
(-E(4))*Sqroot(7) + Sqroot(17)
```

```
gap> Random( SqrtField );
-1 + 1/4*Sqroot(3) + 1/9*Sqroot(6)
```

Most methods for list, matrices, and polynomials also work over *SqrtField*.

Example

```
gap> m:=[[Sqroot(2),Sqroot(3)],[Sqroot(2),Sqroot(5)],[1,0]]*One(SqrootField);
[ [ Sqroot(2), Sqroot(3) ], [ Sqroot(2), Sqroot(5) ], [ 1, 0 ] ]
gap> NullspaceMat(m);
[ [ (-5/4)*Sqroot(2) + (-1/4)*Sqroot(30), 3/4*Sqroot(2) + 1/4*Sqroot(30), 1 ] ]
gap> RankMat(m);
2
gap> m := [[Sqroot(2),Sqroot(3)],[Sqroot(2),Sqroot(5)]];
[ [ Sqroot(2), Sqroot(3) ], [ Sqroot(2), Sqroot(5) ] ]
gap> Determinant( m );  DefaultFieldOfMatrix( m );
(-1)*Sqroot(6) + Sqroot(10)
SqrtField
gap> x := Indeterminate( SqrtField, "x" );; f := x^2+x+1;
x^2+x+1
```

2.3.1 SqrtFieldMakeRational

▷ `SqrtFieldMakeRational(m)` (function)

If *m* is an element of *SqrtField*, or a list or a matrix over *SqrtField*, defined over the Gaussian rationals, then *SqrtFieldMakeRational(m)* returns the corresponding element in $\mathbb{Q}(\iota)$ or defined over $\mathbb{Q}(\iota)$, respectively. This function is used internally, for example, to compute the determinant or rank of a rational matrix over *SqrtField* more efficiently. It is also used in the following three functions.

2.3.2 SqrtFieldPolynomialToRationalPolynomial

▷ `SqrtFieldPolynomialToRationalPolynomial(f)` (function)

Here *f* is a polynomial over *SqrtField* but with coefficients in the Gaussian rationals. The function returns the corresponding polynomial defined over the Gaussian rationals.

2.3.3 SqrtFieldRationalPolynomialToSqrtFieldPolynomial

▷ `SqrtFieldRationalPolynomialToSqrtFieldPolynomial(f)` (function)

If *f* is a polynomial over the Gaussian rationals, then the function returns the corresponding polynomial defined over *SqrtField*.

2.3.4 Factors

▷ `Factors(f)` (operation)

If *f* is a rational polynomial defined over *SqrtField*, then the previous two functions are used to obtain its factorisation over $\mathbb{Q}$.

Example

```
gap> F := SqrtField;; one := One( SqrtField );;
gap> x := Indeterminate( F, "x" );; f := x^5 + 4*x^3 + E(4)*one*x;
x^5+4*x^3+E(4)*x
gap> SqrtFieldPolynomialToRationalPolynomial(f);
x_1^5+4*x_1^3+E(4)*x_1
gap> SqrtFieldRationalPolynomialToSqrtFieldPolynomial(last);
x^5+4*x^3+E(4)*x
gap> f := x^2-1;; Factors(f);
[ x-1, x+1 ]
gap> f := x^2+1;; Factors(f);
[ x^2+1 ]
```

Chapter 3

Real Lie Algebras

3.1 Construction of simple real Lie algebras

A few functions print some information on what they are doing to the info class *InfoCorelg*.

3.1.1 RealFormsInformation

▷ RealFormsInformation(*type*, *rank*) (function)

This function displays information regarding the simple real Lie algebras that can be constructed from the complex Lie algebra of type *type* (which is a string) and rank *rank* (a positive integer). Each Lie algebra is given an index which is an integer, and for each index some information is given on the Lie algebra, such as a commonly used name. In all cases the index 0 refers to the realification of the complex Lie algebra.

Example

```
gap> RealFormsInformation( "A", 4 );

  There are 4 simple real forms with complexification A4
    1 is of type su(5), compact form
    2 - 3 are of type su(p,5-p) with 1 <= p <= 2
    4 is of type sl(5,R)
  Index '0' returns the realification of A4

gap> RealFormsInformation( "E", 6 );

  There are 5 simple real forms with complexification E6
    1 is the compact form
    2 is EI   = E6(6), with k_0 of type sp(4) (C4)
    3 is EII  = E6(2), with k_0 of type su(6)+su(2) (A5+A1)
    4 is EIII = E6(-14), with k_0 of type so(10)+R (D5+R)
    5 is EIV  = E6(-26), with k_0 of type f_4 (F4)
  Index '0' returns the realification of E6

gap> NumberRealForms("D",10);
12
```

3.1.2 NumberRealForms

▷ NumberRealForms(*type*, *rank*) (function)

This function returns the number of (isomorphism types of) all real forms of the simple complex Lie algebras of type *type* and rank *rank*.

3.1.3 RealFormById

▷ RealFormById(*type*, *rank*, *id*) (function)
▷ RealFormById(*type*, *rank*, *id*, *F*) (function)

Let L be the complex Lie algebra of type *type* and rank *rank*. This function constructs the real form of L with index *id* (see RealFormsInformation (3.1.1)). By default this Lie algebra is constructed over the field *SqrtField*. However, by adding as an optional fourth argument the field *F*, it is possible to construct the Lie algebra output by this function over *F*. It is required that the complex unit *E(4)* is contained in *F*. If the index *ind* is 0, then the realification of L is constructed, which, strictly speaking is not a real form of L.

Example

```
gap> RealFormById( "A", 4, 2 );
<Lie algebra of dimension 24 over SqrtField>
gap> RealFormById( "A", 4, 2, CF(4) );
<Lie algebra of dimension 24 over GaussianRationals>
```

3.1.4 AllRealForms

▷ AllRealForms(*type*, *rank*) (function)

This function returns all real forms of the simple complex Lie algebras of type *type* and rank *rank* up to isomorphism. In the same way as with RealFormById (3.1.3) it is possible to add the base field as an optional third argument.

3.1.5 RealFormParameters

▷ RealFormParameters(*L*) (attribute)

For a real Lie algebra *L* constructed by the function RealFormById (3.1.3), this function returns a list of the parameters defining *L* as a real form of its complexification. The first element of the list is the type of *L* (given by a string), the second element is its rank, the third and fourth elements are the list of signs and the permutation defining the Cartan involution (see Section 1.1).

3.1.6 IsRealFormOfInnerType

▷ IsRealFormOfInnerType(*L*) (property)

Returns *true* if and only if the real form *L* is a defined by an inner involutive automorphism.

3.1.7 IsRealification

▷ IsRealification(*L*) (property)

Returns *true* if and only if the real form *L* is the realification of a complex simple Lie algebra.

3.1.8 CartanDecomposition

▷ CartanDecomposition(*L*) (attribute)

The Cartan decomposition of *L* as a record with entries *K*, *P*, and *CartanInv*, such that $L = K \oplus P$ is the Cartan decomposition with corresponding Cartan involution *CartanInv*, which is defined as a function on *L*.

The Lie algebras constructed by RealFormById (3.1.3) have this attribute stored. For other semisimple real Lie algebras it is computed. However, we do remark that the in the computation the root system is computed with respect to a Cartan subalgebra. If the program does not succeed in splitting the Cartan subalgebra over the base field of *L*, then the computation will not succeed.

Example

```
gap> L:= RealFormById( "A", 5, 3 );
<Lie algebra of dimension 35 over SqrtField>
gap> H := CartanSubalgebra(L);;
gap> K:= LieCentralizer( L, Subalgebra( L, [Basis( H )[1]] ) );
<Lie algebra of dimension 17 over SqrtField>
gap> DK:= LieDerivedSubalgebra( K );
<Lie algebra of dimension 15 over SqrtField>
gap> CartanDecomposition( DK );
rec( CartanInv := function( v ) ... end,
K := <Lie algebra of dimension 15 over SqrtField>,
P := <vector space over SqrtField, with 0 generators> )
# We see that the semisimple subalgebra DK is compact.
```

3.1.9 RealStructure

▷ RealStructure(*L*) (attribute)
▷ RealStructure(*L: basis := B*) (attribute)

The real structure of the real form *L* is the (complex) conjugation with respect to *L*, that is, the function which maps an element in *L* to the element constructed as follows: write it as a linear combination of the basis elements of *L* and replace each coefficient by its complex conjugate. If the optional argument *basis:=B* is given, then *B* has to be a basis whose span contains *L* (which is not checked by the code); in this case the linear combination is done with respect to *B*. The latter construction is important when one considers a subalgebra *M* of a real form *L*; here one could either do *Realstructure(M:basis:=Basis(L))* or *SetRealStructure(M,RealStructure(L))*.

3.2 Isomorphisms

3.2.1 IsomorphismOfRealSemisimpleLieAlgebras

▷ IsomorphismOfRealSemisimpleLieAlgebras(*K*, *L*) (function)

Here *K*, *L* are two real forms of a semisimple complex Lie algebra. This function returns an isomorphism if one exists. Otherwise *false* is returned.

Example

```
gap> L:=RealFormById("E",6,3);;
gap> H:=CartanSubalgebra(L);;
gap> K:=LieCentralizer(L,Subalgebra(L,Basis(H){[1,2,4]}));;
gap> DK:=LieDerivedSubalgebra(K);
<Lie algebra of dimension 8 over SqrtField>
gap> IdRealForm(DK);
[ "A", 2, 2 ]
gap> M:=RealFormById("A",2,2);
<Lie algebra of dimension 8 over SqrtField>
gap> IsomorphismOfRealSemisimpleLieAlgebras(DK,M);
<Lie algebra isomorphism between Lie algebras of dimension 8 over SqrtField>
```

3.3 Cartan subalgebras and root systems

3.3.1 MaximallyCompactCartanSubalgebra

▷ MaximallyCompactCartanSubalgebra(*L*) (attribute)

Here *L* is a real semisimple Lie algebra. This function returns a maximally compact Cartan subalgebra of *L*.

3.3.2 MaximallyNonCompactCartanSubalgebra

▷ MaximallyNonCompactCartanSubalgebra(*L*) (attribute)

Here *L* is a real semisimple Lie algebra. This function returns a maximally non-compact Cartan subalgebra of *L*.

3.3.3 CompactDimensionOfCartanSubalgebra

▷ CompactDimensionOfCartanSubalgebra(*L*) (function)
▷ CompactDimensionOfCartanSubalgebra(*L*, *H*) (function)

Here *L* is a real semisimple Lie algebra. This function returns the compact dimension of the Cartan subalgebra *H*. If *H* is not given, then *CartanSubalgebra(L)* will be taken. The compact dimension will be stored in the Cartan subalgebra, so that a new call to this function, with the same input, will return the compact dimension immediately.

3.3.4 CartanSubalgebrasOfRealForm

▷ `CartanSubalgebrasOfRealForm(L)` (attribute)

Here `L` is a real form of a complex semisimple Lie algebra. This function returns a list of Cartan subalgebras of `L`. They are representatives of all classes of conjugate (by the adjoint group) Cartan subalgebras of `L`.

3.3.5 CartanSubspace

▷ `CartanSubspace(L)` (attribute)

Here `L` is a real semisimple Lie algebra. This function returns a Cartan subspace of `L`. That is a maximal abelian subspace of the subspace `P` given in the `CartanDecomposition` (3.1.8) of `L`.

3.3.6 RootsystemOfCartanSubalgebra

▷ `RootsystemOfCartanSubalgebra(L)` (operation)
▷ `RootsystemOfCartanSubalgebra(L, H)` (operation)

Here `L` is a semisimple Lie algebra, and `H` is a Cartan subalgebra. (If `H` is not given, then *`CartanSubalgebra(L)`* will be taken.) This function returns the root system of `L` with respect to `H`. It is necessary that the eigenvalues of the adjoint maps corresponding to all elements of `H` lie in the ground field of `L`. However, even if they do, it is not guaranteed that this function succeeds, as it may happen that GAP has no polynomial factorisation algorithm over the ground field.

The root system is stored in `H`, so that a new call to this function, with the same input, will return the same root system.

3.3.7 ChevalleyBasis

▷ `ChevalleyBasis(R)` (attribute)

Here `R` is a root system of a semisimple Lie algebra `L`. This function returns a Chevalley basis of `L`, consisting of root vectors of `R`.

3.4 Diagrams

In this section we document the functionality for computing the Satake and Vogan diagrams of a real semisimple Lie algebra. In both cases the relevant function computes an object, which, when printed, does not reveal much information. However, *`Display`* with as input such an object, displays the diagram. Here we use the convention that every node is represented by an integer; nodes that are painted black are represented by integers in brackets; and the involution (i.e., the arrows in the diagram) are represented by a permutation of the nodes, printed on a line below the diagram.

3.4.1 VoganDiagram

▷ `VoganDiagram(L)` (attribute)

Here L is a real semisimple Lie algebra. This function returns the Vogan diagram of L.

Example

```
gap> L:= RealFormById( "E", 6, 3 );;
gap> K:= LieCentralizer( L, Subalgebra( L, Basis( CartanSubalgebra(L) ){[1]} ) );
<Lie algebra of dimension 36 over SqrtField>
gap> DK:= LieDerivedSubalgebra( K );
<Lie algebra of dimension 35 over SqrtField>
gap> vd:= VoganDiagram(DK);
<Vogan diagram in Lie algebra of type A5>
gap> Display( vd );
A5:  1---(2)---3---4---5
Involution: ()
```

3.4.2 SatakeDiagram

▷ `SatakeDiagram(`L`)` (attribute)

Here L is a real semisimple Lie algebra. This function returns the Satake diagram of L.

Example

```
gap> L:= RealFormById( "E", 6, 3 );;
gap> K:= LieCentralizer( L, Subalgebra( L, Basis( CartanSubalgebra(L) ){[1]} ) );
<Lie algebra of dimension 36 over SqrtField>
gap> DK:= LieDerivedSubalgebra( K );
<Lie algebra of dimension 35 over SqrtField>
gap> sd:= SatakeDiagram( DK );
<Satake diagram in Lie algebra of type A5>
gap> Display( sd );
A5:  1---2---(3)---4---5
Involution:  (1,5)(2,4)
```

Chapter 4

Real nilpotent orbits

4.1 Nilpotent orbits in real Lie algebras

CoReLG has a database of the nilpotent orbits of the real forms of the simple Lie algebras of ranks up to 8. When called the first time in a GAP session, CoReLG will first read the database of nilpotent orbits.

4.1.1 NilpotentOrbitsOfRealForm

▷ `NilpotentOrbitsOfRealForm(L)` (attribute)

Here `L` is a real form of a complex simple Lie algebra of rank up to 8. This function returns the list of nilpotent orbits (under the action of the adjoint group) of `L`. For this function to work, `L` must be defined over `SqrtField`.

Example

```
gap> L:= RealFormById( "F", 4, 3 );;
gap> no:= NilpotentOrbitsOfRealForm( L );;
#I CoReLG: read database of real triples ... done
gap> no[1];
<nilpotent orbit in Lie algebra>
```

4.1.2 RealCayleyTriple

▷ `RealCayleyTriple(o)` (attribute)

Here `o` is a nilpotent orbit constructed by `NilpotentOrbitsOfRealForm` (4.1.1) of a simple real Lie algebra. This function returns a real Cayley triple `[ f, h, e ]` corresponding to the orbit `o`. The third element `e` is a representative of the orbit.

Example

```
gap> L:= RealFormById( "F", 4, 2 );;
gap> no:= NilpotentOrbitsOfRealForm( L );;
gap> o:= no[10];
<nilpotent orbit in Lie algebra>
gap> t:=RealCayleyTriple(o);;
gap> theta:= CartanDecomposition(L).CartanInv;
function( v ) ... end
```

```
gap> theta(t[1]) = -t[3];
true
gap> theta(t[2]) = -t[2];
true
gap> t[3]*t[1] = t[2];
true
```

4.1.3 WeightedDynkinDiagram

▷ `WeightedDynkinDiagram(o)` (attribute)

Here `o` is a nilpotent orbit constructed by `NilpotentOrbitsOfRealForm` (4.1.1) of a simple real Lie algebra. This function returns the weighted Dynkin diagram of the orbit, which identifies its orbit in the complexification of the real Lie algebra in which `o` lies.

4.2 Nilpotent orbits in real Theta groups

Let $L = \bigoplus_{i\in\mathbb{Z}_m} L_i$ be a real semisimple Lie algebra, where $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ if $m \geq 1$, and $\mathbb{Z}_0 = \mathbb{Z}$. We provide some functions which help to determine the G_0-orbits of nilpotent elements in L_1, where G_0 is the adjoint group of L_0. An approach to compute these orbits is described in [DFG15]; the first step is to construct, up to G_0-conjugacy, all carrier subalgebras of L. Functions `CarrierAlgsForNilpOrbsInZGrading` (4.2.2) and `CarrierAlgsForNilpOrbsInZmGrading` (4.2.3) do this in the case that L is a split real form of a complex simple Lie algebra.

4.2.1 RealWeylGroup

▷ `RealWeylGroup(L)` (function)
▷ `RealWeylGroup(L, H)` (function)

Here `L` is a real semisimple Lie algebra with Cartan subalgebra `H`. (If `H` is not given, then `CartanSubalgebra(L)` will be taken.) This function returns the real Weyl group $N_G(H)/C_G(H)$ associated with `H`, where G is the adjoint group of `L`. The real Weyl group will be stored in the Cartan subalgebra, so that a new call to this function, with the same input, will return the real Weyl group immediately.

4.2.2 CarrierAlgsForNilpOrbsInZGrading

▷ `CarrierAlgsForNilpOrbsInZGrading(type, rank, d)` (attribute)

Gives a record containing (up to conjugacy) the carrier algebras of the real theta group specified by the input: here `type` and `rank` are the type and rank of the simple real Lie algebra (split real form) where everything happens; `d` is a list of degrees of the simple roots, defining a $\mathbb{Z}$ grading on the Lie algebra. The output is a record with the following entries: `L` the Lie algebra, `grad` the grading that was used (different format for Z-grading, `L0` the 0-component of the grading, `Hs` the Cartan subalgebras of `L0` that are used, `cars` the carrier algebras. Here `cars` is a list of lists; for each Cartan subalgebra of `L0` there is one list: the first corresponds to the split Cartan subalgebra, and so has just the complex carrier algebras (which are also real), the other lists contain lists as well, for each complex carrier

algebra (i.e., for each element of the first list) there is a list containing the real carrier algebras which are strongly H_i-regular, and over the complex numbers conjugate to the given complex carrier algebra. Furthermore, a carrier algebra is given by a record, containing the fields *g0*, *gp* (positive degree), and *gn* (negative degree).

Example

```
gap> ca:=CarrierAlgsForNilpOrbsInZGrading("G",2,[1,0]);
rec(
 Hs  := [ <Lie algebra of dimension 2 over SqrtField>,
          <Lie algebra of dimension 2 over SqrtField> ],
  L  :=   <Lie algebra of dimension 14 over SqrtField>,
  L0 :=   <Lie algebra of dimension 4 over SqrtField>,
cars :=  [
          [rec( g0 := [ v.13+(3)*v.14 ], gn := [ [ v.9 ] ], gp := [ [ v.3 ] ] )]
            ,[ [  ] ]
         ],
grad := rec( g0 := [ v.2, v.8, v.13, v.14 ],
              gn := [ [ v.7, v.9 ], [ v.10 ], [ v.11, v.12 ] ],
              gp := [ [ v.1, v.3 ], [ v.4 ], [ v.5, v.6 ] ] ) )
```

4.2.3 CarrierAlgsForNilpOrbsInZmGrading

▷ `CarrierAlgsForNilpOrbsInZmGrading(`*type, rank, m0, str, num*`)` (attribute)

Gives a record (up to conjugacy) containing the carrier algebras of the real theta group specified by the input: here *type* and *rank* are the type and rank of the simple real Lie algebra (split real form) where everything happens; *m0* is the order of the automorphism defining the grading, *str* is "inner" or "outer", depending on whether the automorphism is inner or not, *num* the *num*-th automorphism in the list *FiniteOrderInnerAutomorpisms(type, rank, m0)* or *FiniteOrderOuterAutomorphisms(type, rank, m0, 2)* which is used to define the grading. The output is as for `CarrierAlgsForNilpOrbsInZGrading` (4.2.2), with the exception that the record entry *grad* is a list with *m0* entries, the *i*-th entry containing the basis for the *i*-th component of the grading.

Example

```
gap> gap> ca:=CarrierAlgsForNilpOrbsInZmGrading("G",2,2,"inner",1);
rec(
 Hs := [ <Lie algebra of dimension 2 over SqrtField>,
         <Lie algebra of dimension 2 over SqrtField>,
         <Lie algebra of dimension 2 over SqrtField>,
         <Lie algebra of dimension 2 over SqrtField> ],
  L  :=  <Lie algebra of dimension 14 over SqrtField>,
  L0 :=  <Lie algebra of dimension 6 over SqrtField>,
cars := [
         [rec( g0 := [ v.13+v.14 ], gn := [ [ v.11 ] ], gp := [ [ v.5 ] ] ),
          rec( g0 := [ v.13, v.14 ], gn := [ [ v.2, v.10 ] ],
               gp := [ [ v.4, v.8 ] ] ),
          rec( g0 := [ v.13+(3/2)*v.14 ], gn := [ [ v.10 ] ],
               gp := [ [ v.4 ] ] ),
          rec( g0 := [ v.1, v.7, v.13, v.14 ],
               gn := [ [ v.8, v.9, v.10, v.11 ], [ v.12 ] ],
               gp := [ [ v.2, v.3, v.4, v.5 ], [ v.6 ] ] ),
          rec( g0 := [ v.13, v.14 ],
```

```
                gn := [ [ v.2, v.9 ], [ v.7 ], [ v.10 ], [ v.12 ], [ v.11 ] ],
                gp := [ [ v.3, v.8 ], [ v.1 ], [ v.4 ], [ v.6 ], [ v.5 ] ] ) ],
          [ [  ], [  ], [  ], [  ], [  ] ], [ [  ], [  ], [  ], [  ], [  ] ],
          [ [  ], [  ], [  ], [  ], [  ] ] ],
grad := [ [ v.1, v.6, v.7, v.12, v.13, v.14 ],
          [ v.2, v.3, v.4, v.5, v.8, v.9, v.10, v.11 ] ] )
```

References

[DFG13] H. Dietrich, P. Faccin, and W. A. de Graaf. Computing with real lie algebras: real forms, cartan decompositions, and cartan subalgebras. *Journal of Symbolic Computation*, 56:27–45, 2013. 4

[DFG15] H. Dietrich, P. Faccin, and W. A. de Graaf. Regular subalgebras and nilpotent orbits of real graded lie algebras. *Journal of Algebra*, 423:1044–1079, 2015. 19

[DG13] H. Dietrich and W. A. de Graaf. A computational approach to the Kostant-Sekiguchi correspondence. *Pacific Journal of Mathematics*, 265:349–379, 2013. 4, 9

[Gra12] W. A. de Graaf. SLA - computing with Simple Lie Algebras. a GAP package, 2012. `http://science.unitn.it/ degraaf/sla.html`, version 0.12. 4

[Kna02] A. W. Knapp. *Lie groups beyond an introduction*, volume 140 of *Progress in Mathematics*. Birkhäuser Boston Inc., Boston, MA, second edition, 2002. 4, 5

Index