DOI: 10.1515/auom-2016-0032
An. Şt. Univ. Ovidius Constanţa
VERSITA
Vol. 24(2),2016, 137–147

Computing abelian subalgebras for linear algebras of upper-triangular matrices from an algorithmic perspective

Manuel Ceballos, Juan Núñez and Ángel F. Tenorio

Abstract

In this paper, the maximal abelian dimension is algorithmically and computationally studied for the Lie algebra $\mathfrak{h}_n$, of $n \times n$ upper-triangular matrices. More concretely, we define an algorithm to compute abelian subalgebras of $\mathfrak{h}_n$ besides programming its implementation with the symbolic computation package MAPLE. The algorithm returns a maximal abelian subalgebra of $\mathfrak{h}_n$ and, hence, its maximal abelian dimension. The order n of the matrices $\mathfrak{h}_n$ is the unique input needed to obtain these subalgebras. Finally, a computational study of the algorithm is presented and we explain and comment some suggestions and comments related to how it works.

1 Introduction

The maximal abelian dimension of a given finite-dimensional Lie algebra $\mathfrak{g}$ is the maximum $\alpha(\mathfrak{g})$ among the dimensions of its abelian subalgebras. This topic has been previously studied by considering both abelian ideals (e.g., see [6, 13]) and abelian subalgebras (e.g. see [4, 5, 12]). Indeed, Ceballos et al. [8] studied both approaches at once. When ideals are taken into consideration, more restrictive hypotheses are necessary. In the present paper, we

Key Words: Maximal abelian dimension, solvable Lie algebra, algorithm.
2010 Mathematics Subject Classification: Primary 17B30, 68W30; Secondary 17–08.
Received: 16.09.2014
Revised: 30.09.2014
Accepted: 16.02.2015

study the least restrictive problem and we consider all the subalgebras of the Lie algebra $\mathfrak{g}$.

When computing the maximal abelian dimensions for finite-dimensional Lie algebras, it is very useful the Schur-Jacobson's upper bound for this value: given the matrix Lie algebra $M_n(\mathbb{K})$, of $n \times n$ square matrices over a field $\mathbb{K}$, the dimension of its abelian subalgebras is upper bounded by $\left\lfloor \frac{n^2}{4} \right\rfloor + 1$, where $\lfloor \cdot \rfloor$ denotes the floor function. Indeed, abelian subalgebra with this dimension can be found. The result was proved by Schur [11] over the complex number field $\mathbb{C}$ and, later, by Jacobson [10] over a general field $\mathbb{K}$. Since every finite-dimensional subalgebra can be faithfully represented by a subalgebra of $\mathcal{M}_n(\mathbb{C})$, the maximal abelian dimension of $\mathfrak{g}$ is upper bounded by $\left\lfloor \frac{n^2}{4} \right\rfloor + 1$.

Regarding this, we have previously studied in [2, 3] the maximal abelian dimension of the Lie algebra $\mathfrak{g}_n$, of $n \times n$ strictly upper-triangular matrices, giving an algorithmic procedure to reach an abelian subalgebra of dimension $\alpha(\mathfrak{g}_n)$ and showing that Schur-Jacobson's bound was not reached for these algebras.

In this paper, we study the maximal dimension of abelian subalgebras for the Lie algebra $\mathfrak{h}_n$, of $n \times n$ upper-triangular matrices. The exact value of the maximal abelian dimension $\alpha(\mathfrak{h}_n)$ was already computed by Ceballos et al. [7], showing that Schur-Jacobson's upper bound is reached and conjecturing the existence of, at most, two abelian subalgebras of dimension $\alpha(\mathfrak{h}_n)$ depending on the parity of n. Additionally, the algorithmic procedure given in [2] can be adapted and adjusted to the case of the Lie algebra $\mathfrak{h}_n$. In this way, the main goal of this paper consists in defining an algorithm which computes the value of $\alpha(\mathfrak{h}_n)$ and a list of abelian subalgebras, including those being of dimension $\alpha(\mathfrak{h}_n)$. Moreover, the algorithm has been implemented with the symbolic computation package MAPLE and computationally studied starting from running.

At this point, we should ask ourselves if the algorithm presented here provides some advantages over any other already existing in the literature. More concretely, Ceballos et al. [8] introduced an algorithm to compute all the abelian subalgebras (and ideals) of a given Lie algebra $\mathfrak{g}$ starting from its law. This algorithm also returned the value of $\alpha(\mathfrak{g})$ and the list of abelian subalgebras (and ideals) of dimension $\alpha(\mathfrak{g})$. However, the algorithm was designed and implemented to be run over any given Lie algebra and, hence, did not take advantage of properties and results previously known for particular types of Lie algebras. Indeed, the algorithm presented in [8] only runs for Lie algebras $\mathfrak{h}_n$ with $n \leq 5$ (i.e. a Lie algebra of dimension 15), appearing computational problems in practice for higher values of n (since $\mathfrak{h}_6$ is a Lie algebra of dimension 21). The algorithm which we are presenting in this paper, allows us

the computation of the maximal abelian dimension and an abelian subalgebra of this dimension for the Lie algebra $\mathfrak{h}_n$ with $n < 2000$.

Lie algebra $\mathfrak{h}_n$ is interesting from both theoretical and applied viewpoints. From a theoretical perspective, every finite-dimensional solvable Lie algebra is isomorphic to a suitable subalgebra of $\mathfrak{h}_n$ for some $n \in \mathbb{N}$ [14, Proposition 3.7.3]. With respect to its applications, we must take into consideration that solvable Lie algebras are very useful for studying problems related to black holes and for giving an alternative description of the scalar manifold in a broad class of supergravity theories (e.g. [1, 9]).

2 Preliminaries

Some preliminary notions on Lie algebras are recalled in this section, bearing in mind that the reader can consult [14] for a general overview on solvable Lie algebras. From here on, we only consider finite-dimensional Lie algebras over the complex number field $\mathbb{C}$.

Given a Lie algebra $\mathfrak{g}$, its maximal abelian dimension, $\alpha(\mathfrak{g})$, is the maximum among the dimensions of its abelian subalgebras.

An abelian subalgebra (indeed, an ideal) of $\mathfrak{g}$ is given by the center $\mathrm{Cen}(\mathfrak{g})$, defined as

$$\mathrm{Cen}(\mathfrak{g}) = \{\boldsymbol{v} \in \mathfrak{g} | [\boldsymbol{v}, \boldsymbol{u}] = 0, \forall \boldsymbol{u} \in \mathfrak{g}\}.$$

Obviously, $\mathrm{Cen}(\mathfrak{g})$ is contained in any abelian subalgebra of dimension $\alpha(\mathfrak{g})$.

Throughout the paper, we work with the solvable Lie algebra $\mathfrak{h}_n$, consisting of the $n \times n$ upper-triangular matrices with the following form:

$$h_n(x_{r,s}) = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ 0 & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & x_{nn} \end{pmatrix}$$

where $n \in \mathbb{N}$ and $x_{ij} \in \mathbb{C}$, for all i, $j \in \mathbb{N}$, with $1 \leq i \leq j \leq n$. We can obtain a basis $\mathcal{B}_n$ of $\mathfrak{h}_n$ by considering the vectors $X_{ij} = h_n(x_{r,s})$ such that:

$$x_{r,s} = \begin{cases} 1, & \text{if } (r,s) = (i,j), \\ 0, & \text{if } (r,s) \neq (i,j), \end{cases}$$

where $1 \leq i \leq j \leq n$. Hence, the law of $\mathfrak{h}_n$ with respect to basis $\mathcal{B}_n$ is as follows

$$\begin{array}{ll} [X_{i,j}, X_{j,k}] = X_{i,k}, & \forall\, 1 \leq i < j < k \leq n; \\ [X_{i,i}, X_{i,j}] = X_{i,j}, & \forall\, 1 \leq i < j \leq n; \\ [X_{k,i}, X_{i,i}] = X_{k,i}, & \forall\, 1 \leq k < i \leq n. \end{array}$$

Note that $\mathrm{Cen}(\mathfrak{h}_n)$ is spanned by $\sum_{i=1}^{n} X_{i,i}$, which is the vector considering the whole main diagonal. Therefore, this vector has to belong to every abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$.

3 Algorithm to obtain abelian subalgebras

Next, we show an algorithm to compute abelian subalgebras of the Lie algebra $\mathfrak{h}_n$, including those of maximum dimension $\alpha(\mathfrak{h}_n)$. First, we give the general structure for the algorithm, stating and explaining its different steps in detail. Then, we show some examples of application for low values of n.

Given a Lie algebra $\mathfrak{h}_n$ for an arbitrary $n \in \mathbb{N}$, the algorithm only needs the value of n as input. Although we show its design starting from inserting the input, the parity of n has influence on both which is the last step before stopping and which is the structure of the abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$ returned as output. Indeed, this algorithm returns the subalgebras conjectured in [7] as the only ones existing in $\mathfrak{h}_n$ with dimension $\alpha(\mathfrak{h}_n)$.

The general reasoning is based on considering column-by-column the vectors in the basis $\mathcal{B}_n$ of $\mathfrak{h}_n$. Additionally, we must bear in mind that the vectors from the i^{th} column do not commute with those from the i^{th} row. Therefore, to avoid nonzero brackets, every vector coming from the i^{th} row has to be removed.

Step 1: n^{th} column.

Let consider the n vectors coming from the n^{th} column for the basis of the subalgebra. Hence, the vector $X_{n,n}$ coming from the n^{th} row must be removed, obtaining the abelian subalgebra $\mathrm{span}(X_{1,n}, \ldots, X_{n-1,n})$.

Step $n-i+1$**:** i^{th} column for $n > i \geq \left[\frac{n}{2}\right] + \xi_{odd}(n) + 1$, where ξ_{odd} is the indicator function of odd numbers.

There are i vectors coming from the i^{th} column. They are added to the generators obtained in the previous step, but removing the $2k-(i-1)$ vectors coming from the i^{th} row too. Consequently, the dimension of the abelian subalgebra increases $i-(n-(i-1)) = 2i-n-1$ with respect to the previous step. This dimension really increases if and only if $2i-n-1 > 0$. Since the previous inequality is equivalent to $i > \frac{n+1}{2} = \left[\frac{n}{2}\right] + \xi_{odd}(n)$, we can assure that $\left[\frac{n}{2}\right] + \xi_{odd}(n) + 1$ is the last column such that the dimension increases with this adding-and-removing procedure.

Step $\left[\frac{n}{2}\right]+1$**:** Add the vector $\sum_{i=1}^{n} X_{i,i}$ to the previous basis, obtaining

the abelian subalgebra of dimension $\left[\frac{n^2}{4}\right]+1$ spanned by the vectors:

$$\begin{matrix} X_{1,\left[\frac{n}{2}\right]+\xi_{odd}(n)+1} & \cdots & X_{1,n} \\ X_{2,\left[\frac{n}{2}\right]+\xi_{odd}(n)+1} & \cdots & X_{2,n} \\ \vdots & \ddots & \vdots \\ X_{\left[\frac{n}{2}\right]+\xi_{odd}(n),\left[\frac{n}{2}\right]+\xi_{odd}(n)+1} & \cdots & X_{\left[\frac{n}{2}\right]+\xi_{odd}(n),n} \end{matrix} \qquad \sum_{i=1}^{n} X_{i,i}$$

Let us note that, for $\mathfrak{h}_n$, we have obtained abelian subalgebras of dimension $\left[\frac{n^2}{4}\right]+1$. This dimension is the value of $\alpha(\mathfrak{h}_n)$ because it corresponds to Schur-Jacobson's upper bound and, hence, we cannot find abelian subalgebras of greater dimension. Moreover, our algorithm returns the abelian subalgebra that Ceballos et al. [7] conjectured as the only one of dimension $\alpha(\mathfrak{h}_n)$ when n is even. Even more, when n is odd, the algorithm returns one of the two abelian subalgebras conjectured as the only ones in the same reference. In fact, if we run Step $\left[\frac{n}{2}\right]+1$ of our algorithm for n odd, we obtain the other algebra conjectured. In this way, this algorithmic procedure would compute all the abelian subalgebras of dimension $\alpha(\mathfrak{h}_n)$ if the conjecture is proved.

To conclude this section, we want to show how works the algorithm for low values of the input n. Since the algebra $\mathfrak{h}_1$ is abelian, we assume $n \geq 2$ for the examples, although the algorithm also works for this value of n.

Example 3.1. *If $n = 2$, then Step 1 generates the abelian subalgebra* $\mathrm{span}(X_{1,2})$. *Step 2 adds the vector $X_{1,1} + X_{2,2}$ to obtain the abelian subalgebra* $\mathrm{span}(X_{1,2}, X_{1,1} + X_{2,2})$ *of dimension $\alpha(\mathfrak{h}_1) = 2$.*

Example 3.2. *If $n = 3$, then Step 1 considers the abelian subalgebra* $\mathrm{span}(X_{1,3}, X_{2,3})$, *consisting in taking the three vectors coming from the 3rd column and removing $X_{3,3}$ corresponding to the 3rd row. Then, Step 2 adds the vector $X_{1,1}+X_{2,2}+X_{3,3}$ and we obtain the abelian subalgebra* $\mathrm{span}(X_{1,3}, X_{2,3}, X_{1,1} + X_{2,2} + X_{3,3})$ *of dimension $\alpha(\mathfrak{h}_3)$.*

Let us assume that we run an additional step in the adding-and-removing procedure. In that case, we would add the vectors from the 2nd column, but removing all those coming from the 2nd row (including $X_{2,3}$ previously inserted). Hence, we would obtain the abelian subalgebra $\mathrm{span}(X_{1,2}, X_{1,3}, X_{1,1} + X_{2,2} + X_{3,3})$ *of dimension 3 as output.*

4 Implementing the algorithm with MAPLE

This section is devoted to explain a step-by-step implementation of the algorithm previously shown. This algorithm only needs a unique input; namely,

the order n of the matrices belonging to $\mathfrak{h}_n$. As outputs, the algorithm returns both the maximal abelian dimension $\alpha(\mathfrak{h}_n)$ and one abelian subalgebras of dimension $\alpha(\mathfrak{h}_n)$. According to the conjecture shown in [7], there exists only one for n even and two for n odd. Indeed, for n odd, if we run an additional step in the adding-and-removing procedure, our algorithm return also the second abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$.

To start running the implementation, three libraries are loaded: `linalg` (to activate some commands related to Linear Algebra), `ListTools` (to apply commands like `union` or `minus`) and `numtheory` (to use the command `iquo`).

The implementation consists of programming a main routine with two subroutines. The main routine `mas` (from maximum of abelian subalgebras) receives the order n of the matrices in $\mathfrak{h}_n$ and returns a basis of the abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$. Moreover, an implicit list with all the isomorphism classes of abelian subalgebras can be obtained starting from the second output of the routine.

The first subroutine, `add_mas`, determines the vectors to be added in each step to the basis of the abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$. Its input is a natural number j corresponding to the column considered in the routine `mas`. A list L is also defined as a local variable and its elements are the vectors to be added to the basis. The final output of this subroutine returns is the list L with the vectors considered.

```
> add_mas:=proc(j)
> local L;
> L:=[];
> for k from 1 to j do L:=[op(L),X[k,j]]; end do;
> return op(L[1..nops(L)]);
> end proc:
```

The second subroutine, `remove_mas`, determines which vectors have to be removed in each step of the algorithm. Starting from its two inputs (the natural numbers i and n), the subroutine removes the vectors coming from the i^{th} row. This subroutine needs a local variable M, which is a list saving the vectors to be removed in each step. This list is the final output of the subroutine.

```
> remove_mas:=proc(i,n)
> local M;
> M:=[];
> for k from i to n do M:=[op(M),X[i,k]]; end do;
> return op(M[1..nops(M)]);
> end proc:
```

Finally, the routine `mas` returns two different outputs: the basis of the abelian subalgebra of dimension $\alpha(\mathfrak{h}_n)$ and the value of $\alpha(\mathfrak{h}_n)$ itself. Only the order n of the matrices in $\mathfrak{h}_n$ is needed as input. To implement this routine, the local variables L, M, P, Q and i have to be defined. L and M are two sets formed by the vectors which have to be added and removed, respectively. The set difference $L \setminus M$ is saved in the variable P, whereas Q is a singleton with

the generator of $\mathrm{Cen}(\mathfrak{h}_n)$. In this way, the routine `mas` computes these sets in each step and returns both the union $P \cup Q$ and its cardinal; i.e. the maximal abelian dimension $\alpha(\mathfrak{h}_n)$ and the abelian subalgebra of that dimension.

```
> mas:=proc(n)
> local L,M,P,i,Q;
> L:={};M:={};P:={};i:=n;Q:={X[1,1]};
> while i>iquo(n,2) do
> L:={op(L),add_mas(i)};M:={op(M),remove_mas(i,n)};i:=i-1;P:=L minus M;
> end do;
> for j from 2 to n do Q:={op(Q)+X[j,j]}; end do;
> Q:=P union Q;
> return {Q,nops(Q)};
> end proc;
```

5 Computational study

To conclude this paper, we summarize some computational data related to the algorithm previously implemented. The implementation was programmed by using MAPLE 12 or higher and was run in an Intel Core 2 Duo T 5600 with a 1.83 GHz processor and 2.00 GB of RAM. Table 1 shows both the computing time and the used used memory to return the outputs in terms of the value n inserted as input.

Table 1: Computing time and used memory.

Input (n)	**dim**$(\mathfrak{h}_n)$	**Comp. time**	**Used memory**
2	3	0 s	0 MB
50	1275	0 s	0 MB
100	5050	0.046 s	3.31 MB
150	11325	0.109 s	5.69 MB
200	20100	0.266 s	7.56 MB
250	31375	0.562 s	11.88 MB
300	45150	0.969 s	15.19 MB
350	61425	1.609 s	16.31 MB
400	80200	2.453 s	18.75 MB
450	101475	3.641 s	24.06 MB
500	125250	5.468 s	40.99 MB
600	180300	9.796 s	63.55 MB
700	245350	16.407 s	101.8 MB
800	320400	27 s	157.04 MB
900	405450	42.205 s	271.45 MB
1000	500500	59.266 s	353.81 MB
1100	605550	83.876 s	423.55 MB
1250	781875	156.641 s	645.26 MB
1500	1125750	303.499 s	1071.24 MB
1750	1532125	522.061 s	1701.19 MB

As we can observe in Table 1, computations have been carried out for $n < 2000$, obtaining both the maximal abelian dimension $\alpha(\mathfrak{h}_n)$ and the abelian subalgebra of this dimension. The computing time is apparently doubling

when the order n increases fifty units. In this way, for $n = 1000$, the routine runs about 1 minute to compute the basis of a maximal abelian subalgebra of $\mathfrak{h}_{1000}$.

Let us note that the computational data shown in Table 1 provides significant improvements to the computation of the maximal abelian dimension of $\mathfrak{h}_n$ and its abelian subalgebras of this dimension. This is so because the algorithm introduced by Ceballos et al. [8] to compute both objects for an arbitrary Lie algebras provides worse computing times. In this way, for Lie algebras of dimension 11, that algorithm requires of running during 1 minute. Moreover, if we consider Lie algebras of dimension 13, the computing time increases up to 805 seconds. Table 1 shows that the computing time of the present algorithm is lower than 1 minute for $\mathfrak{h}_n$ with $n \leq 1000$ (i.e. the dimension is 500,500). Moreover, no value of n in Table 1 requires of computing time being higher that 525 seconds (and we are considering Lie algebras of dimension 1,532,125). Consequently, the algorithm shown here presents a lower complexity and is more efficient for carrying out the computations when considering a Lie algebra $\mathfrak{h}_n$.

For order closer to 2000, some problems appear in relation with an insufficient computational capacity for the computer. These problems entail that the implementation cannot finish the computations involved for n and the routine cannot return us the corresponding outputs. Indeed, these problems are motivated by the very high capacity of memory (almost 2 GB) needed for the computations. In any case, it does not suppose a serious problem because the dimension of the Lie algebra $\mathfrak{h}_{2000}$ is about $2 \cdot 10^6$ (which is not useful in practice).

Now, we show some brief statistics about the relationship between the computing time and the memory used by the algorithm to compute a maximal abelian subalgebra of $\mathfrak{h}_n$ in terms of the order n.

In Figure 1, we can observe how the used memory works with respect to the computing time, giving a graphic representation of their dependency relation. This is given by a very strong positive linear correlation.

Figure 2 shows how the computational time and the used memory is changing as a function of the matrix order n of $\mathfrak{h}_n$. Note that the used memory increases more quickly than the computational time. Moreover, both of them show an exponential-type behavior.

Finally, Figure 3 represents the frequency diagram for the quotients between used memory and computing time. In this case, the behavior of these quotients resembles a negative exponential function.

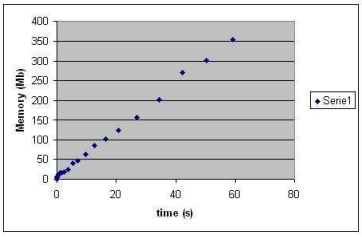

Figure 1: Memory used depending on computing time.

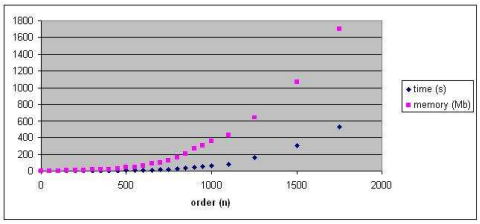

Figure 2: Computing time and used memory with respect to order n.

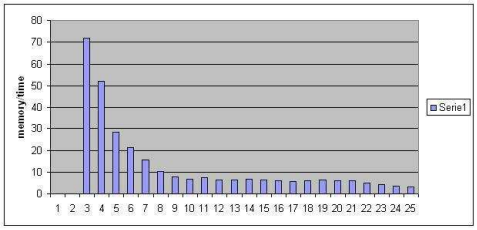

Figure 3: Frequency diagram for quotients used memory/computational time.

References

[1] L. Andrianopoli, R. D'Auria, S. Ferrara, P. Frè, M. Trigiante, R-R scalars, U-duality and solvable Lie algebras. *Nucl. Phys. B* **496** 617–629 (1997).

[2] J.C. Benjumea, F.J. Echarte, J. Núñez and A.F. Tenorio, An Obstruction to Represent Abelian Lie Algebras by Unipotent Matrices. *Extracta Math.* **19** 269–277 (2004).

[3] J.C. Benjumea, J. Núñez and A.F. Tenorio, The Maximal Abelian Dimension of Linear Algebras formed by Strictly Upper Triangular Matrices, *Theor. Math. Phys.* **152** 1225–1233 (2007).

[4] D. Burde and M. Ceballos, Abelian ideals of maximal dimension for solvable Lie algebras. *J. Lie Theory* **22** 741–756 (2012).

[5] R. Campoamor-Stursberg, Number of missing label operators and upper bounds for dimensions of maximal Lie subalgebras. *Acta Phys. Polon. B* **37** 2745–2760 (2006).

[6] C. Caranti, S. Mattarei and F. Newman, Graded Lie algebras of maximal class. *Trans. Amer. Math. Soc.* **349** 4021–4051 (1997).

[7] M. Ceballos, J. Núñez and A.F. Tenorio, The computation of abelian subalgebras in the Lie algebra of upper-triangular matrices. *An. St. Univ. Ovidius Constanta* **16** 59–66 (2008).

[8] M. Ceballos, J. Núñez and A.F. Tenorio, Algorithic method to obtain abelian subalgebras and ideals in Lie algebras. *Int. J. Comput. Math.* **89** 1388–1411 (2012).

[9] P. Frè, Solvable Lie algebras, BPS black holes and supergravity gaugings. *Fortschr. Phys.* **47** 173–181 (1999).

[10] N. Jacobson, Schur's Theorem on commutative matrices. *Bull. Amer. Math. Soc.* **50** 431–436 (1944).

[11] I. Schur, Zur Theorie vertauschbarer Matrizen. *J. Reine. Angew. Mathematik* **130** 66–76 (1905).

[12] A.F. Tenorio, Solvable Lie algebras and maximal abelian dimensions. *Acta Math. Univ. Comenian. (N.S.)* **77** 141–145 (2008).

[13] J.-L. Thiffeault and P.J. Morrison, Classification and Casimir invariants of Lie-Poisson brackets. *Phys. D* **136** 205–244 (2000).

[14] V.S. Varadarajan, *Lie Groups, Lie Algebras and Their Representations*, Springer, New York (1984).

Manuel CEBALLOS,
Departamento de Geometría y Topología,
Facultad de Matemáticas,
Universidad de Sevilla,
Aptdo. 1160, 41080 Sevilla, Spain.
Email: mceballos@us.es

Juan NÚÑEZ,
Departamento de Geometría y Topología,
Facultad de Matemáticas,
Universidad de Sevilla,
Aptdo. 1160, 41080 Sevilla, Spain.
Email: jnvaldes@us.es

Ángel F. TENORIO,
Departamento de Economía, Métodos Cuantitativos e Historia Económica,
Escuela Politécnica Superior,
Universidad Pablo de Olavide,
Ctra. Utrera Km. 1, 41013 Sevilla, Spain.
Email: aftenorio@upo.es