GAP Package
COHOMOLO

by

Derek Holt

Mathematixs Institute
University of Warwick, Coventry, CV4 7AL

Contents

1 Cohomology

This chapter describes functions which may be used to perform certain cohomological calculations on a finite group G. There is a file *gap-dir*/`pkg/cohomolo/gap/cohomolo.tst` which contains simple commands that can be used to test the installation of the package. If you start GAP in the directory *gap-dir*/`pkg/cohomolo/gap`, then you can read the file `cohomolo.tst` into GAP to peform the test.

This Package has been updated from the original GAP3 package with minimal changes, so the user should find the interface unchanged. In fact the only real changes are that the function `InfoCohomology` has been replaced by the `Info` variable `InfoCohomolo`, and the function `SplitExtension` has been renamed `SplitExtensionCHR`, to avoid clashing with an existing GAP function name. (Of course, it does more or less the same thing as the GAP function!)

The following properties of G can be computed:

`(i)` The p-part Mul_p of the Schur multiplier Mul of G, and a presentation of a covering extension of Mul_p by G, for a specified prime p;

`(ii)` The dimensions of the first and second cohomology groups of G acting on a finite dimensional KG module M, where K is a field of prime order; and

`(iii)` Presentations of split and nonsplit extensions of M by G.

All of these functions require G to be defined as a finite permutation group. The functions which compute presentations require, in addition, a presentation of G. Finally, the functions which operate on a module M require the module to be defined by a list of matrices over K. This situation is handled by first defining a GAP record, which contains the required information. This is done using the function `CHR`, which must be called before any of the other functions. The remaining functions operate on this record.

If no presentation of the permutation group G is known, and G has order at most 32767, then a presentation can be computed using the package function `CalcPres` (which calls a standalone C program), or alternatively by the GAP function `Image(IsomorphismFpGroup(`G`))`. On the other hand, if you start with a finitely presented group, then you can create a permutation representation with the function `PermRep` (although there is no guarantee that the representation will be faithful in general).

The functions all compute and make use of a descending sequence of subgroups of G, starting at G and ending with a Sylow p-subgroup of G, and it is usually most efficient to have the indices of the subgroups in this chain as small as possible. If you get a warning message, and one of the function fails because the indices in the chain computed are too large, then you can try to remedy matters by supplying your own chain. See Section 1.10 for more details, and an example.

If you set the Info variable `InfoCohomolo` to 1, then a small amount of information will be printed, indicating what is happening. If *chr* is the cohomology record you are working with, and you set the field *chr*`.verbose` to the value `true`, then you will see all the output of the external programs.

1.1 CHR

1 ▶ `CHR(` *G*`,` *p*`, [`*F*`,` *mats*`] )` F

`CHR` constructs a cohomology-record, which is used as a parameter for all of the other functions in this chapter. G must be a finite permutation group, and p a prime number. If present, F must either be zero or a finitely presented group with the same number of generators as G, of which the relators are satisfied by the generators of G. In fact, to obtain meaningful results, F should almost certainly be isomorphic to G. If present, *mats* should be a list of invertible matrices over the finite field $K = GF(p)$. The list should have the same length as the number of generators of G, and the matrices should correspond to these generators, and define a $GF(p)G$-module, which we shall denote by M.

1.2 SchurMultiplier

1 ▶ `SchurMultiplier(` *chr* `)` F

chr must be a cohomology-record that was created by a call of `CHR(`*G*`,`*p*`,[`*F*`,`*mats*`])`. `SchurMultiplier` calculates the p-part Mul_p of the Schur multiplier *Mul* of G. The result is returned as a list of integers, which are the abelian invariants of Mul_p. If the list is empty, then Mul_p is trivial.

1.3 CoveringGroup

1 ▶ `CoveringGroup(` *chr* `)` F

chr must be a cohomology-record, created by a call of `CHR(`*G*`,`*p*`,`*F*`[,`*mats*`])`, where F is a finitely presented group. `CoveringGroup` calculates a presentation of a covering extension of Mul_p by G, where Mul_p is the p-part of the Schur multiplier *Mul* of G. The set of generators of the finitely presented group that is returned is a union of two sets, which are in one-one correspondence with the generators of F and of Mul_p, respectively.

The relators fall into three classes:

(a) Those that specify the orders of the generators of Mul_p;

(b) Those that say that the generators of Mul_p are central; and

(c) Those that give the values of the relators of F as elements of Mul_p.

1.4 FirstCohomologyDimension

1 ▶ `FirstCohomologyDimension(` *chr* `)` F

chr must be a cohomology-record, created by a call of `CHR(`*G*`,`*p*`,`*F*`,`*mats*`)`. (If there is no finitely presented group F involved, then the third parameter of `CHR` should be given as 0.) `FirstCohomologyDimension` calculates and returns the dimension over $K = GF(p)$ of the first cohomology group $H^1(G, M)$ of the group G in its action on the module M defined by the matrices *mats*.

1.5 SecondCohomologyDimension

1 ▶ `SecondCohomologyDimension(` *chr* `)` F

chr must be a cohomology-record, created by a call of `CHR(`*G*`,`*p*`,`*F*`,`*mats*`)`. (If there is no finitely presented group F involved, then the third parameter of `CHR` should be given as 0.) `SecondCohomologyDimension` calculates and returns the dimension over $K = GF(p)$ of the second cohomology group $H^2(G, M)$ of the group G in its action on the module M defined by the matrices *mats*.

1.6 SplitExtensionCHR

1 ▶ `SplitExtensionCHR(` *chr* `)` F

chr must be a cohomology-record, created by a call of `CHR(`G`,`p`,`F`,`*mats*`)`, where F is a finitely presented group. `SplitExtensionCHR` returns a presentation of the split extension of the module M defined by the matrices *mats* by the group G. This is a straightforward calculation, and involves no call of the external cohomology programs. It is provided here for convenience.

1.7 NonsplitExtension

1 ▶ `NonsplitExtension(` *chr*`[,` *vec*`] )` F

chr must be a cohomology-record, created by a call of `CHR(`G`,`p`,`F`,`*mats*`)`, where F is a finitely presented group. If present, *vec* must be a list of integers of length equal to the dimension over $K = GF(p)$ of the second cohomology group $H^2(G, M)$ of the group G in its action on the module M defined by the matrices *mats*. `NonsplitExtension` calculates and returns a presentation of a nonsplit extension of M by G. Since there may be many such extensions, and the equivalence classes of these extensions are in one-one correspondence with the nonzero elements of $H^2(G, M)$, the optional second parameter can be used to specify an element of $H^2(G, M)$ as a vector. The default value of this vector is `[1,0,...,0]`. The set of generators of the finitely presented group that is returned is a union of two sets, which are in one-one correspondence with the generators of F and of M (as an abelian group), respectively.

The relators fall into three classes:

(a) Those that say that M is an abelian group of exponent p;

(b) Those that define the action of the generators of F on those of M; and

(c) Those that give the values of the relators of F as elements of M.

(*Note*: It is not particularly efficient to call `SecondCohomologyDimension` first to calculate the dimension of $H^2(G, M)$, which must of course be known if the second parameter is to be given; it is preferable to call `NonsplitExtension` immediately without the second parameter (which will return one nonsplit extension), and then to call 'SecondCohomologyDimension', which will at that stage return the required dimension immediately - all subsequent calls of `NonsplitExtension` on *chr* will also yield immediate results.)

1.8 CalcPres

1 ▶ `CalcPres(` *chr* `)` F

`CalcPres` computes a presentation of the permutation group *chr*`.permgp` on the same set of generators as *chr*`.permgp`, and stores it as *chr*`.fpgp`. It currently only works for groups of order up to 32767, although that could easily be increased if required. Note that a presentation of a finite group G can also be computed by the standard GAP function call `Image(IsomorphismFpGroup(`G`))`.

1.9 PermRep

1 ▶ `PermRep(` G`,` K `)` F

`PermRep` calculates the permutation representation of the finitely presented group F on the right cosets of the subgroup K, and returns it as a permutation group of which the generators correspond to those of F. It simply calls the GAP Todd-Coxeter function. Of course, there is no guarantee in general that this representation will be faithful.

1.10 Further Information

Suppose, as usual, that the cohomology record *chr* was constructed with the call CHR(*G*,*p*,[*F*],[*mats*]). All of the functions make use of a strictly decreasing chain of subgroups of the permutation group G starting with G itself and ending with a Sylow p-subgroup P of G. In general, the programs run most efficiently if the indices between successive terms in this sequence are as small as possible. By default, GAP will attempt to find a suitable chain, when you call the first cohomology function on *chr*. However, you may be able to construct a better chain yourself. If so, then you can do this by assigning the record field *chr*.chain to the list L of subgroups that you wish to use. You should do that before calling any of the cohomology functions. Remeber that the first term in the list must be G itself, the sequence of subgroups must be strictly decreasing, and the last term must be equal to the Sylow subgroup stored as *chr*.sylow. (You can change *chr*.sylow to a different Sylow p-subgroup if you like.) Here is a slightly contrived example of this process.

```
gap> G:=AlternatingGroup(16);;
gap> chr:=CHR(G,2);;
gap> SetInfoLevel(InfoCohomolo,1);;
gap> SchurMultiplier(chr);
#Indices in the subgroup chain are:  2027025 315
#WARNING: An index in the subgroup chain found is larger than 50000.
#This calculation may fail. See manual for possible remedies.
#I   Cohomology package: Calling external program.
Out of tree space. Increase TRSP.
#I   External program complete.
Error 'Cohomology' failed for some reason.
 at
Error( "'Cohomology' failed for some reason.\n" );
Cohomology( chr, true, false, false, TmpName(  ) ); called from
<function>( <arguments> ) called from read-eval-loop
Entering break read-eval-print loop, you can 'quit;' to quit to outer loop,
or you can return to continue
brk> quit;

#The first index in the chain found by GAP was hopelessly large.
#Let's try and do better.

gap> P:=chr.sylow;;
gap> H1:=Subgroup(G, [(1,2)(9,10), (2,3,4,5,6,7,8),
>                     (1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)(8,16)]);;
gap> Index(G,H1);
6435
gap> H2:=Subgroup(H1, [(1,2)(5,6),(1,2)(9,10), (2,3,4),
>    (1,5)(2,6)(3,7)(4,8),  (1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)(8,16)]);;
gap> Index(H1,H2);
1225
gap> IsSubgroup(H2,P);
true
#If that had been false, we could have replaced chr.sylow by
#a Sylow 2-subgroup of H2.
gap> Index(H2,P);
81
gap> chr.chain := [G,H1,H2,P];;
gap> SchurMultiplier(chr);
```

```
#Calling external program.
#External program complete.
#Removing temporary files.
[ 2 ]
gap> quit;
```