PUB. IRMA LILLE - 1987

Vol. 9 - n° IV

UN PROGRAMME POUR CALCULER LA COHOMOLOGIE DES ALGEBRES DE LIE

par

MACIAS VIRGOS Enrique

Résumé : Nous expliquons comment calculer le rang des groupes de cohomologie d'une algèbre de Lie, avec un algorithme qui est facile à programmer en Pascal.

Abstract : We explain how to compute the rank of the cohomology groups of a Lie algebra, with an algorithm easy to run as a Pascal program.

Mots clés : Lie algebras, Cohomology

Classification AMS : 17 B 56, 68-04

UN PROGRAMME POUR CALCULER LA COHOMOLOGIE DES ALGEBRES DE LIE

Enrique MACIAS VIRGOS

De nombreux résultats d'Algèbre et de Géométrie fournissent des méthodes théoriques pour calculer les groupes de cohomologie d'une algèbre de Lie (cf. par exemple, [1], [2] ou [3]).

Pourtant, en pratique, il existe essentiellement un seul algorithme combinatoire, dû à Koszul, qui permet ce calcul une fois connues les constantes de structure de l'algèbre de Lie. Nous le détaillerons ultérieurement.

Il n'est cependant utilisable que pour les petites dimensions. En effet, sa complexité a une croissance de l'ordre de $(2k)!/(k!)^2$, et la simplification des calculs dans certains cas particuliers n'est qu'illusoire car les hypothèses restent difficiles à vérifier.

Le but de cette note est de donner une version plus numérique de cet algorithme, et d'en fournir un programme pour micro-ordinateur, écrit dans le langage Pascal.

Je veux remercier le Laboratoire de Géométrie et Topologie pour son accueil pendant la réalisation de ce travail. Mon séjour à Lille a été possible grâce à une Bourse des Gouvernements espagnol et français. Je suis reconnaissant à D. Tanré, qui m'a donné des indications fort utiles, et à A. Matos, qui m'a expliqué quelques points d'Analyse Numérique.

1. Définitions et notations

1.1 Dans la suite, une algèbre de Lie est un espace vectoriel réel V, de dimension finie, muni d'un produit bilinéaire antisymétrique $[,]$ tel que $[x,[y,z]] + [y,[z,x]] + [z,[x,y]] = 0$ pour tout $x,y,z \in V$.

1.2 Soit $E_r(V^*)$ l'espace de toutes les applications réelles r-linéaires alternées $f(x_1,\ldots,x_r)$ avec $x_i \in V$. On a, en particulier:

$E_1 = V^*$ l'espace vectoriel dual de V;

$E_0 = R$ le corps des nombres réels;

$E_r = 0$ pour $r < 0$.

1.3 La différentielle extérieure d_r de E_r en E_{r+1} est donnée par

$$df(x_1,\ldots,x_{r+1}) = \sum_{i<j} (-1)^{i+j} f([x_i,x_j],x_1,\ldots,\hat{x}_i,\ldots,\hat{x}_j,\ldots,x_{r+1})$$

où les $\hat{}$ signifient que x_i, x_j doivent être éliminés.

Comme d'habitude, on pose $df=0$ pour les constantes $f \in E_0$.

1.4 Le résultat principal est $dd=0$, ou, ce qui est équivalent, $\mathrm{Im}d_{r-1} \subset \mathrm{Ker}d_r$; les groupes de cohomologie $H^r(V)$ de l'algèbre de Lie V sont alors définis comme les espaces quotients $\mathrm{Ker}d_r/\mathrm{Im}d_{r-1}$.

1.5 En fait, la somme directe $E(V^*)$ des E_r peut être munie d'une structure d'anneau, appelée algèbre extérieure de V.

Ici, il nous suffira de savoir que, pour tous les $f \in E_r$, $g \in E_p$ on peut définir leur produit $f \wedge g \in E_{r+p}$ de façon que:

(1.6) $$g \wedge f = (-1)^{rp} f \wedge g,$$

et que l'application linéaire d soit une dérivation d'algèbre:

(1.7) $$d(f \wedge g) = df \wedge g + (-1)^r f \wedge dg.$$

On appelle algèbre de cohomologie de V la somme directe H(V) des H^r, munie de la structure d'anneau induite.

2. Expressions locales

2.1 Soit $(x_1,\ldots,x_n)$, n=dimV, une base ordonée de V. Chaque crochet $[x_j,x_i]$ s'écrit de façon unique comme une combinaison linéaire $\Sigma\, s_{ij}^k x_k$. Les nombres réels s_{ij}^k, appelés constantes de structure, vérifient $s_{ii}^k=0$ et $s_{ij}^k=-s_{ji}^k$.

2.2 Considérons la base duale $(x_1^*,\ldots,x_n^*)$ de V^*, définie par

$$x_i^*(x_j) = \delta_{ij}.$$

Une application multilinéaire alternée $f\in E_r(V)$ étant complètement déterminée par ses valeurs sur les r-tuples $(x_{i_1},\ldots,x_{i_r})$ avec $1\leqslant i_1<\ldots<i_r\leqslant n$, on peut choisir comme base de E_r l'ensemble des éléments

$$x_{i_1}^* \wedge \ldots \wedge x_{i_r}^*$$

envoyant $(x_{j_1},\ldots,x_{j_r})$ sur $\det(\delta_{ij})$.

Dans toute la suite, E_r est supposé rapporté à cette base.

2.3 La dimension de $E_r(V^*)$ est alors $C_n^r = \dfrac{n!}{r!(n-r)!}$ si $0\leqslant r\leqslant n$, et 0 ailleurs.

Soit b_r la dimension de $H^r(V)$, $0\leqslant r\leqslant n$; il est clair que $b_0=1$.

Plus généralement, si on note D_r la matrice $C_n^{r+1} \times C_n^r$ associée à la différentielle d_r, on obtient:

(2.4) $$b_{r+1} = \dim(\mathrm{Ker} d_{r+1}) - \dim(\mathrm{Im} d_r) = C_n^{r+1} - \mathrm{rang} D_{r+1} - \mathrm{rang} D_r.$$

Nous verrons par la suite comment calculer les matrices D_r par itération.

3. La différentielle

3.1 Pour r=1, on constate facilement que $d(x_k^*)$ est précisément

(3.2) $$\sum_{i<j} s_{ij}^k \, (x_i^* \wedge x_j^*)$$

donc

(3.3) $$D_1 = \begin{bmatrix} s_{11}^1 & \cdots & s_{11}^n \\ \cdots & \cdots & \cdots \\ s_{n-1,n}^1 & \cdots & s_{n-1,n}^n \end{bmatrix} .$$

3.4 Pour $r \geqslant 2$, notons $s_{i_1 \cdots i_{r+1}}^{j_1 \cdots j_r}$ la composante en $x_{i_1}^* \wedge \cdots \wedge x_{i_{r+1}}^*$ de

(3.5) $$d(x_{j_1}^* \wedge \cdots \wedge x_{j_r}^*) =$$

$$= d(x_{j_1}^* \wedge \cdots \wedge x_{j_{r-1}}^*) \wedge x_{j_r}^* + (-1)^{r-1} x_{j_1}^* \wedge \cdots \wedge x_{j_{r-1}}^* \wedge dx_{j_r}^* ;$$

c'est-à-dire, s_a^b est la somme de deux termes, que nous noterons doréenavant s_G (pour gauche) et s_D (pour droite).

3.6 Observons d'abord que $s_G=0$ lorsque j_r n'apparaît pas dans la combinaison $i_1 \cdots i_{r+1}$. Dans le cas où j_r est un des éléments i_α, on a

(3.7) $$s_G = (-1)^{r+1-\alpha} s_{i_1 \cdots \hat{j}_r \cdots i_{r+1}}^{j_1 \cdots j_{r-1}} ;$$

c'est le produit d'un élément de D_{r-1} par la signature de la permutation $i_1 \cdots \hat{j}_r \cdots i_{r+1} j_r$.

3.8 D'autre part, s_D est toujours nul sauf si les $j_1 \cdots j_{r-1}$ appartiennent tous à $i_1 \cdots i_{r+1}$; dans ce cas-là on aura

(3.9) $$s_D = (-1)^{r-1} (-1)^{\alpha+\beta+1} s_{i_\alpha i_\beta}^{j_r} ;$$

c'est le produit d'un élément de D_1 (où i_α, i_β sont les indices qui restent), par la signature de la permutation

$$i_1 \cdots \hat{j}_1 \cdots \hat{j}_{r-1} \cdots i_{r+1} j_1 \cdots j_{r-1}.$$

4. L'algorithme

4.1 Pour éviter l'utilisation des combinaisons c du type $1 \leqslant i_1 < \ldots < i_r \leqslant n = \dim V$, on les identifiera avec l'entier positif

$$i(c) = (2^{i_1} + \ldots + 2^{i_r})/2. \tag{4.2}$$

4.3 Le poids r de la combinaison c est alors le nombre de chiffres 1 qui apparaissent dans l'expression binaire de i(c).

Réciproquement, chaque nombre entier p, $1 \leqslant p < 2^n$, correspond à une combinaison c(p).

4.4 Appelons (n:r) l'ensemble des entiers de poids r.

Pour $c \in (n:1)$ et $a = i_1 \ldots i_{r+1} \in (n:r+1)$ on définit la fonction

$$\sigma(c,a) = \begin{cases} (-1)^{r-\alpha+1} & \text{si c est égal à l'un des } i_\alpha \text{ de a,} \\ 0 & \text{ailleurs.} \end{cases} \tag{4.5}$$

Considérons maintenant $b = j_1 \ldots j_{r-1} \in (n:r-1)$ et définissons

$$\sigma(b,a) = \begin{cases} (-1)^{\alpha+\beta+r} & \text{si b est contenu dans a, et } i_\alpha, i_\beta \text{ sont les indices restants,} \\ 0 & \text{dans les autres cas.} \end{cases} \tag{4.6}$$

On établit sans difficulté la formule:

$$\sigma(b,a) = (-1)^{r(r+3)/2} \prod_{\nu=1}^{r-1} \sigma(j_\nu, a). \tag{4.7}$$

Finalement, si $c=j_r$, avec (3.7) et (3.9), on obtient une règle itérative permettant le calcul des éléments s_a^b des matrices D_r, $r \geqslant 2$:

$$(4.8) \qquad s_a^b = \sigma(c,a)\, s_{a-c}^{b-c} + (-1)^{r(r+3)/2} \prod_{\nu=1}^{r-1} \sigma(j_\nu,a)\, s_{a-b+c}^{c} \quad .$$

De cette façon, les manipulations combinatoires sont traitées dans notre programme comme des calculs numériques.

Remarquons aussi que, pour un $b \in (n:r)$ donné, les termes j_ν et l'entier i(b) sont obtenus essentiellement avec la même procédure.

5. Programme Pascal

5.1 Nous détaillons maintenant les procédures basiques à utiliser dans un programme pour micro-ordinateur, construit à partir de l'algorithme précédent.

L'Appendice fournit un listing complet d'un tel programme, écrit en langage Pascal.

5.2 Initial, Data: Les données telles que dimV et D_1 (3.3) sont lues à partir d'un fichier quelconque, y compris le clavier. Elles peuvent éventuellement être calculées par un autre programme (par exemple lorsque V est définie comme un ensemble de matrices).

5.3 Sign, Exp2, Log2, Last, Weight, Comb: Ce sont les fonctions élémentaires utilisées tout au long du programme. Pour $x \in Z$, on a $sign(x)=(-1)^x$, et $exp2(x)=2^x$. Last(x) calcule j_r à partir de $c(x)=j_1 \ldots j_r$; de façon que $Log2(x)=j_r+1$ soit l'exposant de la plus grande puissance de 2, inférieure ou égale à x. Le poids de x est Weight(x)=r. Comb(n,k) est le coefficient de Pascal C_n^k .

5.4 Ord, Place, Distribution: Avec les notations utilisées dans le paragraphe (4.), chaque élément s_a^b de D_r est indexé par des entiers (=combinaisons) qui ne correspondent pas à la notation habituelle ligne, colonne. C'est pour cela qu'on a prévu la procédure Distribution, qui construit une matrice auxiliaire définie par

place weight(x),ord(x) = x.

Ici, Ord(x) donne le numéro d'ordre correspondant à x parmi tous les entiers de même poids que x.

5.5 Signature, Sigma: Ce sont les fonctions σ définies dans (4.5), (4.6) et (4.7) pour comparer deux combinaisons.

5.6 Calcul, Algorithm: On utilise ici la formule (4.8). Chaque matrice D_r, $2 \leq r \leq n = \dim V$ est calculée de façon itérative en appliquant notre algorithme à D_{r-1} et D_1; ce sont donc les seules matrices à garder en mémoire à chaque étape.

5.7 Betti, Ecran, Sortie: Ces procédures donnent les dimensions b_r des groupes de cohomologie de V, en vertu de la formule (2.4).

5.8 Rank: Pour calculer les rangs des D_r, on a choisi une procédure basée sur la méthode classique de Gauss pour trianguliariser une matrice. A chaque étape, le programme cherche le pivot de la sous-matrice qui n'a pas encore été triangularisée, et il s'arrête au moment où les pivots des dernières lignes sont nuls.

Etant une procédure standard, nous ne la détaillerons pas.

5.9 Pour améliorer la vitesse du programme, les valeurs de Weight et Ord ne sont calculées qu'une seule fois, puis gardées dans des

variables Ordv et Weightv. On obtient ainsi, sur un HPVectra, des temps d'exécution de l'ordre d'une minute et demie pour dimV = 7 (dimension maximale possible pour une mémoire de 256 K).

5.10 On peut envisager de nombreuses améliorations du programme. Les plus intéressantes permettraient d'utiliser d'autres coefficients que R, voire d'exhiber des bases explicites de la cohomologie.

Références

[1] C. CHEVALLEY et S. EILENBERG : Cohomology theory of Lie groups and Lie algebras. Trans. Amer. Math. Soc. 63 (1948), 85-124.

[2] W. GREUB et alt. : Connections, Curvature and Cohomology. Vol. III. Academic Press 1976.

[3] J.L. KOSZUL : Homologie et cohomologie des algèbres de Lie. Bull. Soc. Math. France 78 (1950), 65-127.

Labo. Géométrie-Topologie. UFR Math.
Université de Lille-Flandres-Artois
59655 - Villeneuve d'Ascq (France)

Dpto. Xeometria e Topoloxia.
Univ. de Santiago de Compostela
C.U. Aptdo.280, 27000-Lugo (Espagne)

APPENDICE

```
Program CohomologyOfLieAlgebras;
(* E. Macias  January 1987 *)
const
     dimax=7;
     combmax=35;
type
     tableau=array[1..combmax,1..combmax] of real;
     supertableau=array[1..3] of tableau;
     vector=array[1..128] of integer;
     minitableau=array[1..dimax,1..combmax] of integer;
VAR
     dif: supertableau;
     dim: integer;
     place: minitableau;
     ordv,weightv: vector;

(*1.1 exponentielle base -1 *)
function sign(x:real):integer;
var
     y:integer;
begin
     y:=round(x);
     sign:=+1;
     if odd(y) then sign:=-1;
end;

(*1.2 exponentielle de base 2 *)
function exp2(x:integer):integer;
var
     i,e:integer;
begin
     e:=1;
     for i:=1 to x do e:=2*e;
     exp2:=e;
end;

(*1.3 le plus grand k tel que 2**k soit plus petit ou égal à x *)
function log2(x:integer):integer;
var
     k:integer;
begin
     k:=0;
     while exp2(k)<=x do k:=k+1;
     log2:=k-1;
end;

(*1.4*)
function last(x:integer):integer;
begin
     last:=exp2(log2(x));
end;
```

```
(*1.5 poids= nombre de 1 dans l'expression binaire*)
function weight(x:integer):integer;
label 1;
var
     w: integer;
begin
     w:=0;
     1;
     w:=w+1;
     if x>last(x) then
          begin
          x:=x-last(x);
          goto 1;
          end;
     weight:=w;
end;

(*1.6 nombre de combinaisons *)
function comb(n,k:integer):integer;
var
     i:integer;
     c:real;
begin
     c:=1;
     for i:=0 to k-1 do c:=c/(k-i)*(n-i);
     comb:=round(c);
end;

( 2.1 place de x dans les entiers de même poids  )
function ord(x,dim:integer):integer;
label 1;
var
     i,w,o:integer;
begin
     o:=0;
     w:=weight(x);
     for i:=1 to exp2(dim)-1 do
          begin
          if weight(i)=w then
               begin
               o:=o+1;
               if i=x then goto 1;
               end;
          end;
     1;
     ord:=o;
end;

(*2.2 *)
procedure distribution
(dimax,combmax:integer; var ordv,weightv:vector; var place:minitableau);
```

```
var
     q,r,s,t:integer;
begin
     writeln('patientez svp');
     for t:=1 to exp2(dimax)-1 do
          begin
          ordv[t]:=ord(t,dim);
          weightv[t]:=weight(t);
          end;
     for r:=1 to dimax do
     for s:=1 to combmax do
          place[r,s]:=0;
     for q:=1 to exp2(dimax)-1 do
          place[weightv[q],ordv[q]]:=q;
end;

(*2.3 appartenance d'une combinaison à une autre *)
function signature(cb1,cb2:integer):integer;
var
     p,i,c:integer;

          (*2.3.1 appartenence d'un chiffre à une combinaison *)
          function sigma(chif,cb:integer):integer;
          var
                j,d:integer;
          begin
          j:=weight(cb);
          d:=cb;
          while( (d>0) and (chif≠log2(d)+1) ) do
                begin
                j:=j-1;
                d:=d-last(d);
                end;
          if j=0 then sigma:=0 else sigma:=sign(weight(cb)-j);
          end;

begin
     p:=1;
     c:=cb1;
     for i:=1 to weight(cb1) do
          begin
          p:=p*sigma(log2(c)+1,cb2);
          c:=c-last(c);
          end;
     signature:=p;
end;

(*3.1 affecte 0 aux variables *)
procedure initial(var dif:supertableau; var dim:integer);
var
     i,j,k:integer;
begin
     for k:=1 to 3 do
     for i:=1 to combmax do
```

```
     for j:=1 to combmax do
     dif[k][i,j]:=0;
     write('dimension algèbre= ');
     readln(dim);
end;

(*3.2 entrée des constantes de structure*)
procedure data(dim:integer; var dif:supertableau);
var
     i,j,k,ligne,colonne:integer;
begin
     writeln('constantes  Bi,Bj = SOMM c(i,j,k) Bk');
     writeln('  ');
     for i:=1 to dim-1 do
     begin
     for j:=i+1 to dim do
          begin
          for k:=1 to dim do
          begin
                ( place de ij k dans dif[1] )
                ligne:=round( (exp2(i)+exp2(j))/2 );
                ligne:=ordv[ligne];
                colonne:=round( exp2(k)/2 );
                colonne:=ordv[colonne];

                ( graphisme )
                write('c(',i,',',j,',',k,')=');
                write(dif[1][ligne,colonne]:10:3,'?  ');
          read(dif[1][ligne,colonne]);
          dif[3][ligne,colonne]:=dif[1][ligne,colonne];
                DelLine;
                write('c(',i,',',j,',',k,')=');
                writeln(dif[1][ligne,colonne]:10:3);
          end;
          end;
     end;
end;


(*4 calcul du rang d'une matrice li x co*)
function rank(matrix:tableau; li,co:integer):integer;
type
     couplentiers=array[1..2] of integer; (*position d'un élément*)
var
     i,j:integer;
     ra:integer; (*marque de sous-matrice*)
     lipiv,copiv:integer; (*position du pivot*)
     piv:real; (*valeur du pivot*)
label 1;

(*4.1 à la recherche du pivot perdu d'une sous-matrice *)
procedure PosPiv(corner:integer; var lipiv,copiv:integer);
var
```

```
     i,j:integer;
     p:couplentiers;
begin
     p[1]:=corner;
     p[2]:=corner;
     for i:=corner to li do
     begin
          for j:=corner to co do
          if abs(matrix[ p[1],p[2] ])<=abs(matrix[i,j]) then
               begin
               p[1]:=i;
               p[2]:=j;
               end;
     end;
     lipiv:=p[1];
     copiv:=p[2];
end;

(*4.2 le pivot passe au coin superieur de la sous-matrice *)
procedure permut(var matrix:tableau; corner,lipiv,copiv:integer);
var
     i,j:integer;
          (*3.2.1*)
          procedure change(var x,y:real);
          var
               z:real;
          begin
               z:=x;
               x:=y;
               y:=z;
          end;

begin
     for j:=1 to co do
          change(matrix[corner,j],matrix[lipiv,j]);
     for i:=1 to li do
          change(matrix[i,corner],matrix[i,copiv]);
end;

(*4.3  calcul du rang *)
begin
     ra:=0;
     piv:=0;
     lipiv:=0;
     copiv:=0;
     1;
     if ( (ra<li) and (ra<co) ) then
     begin
          pospiv(ra+1,lipiv,copiv);
          permut(matrix,ra+1,lipiv,copiv);
          piv:=matrix[ra+1,ra+1];
          if abs(piv)>= 1e-10 then
```

```
          begin
                ra:=ra+1;
                for i:=1 to li do
                begin
                for j:=ra to co do
                matrix[i,j]:=matrix[i,j]- (matrix[i,ra]/piv)*matrix[ra,j];
                end;
          goto 1;
          end;
     end;
rank:=ra;
end;


(*0.1 expression de la différentielle extérieure *)
procedure algorithm(k,i,j:integer; var diff:tableau);
var
     left,right:integer;
     d:real;
begin
     d:=0;
     right:=signature(j-last(j),i);
     if right≠0 then
          d:= sign(k*(k+3)/2) * right *
              dif[1] [ ordv[i-j+last(j)],ordv[last(j)] ] ;
     left:=signature( last(j),i );
     if left≠0 then
          d:= d + left *
              dif[2] [ ordv[i-last(j)],ordv[j-last(j)] ] ;
     diff[ordv[i],ordv[j]]:=d;
end;

(*0.2 calcul de dif[k] à partir de dif[k-1]* )
procedure calcul(k,lignes,colonnes:integer; var dif:supertableau);
var
     i,j,l,c:integer;
begin
     for i:=1 to combmax do
     begin
          for j:=1 to combmax do
          begin
          dif[2][i,j]:=dif[3][i,j];
          dif[3][i,j]:=0;
          end;
     end;
     for i:=1 to lignes do
     begin
          l:=place[k+1,i];
          for j:=1 to colonnes do
          begin
          c:=place[k,j];
          algorithm(k,l,c,dif[3]);
```

```
          end;
     end;
end;


(*0.3 programme principal *)
var
     betti:array[1..dimax] of integer;
     k,lignes,colonnes,control:integer;

          (*0.3.1 *)
          procedure Ecran;
          begin
          writeln('rang= ',betti[k+1]);
          writeln('cont? pressez la touche ENTER');
          readln(control);
          writeln('patientez svp');
          end;

          (*0.3.2 *)
          procedure Sortie;
          begin
          betti[1]:=0;
          writeln('  ');
          writeln('b(0)= 1');
          for k:=1 to dim-1 do
                writeln('b(',k,')= ',comb(dim,k)-betti[k+1]-betti[k]);
          writeln('b(',dim,')= ',1-betti[dim]);
          end;

BEGIN
     initial(dif,dim);
     distribution(dimax,combmax;ordv,weightv,place);
     data(dim,dif);
     writeln('  ');
     for k:=1 to dim-1 do
          begin
          lignes:=comb(dim,k+1);
          colonnes:=comb(dim,k);
          if k>=2 then
          calcul(k,lignes,colonnes,dif);
          betti[k+1]:=rank(dif[3],lignes,colonnes);
          ecran;
          end;
     sortie;
END.
```