Int. J. Open Problems Comput. Math. , Vol. 5, No. 1, March, 2012
ISSN 2074-2827; Copyright ©ICSRS Publication, 2012
www.i-csrs.org

Application of Statistical Techniques for Comparing Lie Algebra Algorithms

D. Fernández-Ternero[1], J. Núñez[1], and A. F. Tenorio[2]

[1]Dpto. Geometría y Topología. Universidad de Sevilla
e-mail: {desamfer, jnvaldes}@us.es
[2]Dpto. Economía, Métodos Cuantitativos e H.ª Económica.
Universidad Pablo de Olavide
e-mail: aftenorio@upo.es

Abstract

This paper is devoted to study and compare two algebraic algorithms related to the computation of Lie algebras by using statistical techniques. These techniques allow us to decide which of them is more suitable and less costly depending on several variables, like the dimension of the considered algebra.

Keywords: *Statistical techniques, algorithm, Lie algebras, computing time, complexity.*
2000 Mathematics Subject Classification: *17B30, 68W40, 68Q25.*

1 Introduction

In this paper, several statistical techniques are used to decide on the feasibility of two algebraic algorithms. More concretely, the main goal is to carry out a statistical study starting from the computational data obtained when implementing and running these two algorithms in MAPLE to compute the laws of two families of solvable Lie algebras. Indeed, these data are structured according to the following two items: the computing time and the complexity of these algorithms.

So we are interested in knowing whether there exists some type of dependence between computing time and used memory. In the affirmative case, we would like to determine the order of this dependence (linear, quadratic, cubic, etc.) with respect to a third set of data: the dimensions of the algebras, for instance.

To do so, the two algorithms (previously obtained and implemented in Benjumea et al. (2009) and Núñez and Tenorio (2010)) are schematically compared to show the differences existing between them. Mainly, these differences are related to both the computing time and the used memory when running the two implementations.

We are using statistical techniques to compare these algorithms because, in our opinion, these techniques can give us a valuable help to decide if the research method for dealing with a particular mathematical topic (in this case, Lie algebras) is appropriate to obtain notable results or, on the contrary, the efforts made in this study are not proportional to the results so obtained.

Moreover, it might be thought that we are comparing algorithms performing two essentially different tasks and, hence, comparing them would make absolutely no sense. However, this is not our intention. Which we try to get with this process is to study if it is significatively easier to obtain precise results in the case of nilpotent Lie algebras than in the case of solvable ones, which would be expected.

Then, getting back to Lie algebras, it is convenient to recall that the relation between Lie groups and Lie algebras has been a profusely studied topic in both Mathematics and, more concretely, Computer Algebra.

Let us note that the computational study of Lie algebras has been developed throughout the last four decades. Since the seventies of the 20th century, several authors have used different software and algorithms to study the structure of Lie algebras (Beck and Kolman, 1973, for instance). Moreover, the recent literature on this subject is even considering specialized computational packages, like MAGMA (de Graaf, 2005) or GAP (Draisma, 2003).

When studying Representation Theory for finite-dimensional Lie algebras, we can focus on representations by means of linear algebras, because there exist isomorphisms between a given Lie algebra and a specific linear algebra (see Ado Theorem in Jacobson, 1955). Indeed, some families of Lie algebras can be represented by matrices verifying some special properties. In this sense, every solvable Lie algebra is isomorphic to a subalgebra of the Lie algebra $\mathfrak{h}_n$, of $n \times n$ upper-triangular matrices, for some particular $n \in \mathbb{N}$ (see Theorem 9.11 in Fulton and Harris, 1991). Analogously, any nilpotent Lie algebra is isomorphic to a subalgebra of the Lie algebra $\mathfrak{g}_n$, of $n \times n$ strictly upper-triangular matrices, for some particular $n \in \mathbb{N} \setminus \{1\}$ (see Theorem 9.9 in Fulton and Harris, 1991).

So this paper compares two algorithms such that their outputs are the laws of the Lie algebras $\mathfrak{h}_n$ and $\mathfrak{g}_n$, taking into consideration that both algorithms are based on their respective usual associated Lie groups. As we have previously commented, both two algorithms were already introduced and explained in Benjumea et al. (2009) and Núñez and Tenorio (2010), and they are now computationally compared by using statistical techniques.

2 Preliminaries

First, we recall some notions and results about Lie groups and Lie algebras. For a general overview, the reader can consult Varadarajan (1984).

Given a finite-dimensional Lie group, its associated Lie algebra can be computed by linearizing the Lie group. This method can be consulted in Varadarajan (1984). Precisely, the algorithms shown here are based on its direct application. Next we briefly summarize this method and the consequent algorithms.

According to Theorem 9.11 in Fulton and Harris (1991), given a solvable Lie algebra, there exists $n \in \mathbb{N}$ such that this algebra is isomorphic to a subalgebra of the Lie algebra $\mathfrak{h}_n$, of $n \times n$ upper-triangular matrices. Analogously, if the given Lie algebra is nilpotent, then the isomorphism is between it and a subalgebra of the Lie algebra $\mathfrak{g}_n$, of $n \times n$ strictly upper-triangular matrices, for some $n \in \mathbb{N} \setminus \{1\}$ (see Theorem 9.9 in Fulton and Harris, 1991). The Lie groups H_n and G_n associated with these two algebras are formed by the $n \times n$ upper-triangular matrices shown in Table 1.

Table 1: Expression of the elements in the Lie groups G_n and H_n.

G_n	$g_n(x_{r,s}) = \begin{pmatrix} 1 & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & 1 & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$	where $x_{i,j} \in \mathbb{C}$.
H_n	$h_n(x_{r,s}) = \begin{pmatrix} e^{x_{1,1}} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & e^{x_{2,2}} & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & e^{x_{n-1,n-1}} & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & e^{x_{n,n}} \end{pmatrix}$	

3 Algorithms to obtain laws of Lie algebras

The algorithms, the computational study of which is given here, were programmed to compute the laws of the Lie algebras $\mathfrak{g}_n$ and $\mathfrak{h}_n$ in Núñez and Tenorio (2010) and Benjumea et al. (2009), respectively. Both of them were based on obtaining the Lie algebras of left-invariant smooth vector fields associated with the Lie groups G_n and H_n, respectively.

Both two were implemented with MAPLE and structured in five steps. The first four steps computed a basis for the algebra $\mathfrak{g}_n$ or $\mathfrak{h}_n$, according to the one studied in each case. The fifth step was devoted to compute the nonzero brackets in the law of the studied algebra with respect to the basis obtained in the previous four steps. In this way, the outputs of these algorithms are a basis of the Lie algebra and its law with respect to that basis. Let us note that a unique input is necessary for the algorithms to start all their computations, namely: the order of the matrices in the Lie group G_n or H_n, depending on each case. Each of the five steps are summarized in Table 2, bearing in mind that more comprehensive explanations can be consulted in Núñez and Tenorio (2010) for $\mathfrak{g}_n$ and in Benjumea et al. (2009) for $\mathfrak{h}_n$.

Table 2: Brief explanation of the steps in the algorithms.

Step	**L.A.** $\mathfrak{h}_n$ (Benjumea et al., 2009)	**L.A.** $\mathfrak{g}_n$ (Núñez & Tenorio, 2010)
1: Dimension of associated Lie group.	$\dim(H_n) = \frac{n(n+1)}{2}$	$\dim(G_n) = \frac{n(n-1)}{2}$
2: Matrix of associated Lie group.	$\begin{pmatrix} e^{x_{1,1}} & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & e^{x_{2,2}} & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & e^{x_{n-1,n-1}} & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & e^{x_{n,n}} \end{pmatrix}$	$\begin{pmatrix} 1 & x_{1,2} & \cdots & x_{1,n-1} & x_{1,n} \\ 0 & 1 & \cdots & x_{2,n-1} & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & x_{n-1,n} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$
3: 1-parameter subgroup $\varphi_{i,j}$ of associated Lie group.	The coordinates $x_{i,j}$ in Step 2 are replaced by $x_{r,s}(t) = \begin{cases} 0, & \text{if } (r,s) \neq (i,j); \\ t, & \text{if } (r,s) = (i,j). \end{cases}$	
4: Basis of Lie algebra.	• Computing the orbits associated with $\varphi_{i,j}$: $h_n(x_{r,s}) \cdot \varphi_{i,j}(t)$, $\forall (i,j) \in \{1,2,\dots,n\}^2$ such that $i \leq j$ | $g_n(x_{r,s}) \cdot \varphi_{i,j}(t)$, $\forall (i,j) \in \{1,2,\dots,n\}^2$ such that $i < j$ • Differentiating the coordinates of these orbits with respect to the parameter t. • Replacing t with 0 in these derivatives.	
5: Nonzero brackets according to basis in Step 4.	• Defining a procedure to compute the brackets with respect to the basis. • Deciding which brackets are nonzero with respect to the basis. • For each nonzero bracket, determining the resulting vector field of the basis.	

4 Computing time and complexity

This section shows and studies some differences appearing in the previously cited algorithms. Such differences are related to both the computing time and the memory used to carry out the computations according to the order n of the matrices involved. These computations have been carried out for matrices of order less than 15 and 12 for the Lie algebras $\mathfrak{g}_n$ and $\mathfrak{h}_n$, respectively. From here on, the respective algorithms are denoted by G and H. We have used a personal computer Intel Pentium IV with 2.5 GHz and 256 MB of RAM. In Table 3, some computational data are shown starting from the implementations considered in this paper for the algorithms G and H.

Table 3: Data obtained with the implementations.

Input: n **(order** n**)**	**Dim. of** $\mathfrak{g}_n$	**Comp. time**	**Used mem.**	**Dim. of** $\mathfrak{h}_n$	**Comp. time**	**Used mem.**
2	1	0 s	0 B	3	0.5 s	1.00 MB
3	3	0 s	832 KB	6	0.5 s	1.50 MB
4	6	0.25 s	1.31 MB	10	5 s	1.62 MB
5	10	1.3 s	1.44 MB	15	37.8 s	1.75 MB
6	15	7.8 s	1.50 MB	21	207.4 s	1.81 MB
7	21	37.2 s	1.62 MB	28	916 s	1.94 MB
8	28	144 s	1.69 MB	36	3427.9 s	2.06 MB
9	36	486.1 s	1.87 MB	45	11541.4 s	2.19 MB
10	45	1449.7 s	2.00 MB	55	33307.6 s	2.37 MB
11	55	3979.1 s	2.19 MB	66	97368.5 s	2.62 MB
12	66	10011.8 s	2.44 MB	78	262755.8 s	2.87 MB
13	78	23529.3 s	2.69 MB	91	—	—
14	91	51288.0 s	3.06 MB	105	—	—
15	105	110412.2 s	3.37 MB	120	—	—

Moreover, Table 3 shows that the computing time increases faster for the implementation of the algorithm H than the one of the algorithm G. By comparing these computational data with respect to the dimension of the algebras and the order of their matrices, we can observe in Figures 1 and 2 that the results corresponding to both the computing time and the used memory are quite a lot lower for G than for H. Both the computing time and the used memory increase remarkably due to Step 5 and, more concretely, to the decision-making procedures for determining the nonzero brackets and the basis vector resulting from each of these nonzero brackets.

Regarding the complexity of the algorithms G and H, we have considered the number of operations carried out in the worst case. In this way, the value 1 is assigned for both the commands `if` and `if else` when counting the operations in their respective implementations.

Figure 1: Comparison of the computing time and the used memory of algorithms G and H with respect to the dimension of the algebras.

Figure 2: Comparison of the computing time and the used memory of algorithms G and H with respect to the order of the matrices.

Besides, the big O notation has been used to express the complexity of each algorithm starting from their respective implementations. Let us recall that, given two functions $f, g : \mathbb{R} \to \mathbb{R}$, we can say that $f(x) = O(g(x))$ if and only if there exists $M \in \mathbb{R}^+$ and $x_0 \in \mathbb{R}$ such that $|f(x)| < M \cdot g(x)$, for all $x > x_0$.

Next we prove that the complexity is polynomial for both algorithms. In fact, Step 5 is the most computationally expensive for both of them. To do so, we summarize the computation of the complexity for each step in the algorithms, showing the difference for the complexity order in Step 5 too. Tables 4 and 5 show both the complexity order and the total number of performed computations for the algorithms G and H, respectively.

Table 4: Complexity of each step and total complexity of algorithm G.

	Complexity	Computations
Step 1	$O(1)$	3
Step 2	$O(n^2)$	$2 + \sum_{i=1}^{n(n-1)/2} 1 + \sum_{j=1}^{n}\left(\sum_{k=1}^{j-1} 1 + 1 + \sum_{k=1}^{n-j} 1\right)$
Step 3	$O(n^4)$	$\sum_{h=1}^{n-1} \sum_{i=h+1}^{n}\left(2 + \sum_{j=1}^{n}\left(\sum_{k=1}^{n} 1 + 1\right)\right)$
Step 4	$O(n^4)$	$\sum_{h=1}^{n-1} \sum_{i=h+1}^{n}\left(3 + \sum_{j=1}^{n}\sum_{k=j}^{n} 1\right) + \sum_{l=1}^{n(n-1)/2}\left(1 + \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} 1\right)$
Step 5	$O(n^6)$	$3 + \sum_{h=1}^{n(n-1)/2} 1 + \sum_{i=1}^{n(n-1)/2-1} \sum_{j=i+1}^{n(n-1)/2}\left(1 + 3 \cdot \sum_{k=1}^{n(n-1)/2} 1\right)$
Total	$O(n^6)$	

As we can see in Tables 4 and 5, the complexity order for Steps 1 to 4 does not depend on the algorithm considered. Indeed, the complexity is the same for each step, being of polynomial order 4 at most. Nevertheless, let us emphasize that an essential difference can be observed for the complexity order in Step 5: Although this is also polynomial for both algorithms, its order is quite a lot greater for H than for G.

Table 5: Complexity of each step and total complexity of algorithm H.

	Complexity	Computations
Step 1	$O(1)$	3
Step 2	$O(n^2)$	$6+\sum_{i=1}^{n(n+1)/2} 1+\sum_{j=1}^{n}\left(\sum_{k=1}^{j-1} 1+1+\sum_{k=1}^{n-j} 1\right)$
Step 3	$O(n^4)$	$\sum_{h=1}^{n}\sum_{i=h}^{n}\left(3+\sum_{j=1}^{n}\left(\sum_{k=1}^{n} 1+1\right)\right)$
Step 4	$O(n^4)$	$\sum_{h=1}^{n}\sum_{i=h}^{n}\left(3+\sum_{j=1}^{n}\sum_{k=j}^{n} 1\right)+\sum_{j=1}^{n}\left(3+\sum_{k=1}^{n-j} 1\right)+\sum_{l=1}^{n(n+1)/2}\left(1+\sum_{j=1}^{n}\sum_{k=j}^{n} 1\right)$
Step 5	$O(n^{14})$	$3+\sum_{h=1}^{n(n+1)/2} 1+A$
Total	$O(n^{14})$	

$$\text{where } A=\sum_{i=1}^{n(n+1)/2-1}\ \sum_{j=1}^{n(n+1)/2}\left(1+\sum_{k=1}^{n(n+1)/2} 1+\sum_{k=1}^{n(n+1)/2}\ \sum_{k=1}^{n(n+1)/2} 1\right).$$

5 Statistical study for the computational data

Next, by using the data in Table 3, a statistical study is achieved to check whether the computing time and the used memory obtained in a theoretical way match with the ones obtained empirically when running the algorithms.

In this way, some statistical variables are considered: the order of the matrices n, the computing times T1 and T2 and the used memories M1 and M2 for the algorithms G and H, respectively. Besides, two additional variables C1 and C2 are used to represent the complexity order of the algorithms G and H, respectively. Remember that these two orders were previously determined as the polynomials n^6 and n^{14}, respectively. The values for each variable can be seen in Table 6.

Regarding the computing time in comparison with the complexity order, a linear correlation analysis is performed. In Table 7, the Pearson coefficient R is shown, being very close to 1 for both algorithms and furthermore with a 2-tailed significance level lower than 0.01. Hence, the computing time in both algorithms fits quite well with a polynomial model similar to the complexity order for each algorithm.

When studying the used memory in each algorithm, the value of the Pearson coefficient is not as close to 1 as that obtained for the computing time (see Table 8).

Therefore, we consider advisable and suitable the study of the regression by using non-linear estimation models (including polynomial ones). The results obtained in this study are shown in Tables 9 and 10. As some models require that non-missing values are positive, we have ignored every null value of used memory to get the corresponding regression analysis.

Figures 3 and 4 represent the models with Pearson coefficient R being the closest to 1. The best models are linear, quadratic and cubic for the algorithm G; whereas the logarithmic, quadratic and cubic ones are the most appropriate for the algorithm H. For both cases, the equations of the compound, growth and exponential models which we obtained are similar.

Figure 3: Estimation models of used memory for the algorithm G.

Figure 4: Estimation models of used memory for the algorithm H.

For both algorithms, the regression model which better fits the used memory in function of the order n is the cubic model with null constant, because this gives the best possible value for the Pearson coefficient (i.e. $R = 1$). Consequently, the used memory also fits accurately with a polynomial model, although its order is less than that of the theoretical complexity of the algorithm. In this way, the behavior of the used memory is better than the one that can be theoretically expected.

Specifically, the estimated equations are $G(n) = 0.437n - 0.042n^2 + 0.002n^3$ and $H(n) = 0.680n - 0.086n^2 + 0.004n^3$. So we can use these equations to realize estimations of the used memory for values of the order greater than the considered values in Table 3. For example, for order 15 in the algorithm H, we can estimate that the needed memory is approximately 4.35 MB.

6 Some conclusions

As we pointed out in Introduction, our main goal is to take advantage of statistical techniques to decide which way is the most appropriate to study certain mathematical topics: Lie algebras, in our particular case. Indeed, we want to determine which algorithm in our study is the best.

Starting from the data obtained by using these statistical techniques when comparing the utility of the algorithms G and H to deal with Lie algebras, we can assert that there are really few significative differences between them with respect to their main technical characteristics.

Among these differences, we find that the computing time increases faster for H than for G; whereas this does not happen with the used memory and both algorithms have a similar behavior with respect to this variable.

Apart from that, both algorithms have a polynomial complexity, the order of which does not depend on the algorithm for its first four steps. However, when taking into account the complexity of the fifth step, the total complexity order is quite a lot greater for H than for G.

Finally, we can clearly assert that the cubic model with null constant is the regression which better fits the used memory for both algorithms in function of the order n. In this way, we can estimate the used memory for values of n not appearing in Table 3. Particularly, for $n = 15$, the used memory is almost 4.5 MB when running the algorithm H.

7 Open Problems

The final section of this paper is devoted to expound some problems still unsolved and to be tackled in the future. First, it would be necessary to find the reason why the theoretical behavior of the used memory for both algorithm is worse than the one determined by the empirical data.

The following fact should be also explained: Whereas the used memory for n in the algorithm H is nearly similar to the one for $n + 1$ in the algorithm G, their respective computing times are significatively different and no trivial similarity relation can be arisen from them.

Another interesting question is focused on determining an alternative algorithm for solvable Lie algebras in such a way that the complexity order decreases, not being more than the double of the complexity order in the nilpotent case; but preserving all the statistical properties and relations of the one studied here.

Finally, another question rises up from searching a complete view of the Lie-algebra representation. More concretely, an algorithm, analogous to the ones studied here, can be stated and implemented for non-solvable (and hence non-nilpotent) Lie algebras when considering the general linear algebra $\mathfrak{gl}(n)$, of $n \times n$ square matrices. Once the algorithm was implemented, we might wonder if it conserves some of the statistical relations among computing time, used memory, complexity order and input order, previously determined for the nilpotent and solvable cases.

ACKNOWLEDGEMENTS. The authors thank to Professor Jebril for his useful suggestions and comments.

References

[1] Beck RE, Kolman B, "Computers in Lie algebras I. Calculation of inner multiplicities", *SIAM J Appl Math*, Vol. 25, (1973), pp. 300–312.

[2] Benjumea JC, Núñez J, Tenorio AF, "Computing the law of a family of solvable Lie algebras", *Internat J Algebra Comput*, Vol. 19, (2009), pp. 337–345.

[3] de Graaf WA, "Classification of Solvable Lie Algebras", *Experiment Math* Vol. 14, (2005), pp. 15–25.

[4] Draisma J, "Constructing Lie algebras of first order differential operators", *J Symbolic Comput*, Vol. 36, (2003), pp.685–698.

[5] Fulton W, Harris J, *Representation Theory: A first course*, Springer-Verlag: New York, (1991).

[6] Jacobson N, "A note on automorphisms and derivations of Lie algebras", *Proc Amer Math Soc* Vol. 6, (1955), pp. 281–283.

[7] Núñez J, Tenorio AF, "A computational study of a family of nilpotent Lie algebras", *J. Supercomputing*, (2010), doi: 10.1007/s11227-010-0430-2.

[8] Varadarajan VS, *Lie Groups, Lie Algebras and Their Representations*, Springer: New York, (1984).

Table 6: Statistical variables.

Order (n)	**T1**	**M1**	**C1**	**T2**	**M2**	**C2**
2	0.0	0.0	64	0.5	1.00	16384
3	0.0	0.83	729	0.5	1.50	4782969
4	0.25	1.31	4096	5.0	1.62	268435456
5	1.3	1.44	15625	37.8	1.75	6103515625
6	7.8	1.50	46656	207.4	1.81	7.84E+010
7	37.2	1.62	117649	916.0	1.94	6.78E+011
8	144.0	1.69	262144	3427.9	2.06	4.40E+012
9	486.1	1.87	531441	11541.4	2.19	2.29E+013
10	1449.7	2.00	1000000	33307.6	2.37	1.00E+014
11	3979.1	2.19	1771561	97368.5	2.62	3.80E+014
12	10011.8	2.44	2985984	262755.8	2.87	1.28E+015
13	23529.3	2.69	4826809	—	—	—
14	51288.0	3.06	7529536	—	—	—
15	110412.2	3.37	11390625	—	—	—

Table 7: Linear correlation of computing time.

	T1 vs. C1	T2 vs. C2
Pearson Correlation	0.966	0.997
Sig. (2-tailed)	0.000	0.000
N	14	11

Table 8: Linear correlation of used memory.

	M1 vs. C1	M2 vs. C2
Pearson Correlation	0.821	0.698
Sig. (2-tailed)	0.000	0.017
N	14	11

Table 9: Regression analysis of used memory for the algorithm G.

	Model Summary		Parameter Estimates		
Equation	R Square	Significance	b1	b2	b3
Linear	0.991	0.000	0.216		
Logarithmic	0.978	0.000	0.976		
Inverse	0.481	0.006	9.218		
Quadratic	0.992	0.000	0.248	-0.003	
Cubic	**0.999**	0.000	0.437	-0.042	0.002
Compound	0.972	0.000	1.076		
Power	0.906	0.000	0.323		
S	0.283	0.050	2.429		
Growth	0.972	0.000	0.074		
Exponential	0.972	0.000	0.074		

Table 10: Regression analysis of used memory for the algorithm H.

	Model Summary		Parameter Estimates		
Equation	R Square	Significance	b1	b2	b3
Linear	0.966	0.000	0.261		
Logarithmic	0.993	0.000	1.071		
Inverse	0.487	0.012	6.282		
Quadratic	0.989	0.000	0.421	-0.017	
Cubic	**0.999**	0.000	0.680	-0.086	0.004
Compound	0.984	0.000	1.095		
Power	0.981	0.000	0.367		
S	0.327	0.052	1.771		
Growth	0.984	0.000	0.091		
Exponential	0.984	0.000	0.091		