# Sophus

## Computing in nilpotent Lie algebras

Version 1.12

October 2004

**Csaba Schneider**

**Csaba Schneider** — Email: csaba.schneider@sztaki.hu

— Homepage: http://www.sztaki.hu/˜schneider

— Address: Informatics Laboratory
Computer and Automation Research Institute
The Hungarian Academy of Sciences
1111 Budapest, Lágymányosi u. 11.
Hungary

## Abstract

Sophus is a GAP 4 package to compute with nilpotent Lie algebras over finite prime fields. In particular, the package can be used to compute certain central extensions and the automorphism group of such Lie algebras. Sophus also enables its user to test isomorphism between two nilpotent Lie algebras. The author of the package used it to construct all Lie algebras of dimension at most 9 over $\mathbb{F}_2$

## Copyright

## Acknowledgements

# Contents

# Chapter 1

# The theory

The Sophus package was originally designed to aid the author to classify some small-dimensional nilpotent Lie algebras over small fields. The classification follows the ideas that were used to classify small $p$-groups by O'Brien [O'B90]. The theory developed by O'Brien could easily be adopted to Lie algebras, and the details of this new theory can be found in [Sch]. Here we only summarise the main ideas, so that the user can understand the procedures implemented in this package. In this section $L$ denotes a finitely generated, and hence finite-dimensional, nilpotent Lie algebra. Suppose that $L$ has nilpotency class $c$, and hence the lower central series is as follows:

$$L = \gamma_1(L) > \gamma_2(L) = L' > \gamma_3(L) > \cdots > \gamma_c(L) > \gamma_{c+1}(L) = 0.$$

We say that a basis $\mathcal{B} = \{b_1, \ldots, b_n\}$ for $L$ is *compatible with the lower central series* if there are indices $1 = i_1 < i_2 < \cdots < i_c < n$ such that $\{b_{i_k}, \ldots, b_n\}$ is a basis of $\gamma_k(L)$ for $k \in \{1, \ldots, c\}$. We compute the structure constants table with respect to this basis, that is, we compute coefficients $\alpha_{i,j}^k$ for $1 \leq i < j < k \leq n$ such that

$$[b_i, b_j] = \sum_{k=j+1}^{n} \alpha_{i,j}^k b_k.$$

Suppose that $b_i \in \gamma_j(L) \setminus \gamma_{j+1}(L)$. Then we say that the number $j$ is the *weight* of the basis element $b_i$.

Note that in the nilpotent Lie algebra $L$ minimal generating sets have the same size, namely the dimension of $L/L'$. If $\dim L/L' = d$ then we call $L$ a *d-generator algebra*. We call a basis $\mathcal{B}$ a *nilpotent basis* if the following hold.

- The basis $\mathcal{B}$ is compatible with the lower central series.

- For each $b_i \in \mathcal{B}$ with weight $w \geq 2$ there are $b_{j_1}$, $b_{j_2} \in \mathcal{B}$ with weight 1 and $w - 1$, respectively such that $b_i = [b_{j_1}, b_{j_2}]$. The product $[b_{j_1}, b_{j_2}]$ is called the definition of $b_i$.

A Lie algebra $K$ is said to be a *central extension* of $L$ if $L \cong K/I$ for some ideal $I$ such that $I \leq Z(K) \cap K'$. Suppose that $c$ denotes the nilpotency class of $L$. Then a Lie algebra $K$ is an *immediate descendant* of $L$ if $K$ has class $c + 1$ and $K/\gamma_{c+1}(K) \cong L$. As in this case $\gamma_{c+1}(K) \leq Z(K) \cap K'$, it follows that an immediate descendant $K$ is a central extension of $L$. If $s = \dim \gamma_{c+1}(K)$ then $K$ is said to be a *step-s* immediate descendant of $L$.

Let $L$ be a $d$-generator nilpotent Lie algebra with class $c$, and let $F$ be a free Lie algebra of rank $d$. Choose an ideal $I$ of $F$ such that $L \cong F/I$. Then the Lie algebra $L^* = F/[I,F]$ is called the *Lie cover* of $L$. The *Lie multiplicator* in $L^*$ is the subspace $I/[I,F]$ and the *Lie nucleus* is $\gamma_c(L^*)$. It clear from the definition that $L^*/M \cong L$. It is verified in [Sch] that, up to isomorphism, the Lie cover, the

Lie multiplicator and the Lie nucleus are determined by the isomorphism type of *L*. Further, each central extension of the nilpotent Lie algebra *L* is a quotient of the Lie cover $L^*$. Thus it is possible to obtain all such descendants by first computing the Lie cover; this procedure is explained in [Sch]. Similar ideas can be used to compute the automorphism group of a nilpotent Lie algebra, and to verify isomorphism between two nilpotent Lie algebras; see [Sch] for details.

The main functions in Sophus are thus able to compute

- a nilpotent basis for a nilpotent Lie algebra;

- the cover of a nilpotent Lie algebra;

- the immediate descendants of a nilpotent Lie algebra;

- the full automorphism group of a nilpotent Lie algebra.

There is also a function in the package to check if two nilpotent Lie algebras are isomorphic. After repeated applications of the immediate descendants algorithm, it is, in theory, possible to list all nilpotent Lie algebras of a given dimension over a prime field $\mathbb{F}_p$. Of course, this computation requires relatively large computational resources, and quickly becomes unfeasible as the dimension or the characteristic *p* grows.

The Sophus package was written for the GAP 4 computer algebra system. In many procedures it is very important that we can compute the stabiliser of a subspace under some matrix group action. This is carried out using the procedures implemented in the autpgrp package [EO]. Hence this package is required to run Sophus.

The current version of Sophus deals with general nilpotent Lie algebras over finite prime fields. If you are to compute with Lie algebras obtained from group algebras via the bracket operation, then another GAP package LAGUNA [VBS] may also offer some very efficient methods.

# Chapter 2

# A sample calculation with Sophus

Before listing the functions of Sophus we present a sample calculation to show the reader what Sophus is able to compute. We list the isomorphism types of the 7-dimensional nilpotent Lie algebras over $\mathbb{F}_2$.

There is just one nilpotent Lie algebra with dimension 1 and dimension 2. We also set L3 to be a list containing the abelian Lie algebra with dimension 3.

```
──────────────────────────────── Example ────────────────────────────────

  gap> L1 := [ AbelianLieAlgebra( GF(2), 1 ) ];;
  gap> L2 := [ AbelianLieAlgebra( GF(2), 2 ) ];;
  gap> L3 := [ AbelianLieAlgebra( GF(2), 3 ) ];;

```

Any 3-dimensional non-abelian nilpotent Lie algebra is an immediate descendant of a 2-dimensional Lie algebra. So we compute the step-1 descendants of L1[1] and append them to L3.

```
──────────────────────────────── Example ────────────────────────────────

  gap> Append( L3, Descendants( L2[1], 1 ));
  gap> L3;
  [<Lie algebra of dimension 3 over GF(2)>,
  <Lie algebra of dimension 3 over GF(2)> ]]

```

Now we compute the list of 4-dimensional Lie algebras. First we set L4 to contain the 4-dimensional abelian Lie algebra. Then we compute the step-1 descendants of the 3-dimensional algebras and append these descendants to L4.

```
──────────────────────────────── Example ────────────────────────────────

  gap> L4 := [ AbelianLieAlgebra( GF(2), 4 ) ];;
  gap> for i in L3 do
  gap> Append( L4, Descendants( i, 1 ));
  gap> od;
  gap> L4;
  [[ <Lie algebra of dimension 4 over GF(2)>,
  <Lie algebra of dimension 4 over GF(2)>,
  <Lie algebra of dimension 4 over GF(2)> ]]

```

We continue this way up to dimension 7.

```
                              Example
 gap> L5 := [ AbelianLieAlgebra( GF(2), 5 ) ];;
 gap> for i in L3 do
 gap> Append( L5, Descendants( i, 2 ));
 gap> od;
 gap> for i in L4 do
 gap> Append( L5, Descendants( i, 1 ));
 gap> od;
 gap> L6 := [ AbelianLieAlgebra( GF(2), 6 ) ];;
 gap> for i in L3 do
 gap> Append( L6, Descendants( i, 3 ));
 gap> od;
 gap> for i in L4 do
 gap> Append( L6, Descendants( i, 2 ));
 gap> od;
 gap> for i in L5 do
 gap> Append( L6, Descendants( i, 1 ));
 gap> od;
 gap> L7 := [ AbelianLieAlgebra( GF(2), 6 ) ];;
 gap> for i in L4 do
 gap> Append( L7, Descendants( i, 3 ));
 gap> od;
 gap> for i in L5 do
 gap> Append( L7, Descendants( i, 2 ));
 gap> od;
 gap> for i in L6 do
 gap> Append( L7, Descendants( i, 1 ));
 gap> od;
 gap> Length( L7 );
 202
 gap>
```

This computation shows that there are 202 pairwise non-isomorphic nilpotent Lie algebras over $\mathbb{F}_2$.

Let us compute the automorphism group of a nilpotent Lie algebra from our list. We compute this automorphism group in the hybrid format used by Sophus, then we compute this group as a standard GAP object.

```
                              Example
 gap> AutomorphismGroupOfNilpotentLieAlgebra( L7[100] );
 rec( glAutos := [  ],
   agAutos := [ Aut: [ v.1, v.1+v.2, v.3, v.4, v.5, v.5+v.6, v.7 ],
       Aut: [ v.1, v.2+v.3, v.3, v.4, v.5, v.6, v.7 ],
       Aut: [ v.1+v.3, v.2, v.3, v.4+v.5, v.5, v.6+v.7, v.7 ],
       Aut: [ v.1+v.4, v.2, v.3+v.5, v.4+v.6, v.5+v.7, v.6+v.7, v.7 ],
       Aut: [ v.1, v.2+v.4, v.3, v.4+v.5, v.5, v.6+v.7, v.7 ],
       Aut: [ v.1+v.5, v.2, v.3, v.4+v.7, v.5, v.6, v.7 ],
       Aut: [ v.1, v.2+v.5, v.3, v.4, v.5, v.6, v.7 ],
       Aut: [ v.1+v.6, v.2, v.3, v.4+v.7, v.5, v.6, v.7 ],
```

```
        Aut: [ v.1, v.2+v.6, v.3, v.4+v.7, v.5, v.6, v.7 ],
        Aut: [ v.1+v.7, v.2, v.3, v.4, v.5, v.6, v.7 ],
        Aut: [ v.1, v.2+v.7, v.3, v.4, v.5, v.6, v.7 ],
        Aut: [ v.1, v.2, v.3+v.7, v.4, v.5, v.6, v.7 ] ], glOrder := 1,
  glOper := [  ], agOrder := [ 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 ],
  liealg := <Lie algebra of dimension 7 over GF(2)>,
  one := Aut: [ v.1, v.2, v.3, v.4, v.5, v.6, v.7 ], size := 4096,
  field := GF(2), prime := 2 )
gap>
gap> AutomorphismGroup( L7[100] );
<group with 12 generators>
gap>
```

Finally let us check that two Lie algebras from our list are not isomorphic.

——————————————— Example ———————————————

```
gap> AreIsomorphicNilpotentLieAlgebras( L7[100], L7[101] );
false
```

# Chapter 3

# Sophus functions

## 3.1 Some general functions to compute with Lie algebras

### 3.1.1 SophusTest

◊ SophusTest( )           (function)

Tests Sophus functions, returns true if it finds no mistakes, and returns false otherwise. May take a couple of minutes to complete.

### 3.1.2 IsLieNilpotentOverFp

◊ IsLieNilpotentOverFp( L )           (property)

Returns true if L is a nilpotent Lie algebra and its underlying field is a finite prime field.

### 3.1.3 MinimalGeneratorNumber

◊ MinimalGeneratorNumber( L )           (attribute)

Computes the minimal number of generators for $L$, which is the dimension of $L/L'$.

### 3.1.4 AbelianLieAlgebra

◊ AbelianLieAlgebra( F, d )           (function)

Returns the Abelian Lie algebra with dimension $d$ over the field F.

## 3.2 Functions to compute with nilpotent bases

### 3.2.1 NilpotentBasis

◊ NilpotentBasis( L )           (attribute)

Computes a nilpotent basis for $L$. Nilpotent bases are defined in Section 1.

### 3.2.2  LieNBWeights

◊ LieNBWeights( B )                                                            (attribute)

Every element of the nilpotent basis *B* has a weight; See Section 1. This function returns the list of these weights.

### 3.2.3  LieNBDefinitions

◊ LieNBDefinitions( B )                                                        (attribute)

This function returns a list. The i-th element of this list is 0 if B[i] has weight 1. Otherwise the i-th element is [k,l] if the definition of B[i] is [B[k],B[l]]. See Section 1.

### 3.2.4  IsNilpotentBasis

◊ IsNilpotentBasis( B )                                                        (property)

Returns true if the basis B of a Lie algebra was computed with the function NilpotentBasis; false otherwise.

### 3.2.5  IsLieAlgebraWithNB

◊ IsLieAlgebraWithNB( L )                                                      (property)

Returns true if a nilpotent basis for L has already been computed using the function NilpotentBasis; false otherwise.

## 3.3  The cover

### 3.3.1  LieCover

◊ LieCover( L )                                                                (attribute)

Computes the cover for the nilpotent Lie algebra *L* as defined in Section 1.

### 3.3.2  CoverHomomorphism

◊ CoverHomomorphism( C )                                                       (attribute)

The nilpotent Lie algebra C was obtained from a nilpotent Lie algebra L using the LieCover( L ) function call. This function returns the natural homomorphism from C onto L.

### 3.3.3  CoverOf

◊ CoverOf( C )                                                                 (attribute)

The nilpotent Lie algebra C was obtained from a nilpotent Lie algebra L using the LieCover( L ) function call. This function returns L.

### 3.3.4  IsLieCover

◇ `IsLieCover( C )` <span style="float:right">(property)</span>

Returns `true` if the Lie algebra `C` was obtained as the Lie cover of another Lie algebra `L` using the `LieCover( L )` function call.

### 3.3.5  LieMultiplicator

◇ `LieMultiplicator( C )` <span style="float:right">(attribute)</span>

The nilpotent Lie algebra `C` was obtained from a nilpotent Lie algebra `L` using the `LieCover( L )` function call. This function returns the central ideal of `C` which is the multiplicator of `L`; see Section 1.

### 3.3.6  LieNucleus

◇ `LieNucleus( C )` <span style="float:right">(attribute)</span>

The nilpotent Lie algebra `C` was obtained from a nilpotent Lie algebra `L` using the `LieCover( L )` function call. This function returns the central ideal of `C` which is the nucleus of `L`; see Section 1.

## 3.4  Automorphisms of nilpotent Lie algebras

We define a special class of automorphisms for our work.

### 3.4.1  NilpotentLieAutomorphism

◇ `NilpotentLieAutomorphism( L, gens, imgs )` <span style="float:right">(method)</span>

`L` is a nilpotent Lie algebra, `gens` is a generating set, and `imgs` is a subset of `L` with the same length as `gens`. Returns the automorphism of `L` which maps the element of `gens` to the elements of `imgs`. It is the responsibility of the user to make sure that the arguments are given so that the automorphism exists. These automorphisms can be compared, multiplied using the `*` sign, and the inverse of such an automorphism can also be computed in the usual manner.

### 3.4.2  IdentityNilpotentLieAutomorphism

◇ `IdentityNilpotentLieAutomorphism( L )` <span style="float:right">(method)</span>

`L` is a nilpotent Lie algebra; returns the identity automorphism of *L*.

### 3.4.3  IsNilpotentLieAutomorphism

◇ `IsNilpotentLieAutomorphism( A )` <span style="float:right">(property)</span>

Returns `true` if `A` was obtained using a `NilpotentLieAutomorphism` or an `IdentityNilpotentLieAutomorphism` function call.

## 3.5 Automorphism group and isomorphism testing

### 3.5.1 AutomorphismGroup

◊ AutomorphismGroup( L )                                              (method)

L is a nilpotent Lie algebra; returns the automorphism group of L as a group generated by GAP algebra automorphisms. The automorphism group is computed as explained in [Sch].

### 3.5.2 AutomorphismGroupNilpotentLieAlgebra

◊ AutomorphismGroupNilpotentLieAlgebra( L )                           (method)

L is a nilpotent Lie algebra; returns the automorphism group of L in the internally used hybrid format. The automorphism group is computed as explained in [Sch]. The hybrid format, which is very similar to the one used in [EO], is a record that contains the following fields.

- glAutos: a set of automorphisms which together with agAutos generate the automorphism group;

- glOrder: an integer whose product with the numbers in agOrder gives the size of the automorphism group;

- agAutos: a polycyclic generating sequence for a soluble normal subgroup of the automorphism group;

- agOrder: the relative orders corresponding to agAutos;

- liealg: The Lie algebra acted upon by the automorphisms.

- size: the size of the automorphism group.

- field: the underlying field of the Lie algebra.

- prime: the characteristic of the underlying field.

We do not return an automorphism group in the standard form because we wish to distinguish between agAutos and glAutos; the latter act non-trivially on the derived quotient of *L*. This hybrid-group description of the automorphism group permits more efficient computations with it.

### 3.5.3 AreIsomorphicNilpotentLieAlgebras

◊ AreIsomorphicNilpotentLieAlgebras( L, K )                           (method)

Returns true if L and K are isomorphic; false otherwise.

## 3.6 Descendants

### 3.6.1 Descendants

◊ Descendants( L, step )                                              (method)

Returns the step-step descendants of a nilpotent Lie algebra L.

### 3.6.2 DescendantsOfStep1OfAbelianLieAlgebra

◊ DescendantsOfStep1OfAbelianLieAlgebra( L, step )                    (method)

Returns the 1-step descendants of the abelian Lie algebra with dimension d defined over the field of p elements.

## 3.7 Input and output

The package provides with a number of functions that can be used to store lists of Lie algebras. Here we document only the most important ones, see the source code io.gi for the rest.

### 3.7.1 WriteLieAlgebraToString

◊ WriteLieAlgebraToString( L )                                       (function)

Returns a string that encodes the nilpotent Lie algebra L

### 3.7.2 ReadStringToNilpotentLieAlgebra

◊ ReadStringToNilpotentLieAlgebra( string, p, d )                    (function)

Decodes string into a d-dimensional nilpotent Lie algebra defined over the field of p elements.

### 3.7.3 WriteLieAlgebraListToFile

◊ WriteLieAlgebraListToFile( list, name, file )                      (function)

list is a list of nilpotent Lie algebras. Encodes each Lie algebra in list to a string. The list so obtained is written into file. The name of this list will be name.

### 3.7.4 SophusBuildManual

◊ SophusBuildManual( )                                               (function)

Builds Sophus manual.

### 3.7.5 SophusBuildManualHTML

◊ SophusBuildManualHTML( )                                           (function)

Builds Sophus manual in html format.

# References

[EO]     Bettina Eick and Eamonn A. O'Brien. AutPGrp,. A GAP 4 package.  6, 13

[O'B90] E. A. O'Brien.  The *p*-group generation algorithm.  *J. Symbol. Comput.*, 9(5–6):677–698, 1990.  5

[Sch]    Csaba Schneider. A computer-based approach to the classification of nilpotent Lie algebras. arxiv.org/math.RA/0406365.  5, 6, 13

[VBS]    Richard Rossmanith Victor Bovdi, Alexander Konovalov and Csaba Schneider.  LAGUNA, Lie AlGebras and UNits of group Algebras.  A GAP 4 package.  6