The MeatAxe in GAP

A. Hulpke, MATH 666

Irreducibility with Bare Hands\(^1\)

Consider a 4-dimensional module over the field with two elements. (The group happens to be \(S_5\).)

```
gap> Display(a1);
   . 1 . 
   . . 1
   . . . 1
   1 1 1 1

gap> Display(a2);
   1 . . .
   1 1 . .
   . . 1 .
   . . . 1
```

We form a “random” element in the group algebra and verify that it has nullspace dimension (“nullity”)\(^1\):

```
gap> omega:=a1*a2+a1^0;;Display(omega);
   . 1 . 
   . 1 1
   . . 1 1
   . 1 1 .

gap> RankMat(omega);
3
```

Now we take a vector \(y\) in the (row) nullspace and check (with some ad-hoc images) that \(y \cdot A\) is the whole module:

```
gap> nr:=NullspaceMat(omega);;Display(nr);
   . 1 . 
   1 1 1

gap> y:=nr[1];;

gap> Display([y,y*a1,y*a1^2,y*a1^3]);
   . 1 .
   1 1 .
   1 . .
```

\(^1\)For mathematicians the phrase ‘with bare hands’ carries both good and, bad overtones. Compare “The author does not use Lagrange multipliers but carries out the calculations bare handed” with “He did not know how to operate a crane so he built the dam with his bare hands”.

**Tom Körner:** *The Pleasures of Counting*, Cambridge University Press
A larger example

We want to investigate the structure of the regular module of $S_4$ over the field with 5 elements. We start by defining $S_4$.

```gap
gap> g:=Group((1,2,3,4),(1,2));;
```

The next step is to create the regular module. (Since RegularModule returns also corresponding group generators, we need only part of the result.)

```gap
gap> m:=RegularModule(g,GF(5));;
```

The component $m$.generators is a list of matrices, corresponding to the group generators. We can look at these matrices, however GAP by default stores them in compressed format and one would have to ask explicitly to see the entries. (The assumption is that you typically would not want to see $1000 \times 1000$ matrices.)

```gap
gap> m.generators[1];
< immutable compressed matrix 24x24 over GF(5) >
gap> Display(m.generators[1]);
...
...
...
1
...
...
...
```

We now apply NORTON's irreducibility test as built in. It finds a submodule of dimension 8.
We consider (ad hoc, for a full analysis we would have to consider both submodule and factor module) the (induced) factor module, which has dimension $24 - 8 = 16$. (The MeatAxe commands all start with MTX. for a technical reason that does not need to bother us here.)

We again apply the irreducibility test and find a submodule of dimension 3. Iterative application to this submodule shows that it is irreducible.

This 5-dimensional module is not irreducible, but has a submodule of dimension 2, the factor of dimension $5 - 2 = 3$ however is irreducible.
We now have two 3-dimensional modules. We apply the isomorphism test of problem 14 and find that the modules are not isomorphic (and not dual to each other).

Next, we go back to the 13-dimensional module we had before and consider the quotient of its 5-dimensional submodule.

It has an 6 dimensional factor module, on which we concentrate.

This 6-dimensional factor in turn has a 3-dimensional submodule. We find that both this submodule and its factor module are irreducible.
generators := [ [ [ 0*Z(5), Z(5)^2, 0*Z(5) ], [ Z(5)^0, 0*Z(5), 0*Z(5) ] ], [...]
gap> MTX.ProperSubmoduleBasis(m10);
fail

Again we test for isomorphism. We find that these two modules are isomorphic to the two modules we found earlier.

gap> MTX.Isomorphism(m3,m9);
fail
gap> MTX.Isomorphism(m6,m9);
[ [ 0*Z(5), 0*Z(5), Z(5)^0 ], [ Z(5)^0, Z(5)^2, Z(5)^2 ], [ Z(5)^2, Z(5)^3, Z(5)^3 ] ]
gap> MTX.Isomorphism(m3,m10);
[ [ Z(5)^3, Z(5), Z(5)^2 ], [ Z(5)^2, Z(5)^2, 0*Z(5) ], [ Z(5), Z(5)^0, Z(5)^0 ] ]

To find all irreducible modules we would have to continue on the submodules and factor modules we found but discarded. This process can obviously be automatized, which the MTX.CollectedFactors command does. It returns list whose entries are lists of the form [module, count] that return the irreducible constituents with multiplicities.

gap> MTX.CollectedFactors(m);
[ [ rec( field := GF(5), isMTXModule := true, dimension := 1,
  generators := [ [ [ Z(5)^0 ] ], [ [ Z(5)^0 ] ] ] ),
  1],
  [...]
gap> List(last,i->[i[1].dimension,i[2]]);
[ [ 1, 1 ], [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 3, 3 ] ]

By keeping track of the actual submodule bases we can also obtain bases for the actual submodules. As the regular module has too many such submodules, we thus consider a smaller example a particular permutation group of degree 8 and its permutation module. We obtain this module by forming permutation matrices for the group generators, and then form a module for the action given by these matrices:

gap> g:=TransitiveGroup(8,42);; Size(g);
288
gap> mats:=List(GeneratorsOfGroup(g),i->PermutationMat(i,8,GF(2)));
[ [ <a GF2 vector of length 8>, <a GF2 vector of length 8>, [...]
gap> m:=GModuleByMats(mats,GF(2));
rec( field := GF(2), isMTXModule := true, dimension := 8,
  generators := [ [ an immutable 8x8 matrix over GF2>,
    <an immutable 8x8 matrix over GF2>, <an immutable 8x8 matrix over GF2> ] )

We find that this module has proper submodules of dimension 1, 2, 6, 7 and 8.
gap> s:=MTX.BasesSubmodules(m);
[ [ ], <an immutable 1x8 matrix over GF2>, <an immutable 2x8 matrix over GF2>,
  <an immutable 6x8 matrix over GF2>, <an immutable 7x8 matrix over GF2>,
  <an immutable 8x8 matrix over GF2> ]

By considering the matrix ranks of lists of concatenated basis vectors we also find that this module
is uniserial, i.e. there is only one composition series and the modules are contained in each other
simply according to their dimensions. (In particular this module is not fully reducible!)

gap> RankMat(Concatenation(s[1],s[2]));
1
gap> RankMat(Concatenation(s[2],s[3]));
2
gap> RankMat(Concatenation(s[3],s[4]));
6
gap> RankMat(Concatenation(s[4],s[5]));
7