1 Permutaciones y grupos

En GAP, las permutaciones se pueden escribir de muchas formas. O bien como producto de ciclos disjuntos, o usando funciones específicas para crear permutaciones.

```
gap> MappingPermListList([1,2,3,4],[2,3,1,4]);
(1,2,3)
gap> PermList([2,3,1,4]);
(1,2,3)
gap> (1,2,3)(4,6);
(1,2,3)(4,6)
gap> ListPerm((1,2,3)(4,6));
[ 2, 3, 1, 6, 5, 4 ]
gap> PermList(last);
(1,2,3)(4,6)
```

El operador '*' se usa para la composición (hay que tener cuidado en el orden en que se compone). El operador '^' se puede usar para calcular la imagen de un elemento por una permutación.

```
gap> p:=(1,2,3)*(3,4);
(1,2,4,3)
gap> 3^p;
1
gap> (3,4)*(1,2,3);
(1,2,3,4)
```

El orden de una permutación y su signo se pueden calcular de la siguiente forma.

```
gap> SignPerm((1,2,3)(4,6));
-1
gap> Order((1,2,3)(4,6));
6
```

Podemos definir un grupo generado por varias permutaciones, calcular su orden y ver sus elementos, o comprobar si es abeliano (y muchas otras propiedades).

```
gap> g:=Group((1,2,3),(4,5));
Group([ (1,2,3), (4,5) ])
gap> Order(g);
6
gap> Elements(g);
[ (), (4,5), (1,2,3), (1,2,3)(4,5), (1,3,2), (1,3,2)(4,5) ]
gap> IsAbelian(g);
true
```

También podemos hacer cocientes de grupos.

```
gap> h:=Group((1,2,3));
Group([ (1,2,3) ])
gap> g/h;
Group([ f1 ])
gap> Elements(g/h);
[ <identity> of ..., f1 ]
gap> h:=Group((3,4));
Group([ (3,4) ])
gap> g/h;
Error, <N> must be a normal subgroup of <G>  ...
gap> h:=Group((4,5));
Group([ (4,5) ])
gap> g/h;
Group([ f1 ])
gap> Elements(g/h);
[ <identity> of ..., f1, f1^2 ]
```

Con la orden `IsCyclic` podemos saber si un grupo es cíclico.

```
gap> g:=Group((1,2,3),(4,5));;
gap> IsCyclic(g);
true
gap> DirectProduct(CyclicGroup(2),CyclicGroup(2));
<pc group of size 4 with 2 generators>
gap> IsCyclic(last);
false
```

Los grupos abelianos finitos también se pueden definir de la siguiente forma. Calculemos por ejemplo cuántos elementos de $\mathbb{Z}_2 \times \mathbb{Z}_4 \times \mathbb{Z}_{10}$ tienen orden 10.

```
gap> g:=AbelianGroup([2,4,10]);
<pc group of size 80 with 3 generators>
gap> Filtered(Elements(g),x->Order(x)=10);
[ f4, f1*f4, f1*f5, f3*f4, f3*f5, f4*f5, f1*f3*f4, f1*f3*f5, f1*f4*f5,
  f1*f5^2, f3*f4*f5, f3*f5^2, f1*f3*f4*f5, f1*f3*f5^2, f1*f5^3, f3*f5^3,
  f4*f5^3, f1*f3*f5^3, f1*f4*f5^3, f1*f5^4, f3*f4*f5^3, f3*f5^4, f4*f5^4,
  f1*f3*f4*f5^3, f1*f3*f5^4, f1*f4*f5^4, f3*f4*f5^4, f1*f3*f4*f5^4 ]
gap> Length(Filtered(Elements(g),x->Order(x)=10));
28
```

Los homomorfismos de grupos se pueden definir de varias formas. Ponemos aquí como ejemplo cómo hacerlo dando las imágenes de los generadores del dominio de la aplicación.

```
gap> g1:=Group((1,2,3),(4,5));
Group([ (1,2,3), (4,5) ])
gap> g2:=AbelianGroup([6]);
<pc group of size 6 with 1 generators>
gap> GeneratorsOfGroup(g1);
[ (1,2,3), (4,5) ]
gap> GeneratorsOfGroup(g2);
[ f1 ]
gap> f:=GroupHomomorphismByImages(g2,g1,[g2.1],[(1,2,3)(4,5)]);
[ f1 ] -> [ (1,2,3)(4,5) ]
gap> Image(f);
Group([ (1,2,3)(4,5), (1,3,2) ])
gap> Order(last);
6
gap> g1/Image(f);
Group([  ])
gap> Kernel(f);
Group([ <identity> of ..., <identity> of ... ])
gap> Order(last);
1
```

```
gap> Image(f,g2.1^2);
(1,3,2)
gap> PreImage(f,(1,2,3));
f2^2
gap> Elements(g2);
[ <identity> of ..., f1, f2, f1*f2, f2^2, f1*f2^2 ]
```

Por tanto la aplicación f es un isomorfismo de grupos.

2 Matrices

Las matrices en GAPse representan como una lista de listas, de forma que todas ellas tengan la misma longitud y el mismo tipo de elementos. Por tanto hay que tener cuidado con qué comandos son 'destructivos'. La suma, producto e inverso se calculan con las operaciones usuales.

```
gap> a:=[[1,2],[3,4]];;
gap> b:=[[5,6],[7,8]];;
gap> a+b;
[ [ 6, 8 ], [ 10, 12 ] ]
gap> a*b;
[ [ 19, 22 ], [ 43, 50 ] ]
gap> a*(-1);
[ [ -1, -2 ], [ -3, -4 ] ]
gap> a^0;
[ [ 1, 0 ], [ 0, 1 ] ]
```

Y el determinante

```
gap> Determinant(a);
-2
```

Para calcular la forma normal de Hermite sobre los enteros disponemos de la siguiente función.

```
gap> a:=[[1,2,3],[4,5,6]];
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
gap> HermiteNormalFormIntegerMat(a);
[ [ 1, 2, 3 ], [ 0, 3, 6 ] ]
```

Si queremos hacer el cálculo sobre los racionales, usamos `TriangulizeMat`, pero ojo, que esta función destruye el argumento que le pasamos.

```
gap> a:=[[1,2,3],[4,5,6]];;
 gap> TriangulizeMat(a);
gap> a;
[ [ 1, 0, -1 ], [ 0, 1, 2 ] ]
```

Si queremos hacer las cuentas sobre un cuerpo finito, podemos usar el comando `GF` (cuerpo de Galois), y luego embeber la matriz dada en dicho cuerpo con `One`.

```
gap> F:=GF(7);
GF(7)
gap> a:=[[1,2,3],[4,5,6]];;
gap> az7:=a*One(F);
[ [ Z(7)^0, Z(7)^2, Z(7) ], [ Z(7)^4, Z(7)^5, Z(7)^3 ] ]
gap> TriangulizeMat(az7);
gap> az7;
[ [ Z(7)^0, 0*Z(7), Z(7)^3 ], [ 0*Z(7), Z(7)^0, Z(7)^2 ] ]
gap> a;
[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]
gap> Int(Z(7));
3
```

Nótese que `a` no ha sido destruida, pues la operación `a*One(F)` genera una nueva matriz a partir de ella.

3 Sistemas de ecuaciones

Para encontrar una solución de un sistema de la forma $xA = b$ usando GAP, utilizamos el comando `SolutionMat` junto con `NullspaceMat`. El primero nos da una solución particular (en caso de existir) y el segundo nos da las soluciones del sistema homogéneo asociado.

```
gap> a:=[[1,2,3],[4,5,6]];; b:=[1,1,1];;
gap> SolutionMat(a,b);
[ -1/3, 1/3 ]
gap> NullspaceMat(a);
[  ]
```

Con lo que en este caso la solución es única.

```
gap> a:=[[1,2],[3,4],[5,6]];;b:=[1,1];;
gap> SolutionMat(a,b);
[ -1/2, 1/2, 0 ]
gap> NullspaceMat(a);
[ [ 1, -2, 1 ] ]
```

Y en este caso las soluciones son de la forma $(-\frac{1}{2},\frac{1}{2},0)+\lambda(1,-2,1)$. En caso de no existir solución recibimos un `fail` a cambio.

```
gap> a:=[[1,2,3],[4,5,6]];; b:=[1,1,2];;
gap> SolutionMat(a,b);
fail
```

También podemos hacer las cuentas sobre un cuerpo finito.

```
gap> a:=[[1,2],[3,4],[5,6]];;b:=[1,1];;
gap> F:=GF(5);;
gap> SolutionMat(a*One(F),b*One(F));
[ Z(5), Z(5)^3, 0*Z(5) ]
```

4 Espacios vectoriales

GAPya sabe que algunos de sus objetos son espacios vectoriales, y además podemos definir nuevos espacios vectoriales de diferentes formas.

```
gap> x:=X(Rationals,"x");
x
gap> r1:=PolynomialRing(Rationals,1);
PolynomialRing(..., [ x ])
gap> IsVectorSpace(r1);
true
gap> IsVectorSpace(Rationals^4);
true
gap> v:=VectorSpace(Rationals,[[1,2,3],[4,5,6]]);
gap> [3,3,3] in v;
```

```
true
gap> b:=Basis(v);
SemiEchelonBasis( <vector space over Rationals, with 2 generators>,
[ [ 1, 2, 3 ], [ 0, 1, 2 ] ] )
gap> BasisVectors(b);
[ [ 1, 2, 3 ], [ 0, 1, 2 ] ]
gap> Dimension(v);
2
```

También podemos calcular los coeficientes de un vector respecto de una base, o las combinaciones lineales de los elementos de una base.

```
gap> v:=VectorSpace(Rationals,[[1,2,3],[4,5,6]]);;
gap> b:=Basis(v);;
gap> Coefficients(b,[1,1,1]);
[ 1, -1 ]
gap> Coefficients(b,[1,2,1]);
fail
gap> LinearCombination(b,[1,1]);
[ 1, 3, 5 ]
```

La suma e intersección de subespacios es sencilla de calcular.

```
gap> v:=Rationals^4;;
gap> w:=Subspace(v,[[1,0,0,0],[0,1,0,0]]);;
gap> u:=Subspace(v,[[1,1,1,1],[2,2,0,0]]);;
gap> Intersection(w,u);
<vector space of dimension 1 over Rationals>
gap> BasisVectors(Basis(last));
[ [ 1, 1, 0, 0 ] ]
gap> u+w;
<vector space of dimension 3 over Rationals>
gap> BasisVectors(Basis(last));
[ [ 1, 1, 1, 1 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 1 ] ]
```

Si nuestro espacio vectorial es finito, podemos calcular todos sus subespacios.

```
gap>sz22:= Subspaces(GF(2)^2);
Subspaces( ( GF(2)^2 ) )
gap> List(last,w->BasisVectors(Basis(w)));
[ [  ], [ <an immutable GF2 vector of length 2> ],
  [ <an immutable GF2 vector of length 2> ],
  [ <an immutable GF2 vector of length 2> ],
  [ <an immutable GF2 vector of length 2>, <an immutable GF2 vector of length
        2> ] ]
gap> Length(last);
5
gap> List(sz22,Elements);
[ [ [ 0*Z(2), 0*Z(2) ] ], [ [ 0*Z(2), 0*Z(2) ], [ Z(2)^0, 0*Z(2) ] ],
  [ [ 0*Z(2), 0*Z(2) ], [ Z(2)^0, Z(2)^0 ] ],
  [ [ 0*Z(2), 0*Z(2) ], [ 0*Z(2), Z(2)^0 ] ],
  [ [ 0*Z(2), 0*Z(2) ], [ 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2) ],
      [ Z(2)^0, Z(2)^0 ] ] ]
```

Para definir una aplicación lineal entre subespacios podemos usar la función `LeftModuleGeneralMappingByImages`, pues en GAP, los espacios vectoriales son considerados como módulos a izquierda.

Un complemento de un subespacio W de V lo podemos calcular usando el homomorfismo natural de V a V/W.

```
gap> v:=Rationals^4;;
gap> w:=Subspace(v,[[1,0,0,0],[0,1,0,0]]);;
gap> h:= NaturalHomomorphismBySubspace( v, w );
<linear mapping by matrix, ( Rationals^4 ) -> ( Rationals^2 )>
gap> PreImagesRepresentative(h,[1,0]);
[ 0, 0, 1, 0 ]
gap> PreImagesRepresentative(h,[0,1]);
[ 0, 0, 0, 1 ]
```

Al no ser la aplicación biyectiva no podemos usar `PreImage` como hacíamos con un isomorfismo.

```
gap> v1:=Rationals^4;;
gap> v2:=VectorSpace(Rationals,[[1,2],[3,4]]);;
gap> f:=LeftModuleGeneralMappingByImages(v1,v2,BasisVectors(Basis(v1)), [[1,2],[3,4],[1,0],[0,1]]);
[ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ] -> [ [ 1, 2 ], [ 3, 4 ], [ 1, 0 ], [ 0, 1 ] ]
```

```
gap> Image(f);
<vector space over Rationals, with 4 generators>
gap> Dimension(Image(f));
2
gap> Kernel(f);
<vector space over Rationals, with 2 generators>
gap> BasisVectors(Basis(last));
[ [ 1, -1/2, 1/2, 0 ], [ 0, 1, -3, -4 ] ]
gap> v1/Kernel(f)=Image(f);
true
```

También podemos usar notación matricial.

```
gap> LeftModuleHomomorphismByMatrix(Basis(v1),[[1,0],[0,1],[1,1],[-1,1]],
> Basis(v2));
<linear mapping by matrix, ( Rationals^
4 ) -> <vector space over Rationals, with 2 generators>>
gap> Image(f,[1,2,3,4]);
[ 10, 14 ]
gap> PreImagesRepresentative(f,[1,1]);
[ -1/2, 1/2, 0, 0 ]
gap> Kernel(f);
<vector space of dimension 2 over Rationals>
gap> BasisVectors(Basis(last));
[ [ 1, -1/2, 1/2, 0 ], [ 0, 1, -3, -4 ] ]
gap> IsSurjective(f);
true
gap> IsInjective(f);
false
```

5 Diagonalización

Veamos cómo calcular el polinomio característico y sus ceros.

```
gap> a:=[[1,0,1],[3,1,0],[0,1,0]];
```

```
[ [ 1, 0, 1 ], [ 3, 1, 0 ], [ 0, 1, 0 ] ]
gap> F:=GF(7);
GF(7)
gap> x:=X(F,"x");
x
gap> ax:=a*One(x)-x*a^0;
[ [ -x+Z(7)^0, 0*Z(7), Z(7)^0 ], [ Z(7), -x+Z(7)^0, 0*Z(7) ],
  [ 0*Z(7), Z(7)^0, -x ] ]
gap> Determinant(ax);
-x^3+Z(7)^2*x^2-x+Z(7)
gap> RootsOfUPol(last);
[ Z(7)^3, Z(7)^5, Z(7)^5 ]
```

Y ahora los subespacios propios los podemos calcular con `NullspaceMat`.

```
gap> RootsOfUPol(last);
[ Z(7)^3, Z(7)^5, Z(7)^5 ]
gap> Set(last,r->NullspaceMat(TransposedMat(a*One(x)-r*a^0)));
[ [ [ Z(7), -Z(7)^0, Z(7)^0 ] ], [ [ Z(7)^2, Z(7)^5, Z(7)^0 ] ] ]
gap> Int(Z(7));
3
```

Como vemos, el elemento `Z(7)^5` es una raíz doble cuyo subespacio propio tiene dimensión uno, por lo que la matriz no es diagonalizable.

Veamos un ejemplo que sí es diagonalizable.

```
gap> a:=[ [ -1, 0, -1 ], [ 0, -2, 0 ], [ -3, 0, 1 ] ];;
gap> x:=X(Rationals,"x");;
gap> ax:=a-x*a^0;
[ [ -x-1, 0, -1 ], [ 0, -x-2, 0 ], [ -3, 0, -x+1 ] ]
gap> Determinant(ax);
-x^3-2*x^2+4*x+8
gap> RootsOfUPol(last);
[ 2, -2, -2 ]
gap> Set(last,r->NullspaceMat(TransposedMat(a*One(x)-r*a^0)));
[ [ [ 0, 1, 0 ], [ 1, 0, 1 ] ], [ [ -1/3, 0, 1 ] ] ]
```

```
gap> s:=[ [ 0, 1, -1/3 ], [ 1, 0, 0 ], [ 0, 1, 1 ] ];
gap> s^-1*a*s;
[ [ -2, 0, 0 ], [ 0, -2, 0 ], [ 0, 0, 2 ] ]
```