Computations for the Lie algebra $sl(2,\mathbb{C})$

Gilles Hunault

1. Structure of $sl(2,\mathbb{C})$

The set $sl(2,\mathbb{C})$ is composed of square matrices (called $sl-matrices$) of size 2 with complex coefficients and whose trace is 0 so such a matrix is defined by only 3 coefficients a, b and c. The first procedure we write is $sl2mat(a,b,c)$ which builds such a matrix, namely:

```
restart :
read(`r:/replie.prm`) :
sl2mat(a,b,c) ;
```

$$\begin{pmatrix} a & b \\ c & -a \end{pmatrix}$$

We can ask *Maple* to show the text of the procedure:

```
print(sl2mat) ;
```

```
proc(x,y,z) local slm :
slm := array(1..2,1..2,[[x,y],[z,-x]]) :
RETURN(evalm(slm)) :
end :  # proc sl2mat
```

The second procedure $ismatsl2(M)$ tells us if a square matrix M of size 2 belongs to $sl(2,\mathbb{C})$ (that is, if its trace is zero). Let's check for instance if the sum and the product of two sl-matrices is in $sl(2,\mathbb{C})$:

```
M := sl2mat(m1,m2,m3):
N := sl2mat(n1,n2,n3):
ismatsl2(M+N) ;
```

true

ismatsl2(M &* N);

$false$

The procedure *ismatsl*2 is defined by :

```
ismatsl2 := proc(x) :
RETURN(evalb(linalg[trace](x)=0)) :
end :  # proc ismatsl2
```

Next, let's check if multiplication of an *sl*-matrix by a scalar k gives another *sl*-matrix:

ismatsl2(k * M);

$true$

So we may consider $sl(2,\mathbb{C})$ as a vector space for the addition and the multiplication by a scalar. We define now the square bracket *sb* of two (normal) matrices of size 2 by the procedure:

```
sb := proc(a,b) ;
RETURN( evalm( (a &* b) - (b &* a) ) ) :
end :  # proc sb
```

and we can see that the image of any two matrices by this square bracket is in $sl(2,\mathbb{C})$:

ismatsl2(sb(M,N)) :

$true$

This operator is skew-symmetric ($[X, Y] = -[Y, X]$):

linalg[equal](sb(M,N) , -sb(N,M)) ;

$true$

bilinear:

linalg[equal](sb(M+P,N), sb(M,N) + sb(P,N)) ;

$true$

linalg[equal](sb(M,k*N), k*sb(M,N)) ;

$false$

linalg[equal](sb(M,k*N), map(expand,evalm(k*sb(M,N)))) ;

$true$

and it satisfies the Jacobi identity :

print(jacobi) ;

```
jacobi := proc(f,x,y,z) local p1,p2,p3 ;
p1 := f(x,f(y,z)) :
p2 := f(y,f(z,x)) :
p3 := f(z,f(x,y)) :
RETURN(linalg[equal](p1+p2+p3,sl2mat(0,0,0))) :
end :  # proc jacobi
```

jacobi(sb,M,N,P) ;

$true$

2. Representations of $sl(2,\mathbb{C})$

It can be proved that the irreducible representations of $sl(2,\mathbb{C})$ of dimension p have eigenvalues $-p, -p+2, ..., p-2, p$ and that all such irreducible representations are isomorphic to some $Sym^n(\mathbb{C}^2)$. Let's define $sl2eig(p)$ by

```
sl2eig := proc()
# valeurs propres de sym(ARG[1]) :
#     pour x pair   : -x, -x+2,.... 0 ...    x-2, ... x
#     pour x impair : -x, -x+2,.... -1,1 ... x-2, ... x
local argun,  # argument
      nbval,  # nombre de valeurs a renvoyer
      ldvp,   # tableau des valeurs
      v,      # indice dans ldvp
      vv:     # valeur courante dans ldvp
argun := args[1]         : nbval := argun + 1   :
ldvp  := array(1..nbval) : vv    := -argun - 2 :
for v from 1 to nbval do
 vv   := vv + 2 : ldvp[v] := vv :
od : # for v
RETURN(evalm(ldvp)) : end : # sl2eig
```

For instance, $Sym^3(\mathbb{C}^2)$ has eigenvalues

sl2eig(3) ;

$[-3, -1, 1, 3]$

The eigenvalues of $V \otimes W$ are all sums of both eigenvalues and we can compute them using the module $allf$ provided we define $plus$ as:

```
plus := (x,y) → x+y ;
```

For instance, we get the eigenvalues and their multiplicities of the representation $U = Sym^2(\mathbb{C}^2) \otimes Sym^3(\mathbb{C}^2)$ by:

U := allf(sl2eig(2), plus, sl2eig(3)) ;

$$\begin{bmatrix} -5 & -3 & -1 & 1 \\ -3 & -1 & 1 & 3 \\ -1 & 1 & 3 & 5 \end{bmatrix}$$

To decompose U into a direct sum of irreducible $Sym^k(\mathbb{C}^2)$, we have first to structure this array in two columns, the eigenvalues and their multiplicities. We do it by

T := multeig(U);

$$\begin{bmatrix} valp & mult \\ -5 & 1 \\ -3 & 2 \\ -1 & 3 \\ 1 & 3 \\ 3 & 2 \\ 5 & 1 \end{bmatrix}$$

Then we iterate the search of the biggest eigenvalue (say k) and the substraction of the multiplicities of eigenvalues of $Sym^k(\mathbb{C}^2)$ until all eigenvalues have been used. This leads to the procedure *decompirr*:

D := decompirr(T) ;

$[5\ 3\ 1\ 0\ 0\ 0\ 0\ ]$

We have added an option named '*detail*' that may help to understand how it works:

decompirr(T,'detail') ;

```
on retire V5
il reste
```

$$\begin{bmatrix} valp & mult \\ \text{-5} & 0 \\ \text{-3} & 1 \\ \text{-1} & 2 \\ 1 & 2 \\ 3 & 1 \\ 5 & 0 \end{bmatrix}$$

```
on retire V3
il reste
```

$$\begin{bmatrix} valp & mult \\ \text{-5} & 0 \\ \text{-3} & 0 \\ \text{-1} & 1 \\ 1 & 1 \\ 3 & 0 \\ 5 & 0 \end{bmatrix}$$

```
on retire V1
il reste
```

$$\begin{bmatrix} valp & mult \\ \text{-5} & 0 \\ \text{-3} & 0 \\ \text{-1} & 0 \\ 1 & 0 \\ 3 & 0 \\ 5 & 0 \end{bmatrix}$$

And finally we present the direct sum decomposition:

present(D) ;

```
   V 5
+  V 3
+  V 1
```

As a conslusion, we may sum up all these computations by defining the $Clebsh - Gordon$ procedure:

```
cg := proc(a,b) ;
present(
   decompirr(multeig(allf(sl2eig(a),plus,sl2eig(b))))
) ;
end :  # cg
```

so writing

```
cg(3,7) ;
```

gives

```
   V 10
+  V  8
+  V  6
+  V  4
```

which is quicker to type than

```
present(decompirr(multeig(allf(sl2eig(a),plus,sl2eig(b))))) ;
```

to get the same answer.