CRIME

A GAP Package to Calculate Group Cohomology and Massey Products

1.6

17 March 2022

Marcus Bishop

Marcus Bishop
Email: marcus.bishop@gmail.com
Homepage: http://math.uic.edu/~marcus

Copyright

Acknowledgements

This project would not have been possible without Jon Carlson. Jon devised the algorithms used by `ProjectiveResolution`, `CohomologyGenerators`, and `CohomologyRelators`, having already implemented them in Magma, and shared these programs with me.

Contents

Chapter 1

Introduction

1.1 Installation and Loading

Like other GAP packages, you download and unpack this package into GAP's pkg directory. For example, if you were running GAP on some machine with a Unix-based operating system and GAP were installed in the directory /usr/local/gap then you would do the following to install CRIME.

Example

```
$ cd /usr/local/gap/pkg
$ su
% wget https://github.com/gap-packages/crime/releases/download/v1.5/crime-1.6.tar.gz
% tar xvzf crime-1.6.tar.gz
```

In this situation, users would load the package with the LoadPackage command.

Example

```
$ gap
gap> LoadPackage("crime");
```

Users not having root access, using someone else's computer, or having bad relationships with their network administrators, could install the package into their home directories or into some other writable directory such as /tmp and load the package as follows.

Example

```
$ mkdir /tmp/pkg
$ cd /tmp/pkg
% wget https://github.com/gap-packages/crime/releases/download/v1.5/crime-1.6.tar.gz
$ tar xvzf crime-1.6.tar.gz
$ gap -l ';/tmp'
gap> LoadPackage("crime");
```

Even simpler, users can simply install the package in the ~/.gap directory as follows.

Example

```
$ mkdir -p ~/.gap/pkg
$ cd ~/.gap/pkg
% wget https://github.com/gap-packages/crime/releases/download/v1.5/crime-1.6.tar.gz
% tar xvzf crime-1.6.tar.gz
$ gap
gap> LoadPackage("crime");
```

Finally, it would be a good idea to run the test file to confirm that all the functions work properly. This can be accomplished as follows.

Example

```
gap> ReadPackage("crime", "tst/testall.g");
```

You can count yourself lucky if GAP doesn't complain about anything. There is also a longer running test file for those having ample free time described in Appendix B.

1.2 For Further Information

The file `doc/example.*` contains the step-by-step CRIME calculation of the cohomology ring of the quaternion group. The file `doc/explanation.*` contains a theoretical description of how the package calculates the various cohomology products. Users wishing to read the source code can print all the programs in the `gap` directory with the program `gap/print.pl` which should be executed in the `gap` directory.

Chapter 2

Usage

Unless otherwise specified, all the functions described below taking an argument `n` do whatever the manual says they do up to homological degree `n`. These functions are idempotent in the sense that called a second time with the same argument `n`, they do nothing, but called with a bigger `n`, they continue computing from where the previous calculations finished.

2.1 Cohomology Objects

The computation of group cohomology involves several calculations, the results of which are reused in later calculations, and are thus collected in an object of type `CObject`, which is created with the following command.

2.1.1 CohomologyObject

▷ `CohomologyObject(G, M)` (operation)

▷ `CohomologyObject(G)` (operation)

Returns: a `CObject`.

This function creates a `CObject` having components the p-group `G` and the MeatAxe module `M`, which should be a kG-module where G the group `G` and k a field of characteristic p. Note that MeatAxe modules know what k is, but not what G is, which is why this operation requires the user to specify G but not k.

Fortunately, most users don't need to know anything about MeatAxe modules, being interested primarily in the case where $k = \mathbb{F}_p$, and $M = k$ is the trivial kG-module. In this situation, the second invocation creates a cohomology object having components the p-group G and the trivial MeatAxe kG-module $k = \mathbb{F}_p$.

We emphasize that in the first invocation, k can be any field of characteristic p and `M` can be any MeatAxe module over kG, and that `ProjectiveResolution` works when `M` is an arbitrary MeatAxe module, but that all the functions dealing with the ring-structure of $H^*(G,k)$ require that `M` be the trivial module.

The cohomology object is used to store, in addition to the items mentioned above, the boundary maps, the Betti numbers, the multiplication table, etc.

2.2 Minimal Projective Resolutions

Given a p-group G, a field k of characteristic p, and a kG-module M, the function below computes the beginning of the minimal projective resolution of M

$$P_n \to \cdots \to P_2 \to P_1 \to P_0 \to M \to 0$$

where $P_i = (kG)^{\oplus b_i}$ for certain numbers b_i, the *Betti numbers* of the resolution. The minimal kG-projective resolution of M is unique up to chain isomorphism. Because of the minimality of P_* the groups $\mathrm{Ext}^i_{kG}(M,N)$ are simply $\mathrm{Hom}_{kG}(P_i,N)$, and if M and N are both the trivial kG-module k, then $H^i(G,k) = \mathrm{Ext}^i_{kG}(k,k) = k^{b_i}$.

2.2.1 ProjectiveResolution

▷ `ProjectiveResolution(C, n)` (operation)

Returns: a list containing the Betti numbers $b_0, b_1, \ldots, b_n$.

Given a cohomology object `C` having components G and M, this function computes the first `n`+1 terms of the minimal projective resolution P_* of M of the form $P_i = (kG)^{\oplus b_i}$ for $0 \le i \le n$ and returns the numbers b_i as a list.

2.2.2 BoundaryMap

▷ `BoundaryMap(C, n)` (operation)

Returns: the `n`th boundary map.

Given the cohomology object `C`, this function computes a projective resolution to degree `n` if it hasn't been computed already, and returns the `n`th boundary map $P_n \to P_{n-1}$.

The map returned is a $b_n \times b_{n-1}\,|G|$ matrix, having in the ith row the image of the element 1_G from the ith direct summand of P_n.

See the file `doc/example.*` for an example of the usage and interpretation of the result of this function.

2.3 Cohomology Generators and Relators

See [2] for the details of the calculation of cohomology products using composition of chain maps. See also the file `doc/explanation.*` for an explanation of the implementation.

2.3.1 CohomologyGenerators

▷ `CohomologyGenerators(C, n)` (operation)

Returns: a list containing the degrees of the elements of a set of generators of the cohomology ring.

Given a cohomology object `C` having group component G and module component the trivial kG-module, this function computes a set of generators of $H^*(G,k)$ having degree `n` or less, and stores them in `C`. The function returns a list of the degrees of these generators.

The actual cohomology generators are represented by maps $P_i \to k$ for $0 \le i \le n$ and are stored in `C` as matrices. Only their degrees are returned.

2.3.2 CohomologyRelators

▷ `CohomologyRelators(C, n)` (operation)

Returns: a list of generators and a list of relators.

Given a cohomology object `C` having group component G and module component k, this function computes a set of generators of the ideal of relators in $H^*(G,k)$, all having degree n or less.

More specifically, the function returns two lists, the first list containing the variables `z`, `y`, `x`, ... corresponding to the generators of $H^*(G,k)$ if there are fewer than 12 generators and containing the variables `x_1`, `x_2`, `x_3`, ... otherwise. The second list is a list of polynomials in the variables from the first list.

These two lists should be interpreted as follows. A degree `n` approximation of the cohomology ring $H^*(G,k)$ is given by the polynomial ring over k in the non-commuting variables from the first list, (having degrees given by the list returned by `CohomologyGenerators` in section 2.3.1) and subject to the relators in the second list. See section 2.6 for more details still.

For example, consider the following commands.

Example

```
gap> C:=CohomologyObject(DihedralGroup(8));
<object>
gap> CohomologyGenerators(C,10);
[ 1, 1, 2 ]
gap> CohomologyRelators(C,10);
[ [ z, y, x ], [ z*y+y^2 ] ]
```

This tells us that for $G = D_8$ and $k = \mathbb{F}_p$, the cohomology ring $H^*(G,k)$ is the graded-commutative polynomial ring in the variables z, y, x of degrees 1, 1, 2, subject to the relation $zy + y^2$. But since $H^*(G,k)$ is commutative, k being of characteristic 2, we have $H^*(G,k) = k[z,y,x]/(zy+y^2)$. This result can be further improved by taking $z = z+y$, giving $H^*(G,k) = k[z,y,x]/(zy)$.

Observe that in this case, we knew in advance that there was a set of generators for $H^*(G,k)$ all having degree less than 10, and that there was a set of generators of the ideal of relators all having degree less than 10. See see section 2.6 for details.

While this isn't likely to occur, we point out that if there are 12 or more generators and some of the indeterminates `x_1`, `x_2`, `x_3`, ... have already been named, say by a previous call to `CohomologyRelators`, then these variables will retain their old names. If this is confusing, you could restart GAP and do it again.

Finally, `CohomologyRelators` is *not* idempotent for efficiency reasons, so sadly, if you don't uncover all the relators the first time, you will have to start all over from the beginning.

2.4 Tests for Completion

A test or series of tests for completion of the calculation will hopefully be implemented soon. See [2] for the details.

2.5 Cohomology Rings

Whereas the operations in sections 2.3.1 and 2.3.2 calculate a presentation for the cohomology ring, the operation below creates the ring in GAP as a structure constant algebra.

See [2] for the details of the calculation of cohomology products using composition of chain maps. See also the file `doc/explanation.*` for an explanation of the implementation.

2.5.1 CohomologyRing

▷ `CohomologyRing(C, n)` (operation)
▷ `CohomologyRing(G, n)` (operation)

Returns: the cohomology ring of G.

Given a cohomology object `C` with group component G and module component the trivial kG-module, this function returns the degree `n` truncation of the cohomology ring $H^*(G,k)$. See 2.6 for what this means exactly. The object returned is a structure constant algebra.

Users interested only in working with the cohomology ring of a group as a GAP object, and not in calculating generators, relators, induced maps, etc, can use the second invocation of this function, which returns the cohomology ring of the group `G` immediately, throwing away all intermediate calculations.

Observe that the object returned is a degree `n` truncation of the infinite-dimensional cohomology ring. A consequence of this is that multiplying two elements whose product has degree greater than `n` results in zero, whether or not the product is really zero.

Observe also that calling `CohomologyRing` a second time with a bigger `n` does *not* extend the previous ring, but rather, recalculates the entire ring from the beginning. Extending the previous ring appears not to be worth the effort for technical reasons, since almost everything would need to be recalculated again anyway.

Recall that $H^*(G,k)$ is a graded algebra, the components being the cohomology groups $H^i(G,k)$. The following functions were intended to be used for cohomology rings, but in principle, they work for any graded structure constant algebra.

2.5.2 IsHomogeneous

▷ `IsHomogeneous(e)` (operation)

Returns: `true` or `false`.

Given an element `e` of a cohomology ring $H^*(G,k)$, this operation determines whether or not `e` is homogeneous, that is, whether `e` is contained in $H^i(G,k)$ for some `i`.

2.5.3 Degree

▷ `Degree(e)` (method)

Returns: the degree of `e`.

This function returns the degree of the possibly non-homogeneous element `e` of a cohomology ring $H^*(G,k)$. Specifically, if $H^*(G,k) = A_0 \oplus A_1 \oplus A_2 \oplus \cdots$ where $A_i = H^i(G,k)$ then this function returns the minimum n such that `e` is in $A_0 \oplus A_1 \oplus \cdots \oplus A_n$.

Example

```
gap> A:=CohomologyRing(DihedralGroup(8),10);
<algebra of dimension 66 over GF(2)>
gap> b:=Basis(A);
CanonicalBasis( <algebra of dimension 66 over GF(2)> )
gap> x:=b[2]+b[4];
v.2+v.4
gap> IsHomogeneous(x);
```

```
false
gap> Degree(x);
2
```

2.5.4 LocateGeneratorsInCohomologyRing

▷ `LocateGeneratorsInCohomologyRing(C)` (function)

Returns: a list containing the cohomology generators.

Having already called `CohomologyRing` (see 2.5.1), this function returns a list of elements of the cohomology ring which together with the identity element generate the cohomology ring.

This function is a wrapper for `CohomologyGenerators` (see 2.3.1). It points out which elements of the cohomology ring correspond with the generators found by `CohomologyGenerators`.

Example

```
gap> C:=CohomologyObject(SmallGroup(8,4));
<object>
gap> A:=CohomologyRing(C,10);
<algebra of dimension 17 over GF(2)>
gap> L:=LocateGeneratorsInCohomologyRing(C);
[ v.2, v.3, v.7 ]
gap> A=Subalgebra(A,Concatenation(L,[One(A)]));
true
```

2.6 What Happens if `n` Isn't Big Enough?

Since P_* is a *minimal* projective resolution, we have $H^i(G,k) = \mathrm{Hom}_{kG}(P_i,k)$ where $P_i = (kG)^{b_i}$ so that $H^i(G,k)$ has a natural basis consisting of the maps sending the element 1_G of the jth direct summand of P_i to 1_k and all other direct summands to 0, for $1 \le j \le b_i$, where b_i is the kG-rank of P_i.

The command `CohomologyRing(C,n)` forms the vector space whose basis is the concatenation of the natural bases of $H^i(G,k)$ for $1 \le i \le n$ and computes all products of basis elements x and y for which $\deg x + \deg y \le n$. Thinking of $H^*(G,k)$ in terms of it's multiplication table, this means that the function computes the upper left-hand corner of the multiplication table. If $\deg x + \deg y > n$, the product xy is taken to be zero. Therefore, the ring returned by `CohomologyRing` is $H^*(G,k)/J_{>n}$ where $J_{>n}$ is the ideal of all elements of degree $> n$.

The ring determined by `CohomologyGenerators` and `CohomologyRelators` is somewhat different. `CohomologyGenerators` proceeds inductively, taking all natural basis elements of $H^1(G,k)$ as generators, and for $1 < i \le n$, taking all natural basis elements of $H^i(G,k)$ which are *not* products of lower-degree elements as generators. Therefore, unless you know that there is an n for which there exists a generating set of $H^*(G,k)$ consisting of elements of degree n or less, then you are *not* guaranteed that the elements returned by the `CohomologyGenerators` generate $H^*(G,k)$ as a ring. The knowledge of such an n is the subject of section 2.4.

Similarly, `CohomologyRelators` proceeds inductively until degree n, returning a list of polynomials which generate the ideal of relators of degree n or less. Again, you have to already know how big n should be.

The result of the preceding information is that there is a homomorphism $f : k\langle x_1,x_2,\ldots,x_m\rangle / I \to H^*(G,k)$, where $k\langle x_1,x_2,\ldots,x_m\rangle$ is the graded polynomial ring over k in the non-commuting variables $x_1,x_2,\ldots,x_m$, having degrees the numbers in the list returned by `CohomologyGenerators`, and I is the ideal in $k\langle x_1,x_2,\ldots,x_m\rangle$ generated by the elements returned by `CohomologyRelators(C,n)`.

Therefore, if there is a generator of degree greater than n, then f won't be surjective. Similarly, if there is a relator of degree greater than n which is not a consequence of lower degree relators, then f won't be injective. See section 2.4 for a discussion on how big n needs to be to ensure that f will be an isomorphism.

2.7 Induced Maps

Let $f : H \to G$ be a group homomorphism. Then f induces a homomorphism on cohomology $H^*(G,k) \to H^*(H,k)$ which is returned by the following function.

2.7.1 InducedHomomorphismOnCohomology

▷ `InducedHomomorphismOnCohomology(C, D, f, n)` (function)

Returns: the induced homomorphism on cohomology.

This function returns the induced homomorphism $H^*(G,k) \to H^*(H,k)$ where the groups H and G are the components of the cohomology objects `C` and `D` and $f : H \to G$ is a group homomorphism. If the cohomology rings have not yet been calculated, they will be computed to degree n, and in this case, they can then be accessed by calling `CohomologyRing` (see 2.5.1).

2.7.2 SubgroupInclusion

▷ `SubgroupInclusion(H, G)` (function)

Returns: the inclusion $H \to G$

This function returns the group homomorphism $H \to G$ when H is a subgroup of G. The returned map can be used as the `f` argument of `InducedHomomorphismOnCohomology`, in which case the induced homomorphism is the restriction map $\mathrm{Res}^G_H : H^*(G,k) \to H^*(H,k)$.

The following example calculates the homomorphism on cohomology induced by the inclusion of the cyclic group of size 4 into the dihedral group of size 8.

Example

```
gap> G:=DihedralGroup(8);H:=Subgroup(G,[G.2]);
<pc group of size 8 with 3 generators>
Group([ f2 ])
gap> C:=CohomologyObject(H);D:=CohomologyObject(G);
<object>
<object>
gap> i:=SubgroupInclusion(H,G);
[ f2 ] -> [ f2 ]
gap> Res:=InducedHomomorphismOnCohomology(C,D,i,10);;
gap> A:=CohomologyRing(D,10);
<algebra of dimension 66 over GF(2)>
gap> LocateGeneratorsInCohomologyRing(D);
[ v.2, v.3, v.6 ]
gap> A.1^Res; A.2^Res; A.3^Res; A.6^Res;
v.1
0*v.1
v.2
v.3
```

2.8 Massey Products

See [3] for the definitions and [1] for the details of the calculation using the Yoneda cocomplex. See also the file `doc/explanation.*` for an explanation of the implementation.

2.8.1 MasseyProduct

▷ `MasseyProduct(x1, x2, ..., xn)` (function)

Returns: the Massey product $\langle x_1, x_2, \ldots, x_n \rangle$.

Given elements $x_1, x_2, \ldots, x_n$ of the ring returned by `CohomologyRing` (see 2.5) this function computes the n-fold Massey product $\langle x_1, x_2, \ldots, x_n \rangle$ provided that the lower-degree Massey products $\langle x_i, x_{i+1}, \ldots, x_j \rangle$ vanish for all $1 \leq i < j \leq n$ and returns `fail` otherwise.

As an example, recall that the cohomology rings of the cyclic groups C_3 and C_9 of sizes 3 and 9 over $k = \mathbb{F}_3$ are both given by $k\langle z, y \rangle / (z^2)$, so they are isomorphic as rings. However, the following example shows that $\langle z, z, z \rangle$ is non-zero in $H^*(C_3, k)$ but is zero in $H^*(C_9, k)$.

Example

```
gap> A:=CohomologyRing(CyclicGroup(3),10);
<algebra of dimension 11 over GF(3)>
gap> z:=Basis(A)[2];
v.2
gap> MasseyProduct(z,z);
0*v.1
gap> MasseyProduct(z,z,z);
v.3
gap> A:=CohomologyRing(CyclicGroup(9),10);
<algebra of dimension 11 over GF(3)>
gap> z:=Basis(A)[2];
v.2
gap> MasseyProduct(z,z);
0*v.1
gap> MasseyProduct(z,z,z);
0*v.1
gap> MasseyProduct(z,z,z,z,z,z,z,z,z);
v.3
```

Appendix A

Some Benchmarks

Some of the key subroutines have been improved in version 1.2 of CRIME, resulting in a significant speedup of many basic calculations.

The numbers in the following table compare the runtimes of the 1.1 and 1.2 versions of the functions listed in the column headers on all the groups of size 16. All functions were executed on a 2.4 GHz AMD64 processor with 12 GB of RAM. The number in the first column of every row is the Small Group Library number of the group used in that row, so the group used in row `n` is `SmallGroup(16,n)`. The upper number in each box is the runtime for the 1.1 version, whereas the lower number is the runtime for the 1.2 version.

Group	`ProjectiveResolution`	`CohomologyGenerators`	`CohomologyRelators`
1	0:00:00.424	0:00:00.040	0:00:00.092
	0:00:00.380	0:00:00.016	0:00:00.012
2	0:00:00.436	0:00:01.472	0:00:05.848
	0:00:00.072	0:00:00.116	0:00:00.396
3	0:00:02.752	0:00:11.469	0:00:42.887
	0:00:00.352	0:00:01.192	0:00:01.944
4	0:00:00.480	0:00:01.456	0:00:06.320
	0:00:00.068	0:00:00.116	0:00:00.316
5	0:00:00.480	0:00:01.136	0:00:06.045
	0:00:00.080	0:00:00.096	0:00:00.296
6	0:00:00.204	0:00:00.724	0:00:01.892
	0:00:00.036	0:00:00.072	0:00:00.116
7	0:00:00.504	0:00:01.156	0:00:06.140
	0:00:00.088	0:00:00.100	0:00:00.344
8	0:00:00.192	0:00:00.704	0:00:01.800
	0:00:00.036	0:00:00.064	0:00:00.120
9	0:00:00.060	0:00:00.100	0:00:00.284
	0:00:00.020	0:00:00.012	0:00:00.028
10	0:00:10.161	0:00:34.326	0:03:07.104
	0:00:01.524	0:00:04.252	0:00:08.089
11	0:00:10.397	0:00:32.354	0:03:07.355
	0:00:01.716	0:00:04.584	0:00:08.021
12	0:00:01.184	0:00:04.613	0:00:20.789
	0:00:00.192	0:00:00.544	0:00:00.984
13	0:00:01.152	0:00:04.496	0:00:20.990
	0:00:00.196	0:00:00.472	0:00:01.072
14	0:03:26.817	0:07:37.960	0:56:39.273
	0:00:41.919	0:02:07.399	0:01:54.752

Appendix B

Leisure and Recreation: Cohomology Rings of all Groups of Size 16

Below is the output of the test file `tst/batch.g`. The file runs through all groups of size n, which is initially set to 16, calls `ProjectiveResolution`, `CohomologyGenerators`, and `CohomologyRelators` for each group, and prints the results, as well as the runtimes for each operation, to a file like the one shown below. The runtimes in this example have been deleted, having been presented in Appendix A. The example below was computed on a 2.4 GHz AMD64 processor with 12 GB of RAM. See the file `tst/README` for suggestions on dealing with other users when running long-running batch processes.

Example

```
SmallGroup(16,1)
Betti Numbers: [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]
Generators in degrees: [ 1, 2 ]
Relators: [ [ z, y ], [ z^2 ] ]

SmallGroup(16,2)
Betti Numbers: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ]
Generators in degrees: [ 1, 1, 2, 2 ]
Relators: [ [ z, y, x, w ], [ z^2, y^2 ] ]

SmallGroup(16,3)
Betti Numbers: [ 1, 2, 4, 6, 9, 12, 16, 20, 25, 30, 36 ]
Generators in degrees: [ 1, 1, 2, 2, 2 ]
Relators: [ [ z, y, x, w, v ], [ z^2, z*y, z*x, y^2*v+x^2 ] ]

SmallGroup(16,4)
Betti Numbers: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ]
Generators in degrees: [ 1, 1, 2, 2 ]
Relators: [ [ z, y, x, w ], [ z^2, z*y+y^2, y^3 ] ]

SmallGroup(16,5)
Betti Numbers: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ]
Generators in degrees: [ 1, 1, 2 ]
Relators: [ [ z, y, x ], [ z^2 ] ]

SmallGroup(16,6)
Betti Numbers: [ 1, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6 ]
```

```
Generators in degrees: [ 1, 1, 3, 4 ]
Relators: [ [ z, y, x, w ], [ z^2, z*y^2, z*x, x^2 ] ]

SmallGroup(16,7)
Betti Numbers: [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ]
Generators in degrees: [ 1, 1, 2 ]
Relators: [ [ z, y, x ], [ z*y ] ]

SmallGroup(16,8)
Betti Numbers: [ 1, 2, 2, 2, 3, 4, 4, 4, 5, 6, 6 ]
Generators in degrees: [ 1, 1, 3, 4 ]
Relators: [ [ z, y, x, w ], [ z*y, z^3, z*x, y^2*w+x^2 ] ]

SmallGroup(16,9)
Betti Numbers: [ 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2 ]
Generators in degrees: [ 1, 1, 4 ]
Relators: [ [ z, y, x ], [ z*y, z^3+y^3, y^4 ] ]

SmallGroup(16,10)
Betti Numbers: [ 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66 ]
Generators in degrees: [ 1, 1, 1, 2 ]
Relators: [ [ z, y, x, w ], [ z^2 ] ]

SmallGroup(16,11)
Betti Numbers: [ 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66 ]
Generators in degrees: [ 1, 1, 1, 2 ]
Relators: [ [ z, y, x, w ], [ z*y ] ]

SmallGroup(16,12)
Betti Numbers: [ 1, 3, 5, 6, 7, 9, 11, 12, 13, 15, 17 ]
Generators in degrees: [ 1, 1, 1, 4 ]
Relators: [ [ z, y, x, w ], [ z^2+z*y+y^2, y^3 ] ]

SmallGroup(16,13)
Betti Numbers: [ 1, 3, 5, 6, 7, 9, 11, 12, 13, 15, 17 ]
Generators in degrees: [ 1, 1, 1, 4 ]
Relators: [ [ z, y, x, w ], [ z*y+x^2, z*x^2+y*x^2, y^2*x^2+x^4 ] ]

SmallGroup(16,14)
Betti Numbers: [ 1, 4, 10, 20, 35, 56, 84, 120, 165, 220, 286 ]
Generators in degrees: [ 1, 1, 1, 1 ]
Relators: [ [ z, y, x, w ], [  ] ]
```

References

[1] I. C. Borge. A cohomological approach to the classification of p-groups. http://eprints.maths.ox.ac.uk/40/, 2001. 12

[2] J. F. Carlson, L. Townsley, L. Valeri-Elizondo, and M. Zhang. *Cohomology rings of finite groups*, volume 3 of *Algebras and Applications*. Kluwer Academic Publishers, Dordrecht, 2003. 7, 8, 9

[3] D. Kraines. Massey higher products. *Trans. Amer. Math. Soc.*, 124:431--449, 1966. 12

Index