ModIsom
A GAP4 Package

Version 3.0.0

by

Bettina Eick
email: beick@tu-bs.de

February 2024

Contents

1 Introduction

This package contains various algorithms related to finite dimensional nilpotent associative algebras. It also contains many group-theoretical functions related to the Modular Isomorphism Problem. We first give a brief introduction to finite dimensional nilpotent algebras and then an overview of the main algorithms.

Associative algebras and nilpotency

Let A be an associative algebra of dimension d over a field F. Let $\{b_1, \ldots, b_d\}$ be a basis for A. We identify the element $x_1b_1 + \ldots + x_db_d$ of A with the element $(x_1, \ldots, x_d)$ of F^d. The multiplication of A can then be described by a **structure constants table**: a 3-dimensional array with entries $a_{i,j,k} \in F$ satisfying that

$$b_ib_j = \sum_{k=1}^{d} a_{i,j,k}b_k.$$

An associative algebra A is **nilpotent** if its **power series** terminates at the trivial ideal of A; that is

$$A > A^2 > \ldots > A^c > A^{c+1} = \{0\}$$

where A^j is the ideal of A generated by all products of length at least j. The length c of the power series is also called the **class** of A and the dimension of A/A^2 is the **rank** of A. Note that A is generated by $dim(A/A^2)$ elements. Clearly, A does not contain a multiplicative identity.

For computational purposes we describe a nilpotent associative algebra by a weighted basis and a description of the corresponding structure constants table. A basis of a nilpotent associative algebra A is **weighted** if there is a sequence of weights $(w_1, \ldots, w_d)$ so that

$$A^j = \langle b_i \mid w_i \geq j \rangle.$$

Note that $AA^j = A^{j+1}$ for every j. Thus it is possible to choose all basis elements of weight at least 2 so that $b_i = b_kb_l$ holds for some k and l, where b_k is of weight 1 and b_l is of weight $w_i - 1$. This feature allows an effective description of A via a **nilpotent structure constants table**. This contains the structure constants $a_{i,j,k}$ for all i with $w_i = 1$ and $1 \leq j,k \leq d$. For i with $w_i > 1$ it either contains a description as $b_i = b_kb_l$ or the structure constants $a_{i,j,k}$ for $1 \leq j,k \leq d$. It may also contain both or some partial overlap of these informations.

Isomorphisms and Automorphisms

Let A be a finite dimensional nilpotent associative algebra over a finite field. This package contains an implementation of the methods in [Eic08] which allow the determination of the automorphism group $Aut(A)$ and a **canonical form** $Can(A)$.

The automorphism group is given by generators and is represented as a subgroup of $GL(dim(A), F)$. Also the order of $Aut(A)$ is available.

A canonical form $Can(A)$ for A is a nilpotent structure constants table for A which is unique for the isomorphism type of A; that is, two algebras A and B are isomorphic if and only if $Can(A) = Can(B)$ holds. Hence the canonical form can be used to solve the isomorphism problem.

The Modular Isomorphism Problem

The modular isomorphism problem asks whether $\mathbb{F}_pG \cong \mathbb{F}_pH$ implies that $G \cong H$ for two p-groups G and H and $\mathbb{F}$ the field with p elements. This problem was open for a long time until first counterexamples for the prime $p = 2$ were found in [GLMdR22]. It remains open for odd primes and many other interesting classes of groups.

Computational approaches have been used to investigate the modular isomorphism problem. Based on an algorithm by Roggenkamp and Scott [RS93], Wursthorn [Wur93] described an algorithm for checking the modular isomorphism problem; that is, he described an algorithm for checking whether two modular group algebras $\mathbb{F}G$ and $\mathbb{F}H$ are isomorphic, where G and H are finite p-groups. This algorithm has been implemented in C by Wursthorn and has been applied to the groups of order dividing 2^7 without finding a counterexample, see [BKRW99]. The implementation of Wursthorn appears lost, but is in any case not publicly available.

This package contains an implementation of the new algorithm described in [Eic08] for checking isomorphism of modular group algebras. It is based on the fact that the Jacobson radical $J(FG)$ is nilpotent if FG is a modular group algebra for G a finite p-group and FG is isomorphic to FH if and only if the radicals $J(FG)$ and $J(FH)$ are isomorphic. Hence the automorphism group and canonical form algorithm of this package apply and can be used to solve the isomorphism problem for modular group algebras of finite p-groups.

The methods of this package have been used to study the modular isomorphism problem for the groups of order dividing 3^6 and 2^8 ([Eic08]) and for the groups of order 2^9 ([EK11]). It was later used to study also groups of order 3^7 and 5^6 ([MM22]).

A property of a group G is called **F-invariant**, if an isomorphism of F-algebras $FG \cong FH$ implies the same property for H. In the context of the Modular Isomorphism Problem, if G is a finite p-group, then an $\mathbb{F}_p$-invariant is simply called **invariant**. Many invariants of G are known and the package provides functions for them, as well as programs which easily allow to compare all the implemented invariants quickly for a given list of groups.

It also remains open, if replacing the field $\mathbb{F}_p$ in the Modular Isomorphism Problem with a bigger field of characteristic p will change the outcome of the problem for a given pair of groups. The package includes several functions which allow to investigate this question by applying the algorithm for the same groups varying the field.

A nilpotent quotient algorithm

Given a finitely presented associative algebra A over an arbitrary field F, this package contains an algorithm to determine a nilpotent structure constants table for the class-c nilpotent quotient of A, i.e. the algebra A/A^{c+1}. See [Eic11] for details on the underlying algorithm.

Kurosh Algebras

Let $F(d,F)$ denote the free non-unital associative algebra on d generators over the field F. Then

$$A(d,n,F) = F(d,F)/\langle\langle w^n \mid w \in F(d,F)\rangle\rangle$$

is the **Kurosh Algebra** on d generators of exponent n over the field F. Kurosh Algebras can be considered as an algebra-theoretic analogue to Burnside groups.

This package contains a method that allows to determine $A(d,n,F)$ for given d, n, F. This can also be used to determine $A(d,n,F)$ for all fields of a given characteristic. We refer to [Eic11] for details on the algorithms.

This package also contains a database of Kurosh Algebras that have been determined with the methods of this package.

2 Tables

Finite dimensional algebras can be described by structure contants tables. For nilpotent algebras it is not neccessary to store a full structure contants table. To use this feature, we introduce **nilpotent structure constants tables** or just **nilpotent tables** for short. These are used heavily throughout the package.

2.1 Nilpotent tables

Let A be a finite-dimensional nilpotent associative algebra over a field F. Let $(b_1, \ldots, b_d)$ be a **weighted basis** of A; that is, a basis with weights $(w_1, \ldots, w_d)$ satifying that $A^j = \langle b_i \mid w_i \geq j \rangle$. Let

$$b_i b_j = \sum_k a_{i,j,k} b_k.$$

The nilpotent table T for A (with respect to the basis $(b_1, \ldots, b_d)$) is a record with the following entries.

dim
: the dimension d of A;

fld
: the field F of A;

wgs
: the weights $(w_1, \ldots, w_d)$;

rnk
: the rank e of A (i.e. the dimension of A/A^2).

wds
: a list of length d with holes; If the ith entry is bounded, then it is of the form $[k, l]$. In this case, $w_i > 1$ and $b_i = b_k b_l$ and $w_k = 1$ and $w_l = w_i - 1$ holds.

tab
: a partial structure contants table for A; If $tab[i][j][k]$ is bounded, then it is $a_{i,j,k}$. Note that either a full vector $tab[i][j]$ is given or $tab[i][j]$ is unbounded. The entry $tab[i][j][k]$ is available for $1 \leq i, j \leq e$ and if $wds[i]$ is unbounded.

com
: optional; If this is bounded, then it is a boolean. If this boolean is true, then the algebra is assumed to be commutative.

In a nilpotent table not all structure constants are readily available. The following function determines the structure constants for the product $b_i b_j$. If the global variable *STORE* is true, then the function stores the computed entry in the table.

1 ► `GetEntryTable( T, i, j )` F

The result of the multiplication of the elements `v` and `w` in `T` can be obtained using

2 ▶ `MultByTable( T, v, w )` F

We consider two nilpotent tables as equal, if they would be equal if the full set of structure constants tables would be bound. The following function provides an effective check for this.

3 ▶ `CompareTables( T1, T2 )` F

A nilpotent table contains redundant information and hence can be inconsistent. The next functions can be used to check this to some extend.

4 ▶ `CheckAssociativity( T )` F

Checks that $(b_i b_j)b_k = b_i(b_j b_k)$ for all i, j, k. Note that this may be time-consuming.

5 ▶ `CheckCommutativity( T )` F

Checks whether T defines a commutative algebra and sets the entry *com* accordingly.

6 ▶ `CheckConsistency( T )` F

Checks that *wds* and *tab* are compatible. This assumes that CheckAssociativity returns true.

All later described algorithms of this package assume that the tables considered are fully consistent.

```
gap> T := rec( dim := 3,
               fld := GF(2),
               rnk := 2,
               wgs := [ 1, 1, 2 ],
               wds := [ ,, [ 2, 1 ] ],
               tab := [] );;
gap> T.tab[1] := [[0,0,0],[0,0,1]] * One(T.fld);;
gap> T.tab[2] := [[0,0,1],[0,0,0]] * One(T.fld);;
gap> GetEntryTable( T, 3, 1 );
[ 0*Z(2), 0*Z(2), 0*Z(2) ]
```

2.2 Algebras in the GAP sense

We provide functions to convert back and forth between algebras in the GAP sense and nilpotent tables.

1 ▶ `AlgebraByTable( T )` F
▶ `NilpotentTable( A )` F

Note that the second function fails if A is not nilpotent.

For modular group algebras of p-groups, the group algebra itself is not nilpotent (as it contains a unit), but its Jacobson radial is. The following function determines a nilpotent table for the Jacobson radical.

2 ▶ `NilpotentTableOfRad( FG )` F

```
gap> A := GroupRing(GF(2), SmallGroup(8,3));
<algebra-with-one over GF(2), with 3 generators>
gap> NilpotentTableOfRad(A);
rec( dim := 7, fld := GF(2), rnk := 2,
  tab :=
    [
      [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
          [ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
          [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
          [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
```

```
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
        [ [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],,
        [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
            [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ] ],
    wds := [ ,, [ 1, 2 ],, [ 1, 4 ], [ 2, 4 ], [ 1, 6 ] ],
    wgs := [ 1, 1, 2, 2, 3, 3, 4 ] )
```

2.3 Tables for the Modular Isomorphism Problem

A special kind of nilpotent table is available in the context of the Modular Isomorphism Problem. Let G be a finite group. F a field of characteristic p and $I(FG)$ the augmentation ideal of FG which equals its radical and is a nilpotent ideal. Two functions allow to compute class-n quotient of $I(FG)$, i.e. $I(FG)/I(FG)^{n+1}$. The output is a nilpotent table for this quotient, but in addition to the standard entries of a nilpotent table it contains further entries, which allow more efficient computations and can also facilitate manual calculations. This allows to determine the class-n quotient of the augmentation ideal without computing the full augmentation ideal using `NilpotentTableOfRad`. The corresponding table can be computed by

1 ▶ `TableOfRadQuotient( FG, n )` F

or

2 ▶ `ModIsomTable( G, n, [f] )` F

Here `ModIsomTable(G, n)` will produce the quotient with respect to the algebra $\mathbb{F}_pG$, while `ModIsomTable( G, n, f )` will do the same for the algebra $\mathbb{F}_{p^f}G$. respect to the algebra $\mathbb{F}_pG$.

The components `dim`, `fld`, `rnk`, `tab`, `wgs`, `wds` remain unchanged from a usual nilpotent table. The additional components are `commwords`, `powwords` and `pre`. These new components contain additional information on precisely which basis of $I(FG)/I(FG)^{n+1}$ is used and what the result of multiplying basis elements is. We explain how users can understand how the basis looks and how they can multiply two elements in the algebra. The components `T.commords` and `T.powwords` contain information on how the elements of the basis behave with respect to commutators and p-th powers. The component `T.pre` contains information on the construction of the basis and we describe it in more detail.

The dimension of $I(FG)/I(FG)^{n+1}$ is recorded in `T.dim`. The basis of $I(FG)/I(FG)^{n+1}$ is found as in the theory of Jennings going back to [Jen41], cf. [MM22] for the information needed here. The elements of G chosen to provide the basis of subsequent quotients of dimension subgroups are recorded in `T.pre.jen.pcgs`. Let us call these elements $g_1, \ldots, g_m$. Note that $|G| = p^m$. The weights of the elements $g_1 - 1, \cdots, g_m - 1$ are recorded in `T.pre.jen.weights`. If now r is an integer smaller than `T.dim+1`, then the r-th element of the basis of $I(FG)/I(FG)^{n+1}$ is $(g_1 - 1)^{e_1} \cdot \ldots (g_m - 1)^{e_m}$ where `[e_1,...,e_m] = T.pre.exps[r]`. The weight of this element is recorded in `T.wgs[r]` and also `T.pre.weights[r]`. Moreover, the positions of $g_1 - 1, \cdots, g_m - 1$ in the chosen basis of `T` are recorded in `T.pre.poswone`. We elaborate using an example.

We consider the group $G = SmallGroup(3^7, 19)$. The following calculation shows that $I(FG)/I(FG)^9$ has dimension 135 and that the full augmentation ideal $I(FG)$ has dimension 2186.

```
gap> G := SmallGroup(3^7, 19);;
gap> T := ModIsomTable(G, 8);;
gap> T.dim;
135

gap> FG := GroupRing(GF(3), G);;
gap> TT := TableOfRadQuotient(FG, 8);;
gap> TT.dim;
135

gap> T := ModIsomTable(G, 38);;
gap> T.dim;
2186

gap> T := ModIsomTable(G, 39);;
gap> T.dim;
2186
```

We next consider an example how the basis used can be recognized.

```
gap> G := DihedralGroup(8);;
gap> T := ModIsomTable(G, 4);;
gap> T.dim;
7
gap> pcgs := T.pre.jen.pcgs;
Pcgs([ f1, f2, f3 ])
gap> List(pcgs, Order);
[ 2, 4, 2 ]
gap> pcgs[3] in Center(G);
true
gap> T.pre.exps{[1..7]};
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 1, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 1 ], [ 0, 1, 1 ],
  [ 1, 1, 1 ]]
```

We conclude that $I(kG)/I(kG)^5$ is 7-dimensional and if we denote by a a reflection and by b a non-central rotation in G, then the basis used by `T` is: $(a-1)$, $(b-1)$, $(a-1)(b-1)$, (b^2-1), $(a-1)(b^2-1)$, $(b-1)(b^2-1)$, $(a-1)(b-1)(b^2-1)$.

Say continuing the previous example we want to multiply $(b-1)+(a-1)(b-1)+(a-1)(b^2-1)$ and $(a-1)+(b-1)+(b^2-1)$.

```
gap> v := Z(2)^0*[0,1,1,0,1,0,0];
[ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ]
gap> w := Z(2)^0*[1,1,0,1,0,0,0];
[ Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ]
gap> MultByTable(T,v,w);
[ 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ]
```

So the result is $(a-1)(b-1)+(a-1)(b^2-1)$.

To facilitate the translation of elements of the group algebra and a corresponding table of a quotient of the augmentation ideal the functions

3 ▶ `MIPElementTableToAlgebra( v, T, FG )` F

and

4 ▶ `MIPElementAlgebraToTable( el, FG, T )` F

can be used. In the second function of course only a possible representative of `v` in `FG` is provided. Also, only elements from the augmentation ideal of `FG` can be represented using `MIPElementTableToAlgebra`. These functions can be used for instance to obtain representatives in the same class modulo a power of the augmentation ideal which are more practical to work with, as the following example shows.

```
gap> G := SmallGroup(3^7, 19);
<pc group of size 2187 with 7 generators>
gap> T := ModIsomTable(G, 4);;
gap> FG := GroupRing(GF(3), G);
<algebra-with-one over GF(3), with 7 generators>
gap> iota := Embedding(G, FG);
<mapping: Group( [ f1, f2, f3, f4, f5, f6, f7
 ] ) -> AlgebraWithOne( GF(3), ... ) >
gap> a := (T.pre.jen.pcgs[1])^iota;
(Z(3)^0)*f1
gap> b := (T.pre.jen.pcgs[2])^iota;
(Z(3)^0)*f2
gap> z := One(FG);
(Z(3)^0)*<identity> of ...
gap> r := (z + (a-z)*(b-z) )^-1;;
gap> Size(Support(r-z));
1376
gap> el := MIPElementAlgebraToTable(r-z, FG, T);
[ 0*Z(3), 0*Z(3), 0*Z(3), Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3),
  0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3),
  0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ]
gap> MIPElementTableToAlgebra(el, T, FG);
(Z(3))*<identity> of ...+(Z(3)^0)*f3+(Z(3)^0)*f1^2+(Z(3))*f1*f2+(Z(3))*f1*f3+(
Z(3)^0)*f2^2+(Z(3))*f2*f3+(Z(3)^0)*f1^2*f2+(Z(3)^0)*f1*f2^2+(Z(3)^
0)*f1*f2*f3+(Z(3)^0)*f1^2*f2^2
```

We illustrate the information in `T.pre.poswone`:

```
gap> d := (T.pre.jen.pcgs[4])^iota;
(Z(3)^0)*f4
gap> el := MIPElementAlgebraToTable(d-z, FG, T);
[ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3),
  0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3),
  0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ]
gap> Position(last, Z(3)^0);
11
gap> T.pre.poswone[4];
11
```

3 Automorphism groups and Canonical Forms

We refer to [Eic08] for background on the algorithms used in this Chapter. Throughout the chapter, we assume that F is a finite field.

3.1 Automorphism groups

Let T be a nilpotent table over F. The following function can be used to determine the automorphism group of the algebra described by T. The automorphism group is determined as subgroup of $GL(T \cdot dim, T \cdot fld)$ given by generators and its order. There is a variation available to determine the automorphism group of a modular group algebra FG, where F is a finite field and G is a p-group.

1 ▶ `AutGroupOfTable( T )` F
▶ `AutGroupOfRad( FG )` F

In both cases, the automorphism group is described by a record. The matrices in the lists *glAutos* and *agAutos* generate together the automorphism group. The matrices in *agAutos* generate a p-group. The entry *size* contains the order of the automorphism group.

3.2 Canonical forms

Let T be a nilpotent table. The following function can be used to determine the automorphism group of T if the underlying field of T is finite. The canonical form is a nilpotent table which is unique for the isomorphism type of the algebra defined by T. Again there a variation available for modular group algebras.

1 ▶ `CanonicalFormOfTable( T )` F
▶ `CanonicalFormOfRad( FG )` F

The automorphism group of T is determined as a side-product of computing the canonical form. The following functions can be used to return both.

2 ▶ `CanoFormWithAutGroupOfTable( T )` F
▶ `CanoFormWithAutGroupOfRad( FG )` F

In both cases, these functions return a record with entries *cano* and *auto*.

3.3 Example of canonical form computation

We compute the automorphism group and a canonical form for the modular group algebra of the dihedral group of order 8.

```
gap> A := GroupRing(GF(2), SmallGroup(8,3));;
gap> T := TableByWeightedBasisOfRad(A);;
gap> C := CanoFormWithAutGroupOfTable(T);;

# check that the canonical form is not equal to T
gap> CompareTables(C.cano, T);
false
```

```
# the order of the automorphism group
gap> C.auto.size;
512

# the entries of the canonical table as far as they are bounded
gap> C.cano.tab;
[ [ <a GF2 vector of length 7>, <a GF2 vector of length 7>,
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
  [ <a GF2 vector of length 7>, <a GF2 vector of length 7>,
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
  [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
  [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ],
  [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ] ],
  [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ] ]
```

4 The modular isomorphism problem

Applications of the methods in this package include the study of the modular isomorphism problems for the groups of small order from the SmallGroupLibrary - first for groups of order dividing 2^8, 3^6 and 2^9 [Eic07,EKo11] and later also 3^7 and 5^6 [MM22]. This section contains the functions used for this purpose as well as an overview of how the Modular Isomorphism Problem can be studied for any set of groups using on one hand group-theoretical invariants and on the other hand the canonical form of nilpotent algebras.

4.1 Computing bins and checking bins

A set of groups which share all the group-theoretical invariants implemented in the package is called **bin**. To determine such bins the main function available is:

1 ► `BinsByGT( p, n, [L], [false] )` F

If the function is called as `BinsByGT(p, n)`, then it returns a partition of the list `[1..NumberSmallGroups(p^n)]` into sublists so that the groups in the corresponding lists share all the group-theoretical invariants, i.e. the modular group algebras of two groups `SmallGroup(p^n, i)` and `SmallGroup(p^n, j)` over the field $\mathbb{F}_p$ can not be isomorphic if i and j are in different lists.

If the function is called as `BinsByGT(p, n, L)`, then `L` must be a list of groups of order p^n and the function will return a partition of the groups of `L` which share all the group-theoretical invariants. Alternatively, `L` can be a list of group Id's of groups of order p^n.

If the function is called as `BinsByGT(p, n, L, false)` then `L` must be a list of groups of order p^n and `false` disactivates the calculation of the dimensions of the second cohomology groups. This can be switched off as for some type of groups `GAP` can not apply the needed functions and since computing the second cohomology groups is arguably the hardest of the invariants to test manually.

Several variations of `BinsByGT` are available. The first two apply to the case when a list of groups is being studied instead of group Id's.

2 ► `MIPSplitGroupsByGroupTheoreticalInvariants ( L )` F

does the same as `BinsByGT(p,n,L)` but computes the numbers p and n itself. The input variable must be a list of groups of the same order. Similarly

3 ► `MIPSplitGroupsByGroupTheoreticalInvariantsNoCohomology(L)` F

computes `BinsByGT(p, n, L, false)`.

Moreover, all the three functions described before have variations where only those group-theoretical invariants are computed that are known to be $\mathbb{F}$-invariants over any field $\mathbb{F}$ of characteristic p. The input and output of these functions is exactly as for the three previous functions.

4 ► `BinsByGTAllFields(p, n , [L], [false])` F

5 ► `MIPSplitGroupsByGroupTheoreticalInvariantsAllFields(L)` F

6 ► `MIPSplitGroupsByGroupTheoreticalInvariantsAllFieldsNoCohomology(L)` F

The group-theoretical invariants used by the function `BinsByGT` and its variations are described below. Moreover, `GAP` prints more or less information on the progress inside these functions, if `InfoModIsom` is set to 1 or 0, respectively. Examples of the use of the functions are included below.

The main function to apply the algorithm computing the canonical form of nilpotent algebras in the context of the Modular Isomorphism Problem is:

7 ► `MIPSplitGroupsByAlgebras( p, n, bin, [f] )` F

8 ► `MIPSplitGroupsByAlgebras(bin , [f] )` F

If `MIPSplitGroupsByAlgebras(p, n, bin, [f])` is called then the algebras of groups of order p^n with group Id's contained in `bin` are studied. The underlying field is of order p^f or, if `f` is not given, of order p.

If the function is called as `MIPSplitGroupsByAlgebras(bin, [f])` then `bin` must be a list of groups of the same prime power order and the function studies the group algebras of the groups in `bin` over the field with p^f elements or, if no `f` is given, of order p.

More precisely, in the first case when `bin` is a list of integers, for $i \in bin$ let G_i denote `SmallGroup(p^n, i)`. In the second case let G_i just runs through the groups contained in `bin`. Denote by A_i the augmentation ideal of $\mathbb{F}G_i$ where $\mathbb{F}$ is the field of order p^f or simply p, if `f` is not given. The function computes and compares the canonical forms of the algebras A_i/A_i^j for every $i \in bin$ and increasing natural number j.

At each level j it splits the current bins into sub-bins according to the different canonical forms of A_i/A_i^j. Bins of length 1 are then discarded.

The function returns if no further bins are available and provides information at which level the splitting of the bins took place.

For more evolved calculations one can use the function

9 ► `MIPBinSplit(p, n, k, start, step, L, [f])`

Given a list `L` of small group library Id's or a list of groups of order p^n this functions checks isomorphism of the associated modular group algebras using canonical forms for the quotients of the augmentation ideals A of $\mathbb{F}G$. Here $\mathbb{F}$ is either $\mathbb{F}_p$ or $\mathbb{F}_{p^f}$, if `f` is given. The parameter `max` is an integer or `false` that determines the maximal quotients A/A^{max} to be checked (if `false` is given as input, then the quotients are enlarged until non-isomorphic quotients are found or eventually the full augmentation ideal will be checked). The parameter `start` specifies which quotients A/A^{start} are precomputed. The parameter `step` determines in which steps the quotients are enlarged if necessary during the isomorphism check. The output is a record containing three entries: `bins` contains all the groups, for which the non- isomorphism of the associated modular group algebras could not be determined; `splits` contains all the groups, for which the associated group algebras were determined to be non-isomorphic (and the first non-isomorphic quotient); `time` contains the time used for the computation (in milliseconds).

For big algebras all of these function can use a lot of time and memory. To have a better idea on the progress of the calculations one should set `InfoModIsom` to 1.

We first show how to study a fixed order:

```
gap> bins := BinsByGT(2,6);
[ [ 156, 158, 160 ], [ 155, 157 ], [ 173, 176 ], [ 179, 180 ], [ 20, 22 ] ]
gap> List(bins, bin -> MIPSplitGroupsByAlgebras(2, 6, bin));
[ rec( bins := [  ], splits := [ [ 7, [ 156, 158, 160 ] ] ], time := 2195 ),
  rec( bins := [  ], splits := [ [ 7, [ 155, 157 ] ] ], time := 1505 ),
  rec( bins := [  ], splits := [ [ 7, [ 173, 176 ] ] ], time := 3294 ),
  rec( bins := [  ], splits := [ [ 7, [ 179, 180 ] ] ], time := 3233 ),
  rec( bins := [  ], splits := [ [ 4, [ 20, 22 ] ] ], time := 160 ) ]
```

This shows that the Modular Isomorphism Problem has a positive answer for groups of order 64 for the field $\mathbb{F}_2$. The result means e.g. that the smallest quotients (of Loewy layers) such that the augmentation ideals A_1 and A_2 of the groups algebras over $\mathbb{F}_2$ of the groups `SmallGroup(64, 156)` and `SmallGroup(64, 158)` are not isomorphic are A_1/A_1^8 and A_2/A_2^8. These are A_1/A_1^5 and A_2/A_2^5 for the groups `SmallGroup(64, 20)` and `SmallGroup(64, 22)`.

The following shows that the problem also has a positive answer for the group algebras of groups of order 64 over the field $\mathbb{F}_4$. Note that for the groups `SmallGroup(64, 20)` and `SmallGroup(64, 22)` one has to consider deeper quotients in this case.

```
gap> bins := BinsByGTAllFields(2,6);
[ [ 156, 158, 160 ], [ 155, 157 ], [ 173, 176 ], [ 179, 180 ], [ 104, 105 ],
  [ 13, 14 ], [ 20, 22 ], [ 18, 19 ] ]
gap> List(bins, bin -> MIPSplitGroupsByAlgebras(2, 6, bin, 2));
[ rec( bins := [  ], splits := [ [ 7, [ 156, 158, 160 ] ] ], time := 34833 ),
  rec( bins := [  ], splits := [ [ 7, [ 155, 157 ] ] ], time := 22479 ),
  rec( bins := [  ], splits := [ [ 7, [ 173, 176 ] ] ], time := 9806 ),
  rec( bins := [  ], splits := [ [ 7, [ 179, 180 ] ] ], time := 7819 ),
  rec( bins := [  ], splits := [ [ 4, [ 104, 105 ] ] ], time := 2226 ),
  rec( bins := [  ], splits := [ [ 6, [ 13, 14 ] ] ], time := 707 ),
  rec( bins := [  ], splits := [ [ 6, [ 20, 22 ] ] ], time := 3917 ),
  rec( bins := [  ], splits := [ [ 6, [ 18, 19 ] ] ], time := 2891 ) ]
```

The other functions allow to study the problem for groups not coming from the library. The following groups are studied in [GLM24].

```
R := SmallGroup(64, 19);
Q := SmallGroup(64, 18);

DR := DirectProduct(R,Q);
GDR := GeneratorsOfGroup(DR);
z1 := GDR[3];
z2 := GDR[9];
N := Group(z1*z2^(-1));
G := DR/N;

DR := DirectProduct(Q,Q);
GDR := GeneratorsOfGroup(DR);
z1 := GDR[3];
z2 := GDR[9];
N := Group(z1*z2^(-1));
H := DR/N;

gap> MIPSplitGroupsByGroupTheoreticalInvariantsAllFields([G,H]);
[ [ Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]),
      Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]) ] ]
```

```
# the groups can not be split over all fields by group-theoretical invariants

gap> MIPSplitGroupsByAlgebras([G,H]);
rec( bins := [  ],
  splits :=
    [
      [ 4,
          [ Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]),
              Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]) ] ]
     ], time := 44473 )

# over the field of 2 elements it is enough to consider
# the 5-th power of the augmentation ideal
```

The program does not finish in a very reasonable time, if we run it over the field $\mathbb{F}_4$, but we can still check that it is not enough to factor out only the 5th power of the augmentation ideal in this case. One option is to use info level to do this and the other to use `MIPBinsSplit`:

```
gap> SetInfoLevel(InfoModIsom, 1);
gap> MIPSplitGroupsByAlgebras([G,H], 2);
#I  Refine bin
#I    Weights yields bins [ [ 1, 2 ] ]
#I    Layer 1 yields bins [ [ 1, 2 ] ]
#I  layer 2 of dim 15 aut group has order 2961100800 * 2^0
#I     cover is determined
#I     dim(M) = 16 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 2 of dim 15 aut group has order 2961100800 * 2^0
#I     cover is determined
#I     dim(M) = 16 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 2 yields bins [ [ 1, 2 ] ]
#I  layer 3 of dim 39 aut group has order 2937600 * 2^88
#I     cover is determined
#I     dim(M) = 29 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 3 of dim 39 aut group has order 2937600 * 2^88
#I     cover is determined
#I     dim(M) = 29 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
```

```
#I    Layer 3 yields bins [ [ 1, 2 ] ]
#I  layer 4 of dim 81 aut group has order 2937600 * 2^240
#I     cover is determined
#I     dim(M) = 51 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 4 of dim 81 aut group has order 2937600 * 2^240
#I     cover is determined
#I     dim(M) = 51 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 4 yields bins [ [ 1, 2 ] ]
#I  layer 5 of dim 145 aut group has order 7200 * 2^496
#I     cover is determined
#I     dim(M) = 73 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 5 of dim 145 aut group has order 7200 * 2^496
#I     cover is determined
#I     dim(M) = 73 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 5 yields bins [ [ 1, 2 ] ]
#I  layer 6 of dim 231 aut group has order 7200 * 2^800
#I     cover is determined
#I     dim(M) = 95 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
^CError, user interrupt in
  AddRowVector( u, GetEntryTable( T, i, j ), v[i] * w[j]
 ); at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/tables/tables.gi:87 called from
MultByTable( Q, new[Q.wds[i][1]], new[Q.wds[i][2]]
 ) at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/autiso/induc.gi:135 called from
InduceAutoToQuot( Q, G.agAutos[i]
 ) at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/autiso/induc.gi:151 called from
InduceAutosToQuot( G, Q
 ); at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/autiso/autiso.gi:57 called from
ExtendCanoForm( tabs[i], j
 ); at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/grpalg/chkbins.gi:117 called from
MIPBinSplit( p, n, false, start, step, list, f
 ) at /home/leo/gap-4.10.1/pkg/modisom-2.5.3/gap/grpalg/chkbins.gi:177 called from
...  at *stdin*:39
you can 'return;'
```

```
brk> quit;   # this was not progressing for several hours

gap> Size(G) = Size(H);
true
gap> Size(G) = 2^10;
true

gap> MIPBinSplit(2, 10, 4, 4, 1, [G,H], 2);
#I  Refine bin
#I    Weights yields bins [ [ 1, 2 ] ]
#I    Layer 1 yields bins [ [ 1, 2 ] ]
#I  layer 2 of dim 15 aut group has order 2961100800 * 2^0
#I     cover is determined
#I     dim(M) = 16 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 2 of dim 15 aut group has order 2961100800 * 2^0
#I     cover is determined
#I     dim(M) = 16 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 2 yields bins [ [ 1, 2 ] ]
#I  layer 3 of dim 39 aut group has order 2937600 * 2^88
#I     cover is determined
#I     dim(M) = 29 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 3 of dim 39 aut group has order 2937600 * 2^88
#I     cover is determined
#I     dim(M) = 29 and dim(U) = 5
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 3 yields bins [ [ 1, 2 ] ]
#I  layer 4 of dim 81 aut group has order 2937600 * 2^240
#I     cover is determined
#I     dim(M) = 51 and dim(U) = 9
#I     extended autos
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I  layer 4 of dim 81 aut group has order 2937600 * 2^240
#I     cover is determined
#I     dim(M) = 51 and dim(U) = 9
#I     extended autos
```

```
#I     computed stabilizer
#I     got quotient
#I     induced autos
#I    Layer 4 yields bins [ [ 1, 2 ] ]
rec(
  bins :=
    [
      [ Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]),
          Group([ f1, f2, f7, f3, f4, f10, f5, f6, f7, f8, f9, f10 ]) ] ],
  splits := [  ], time := 9469981 )
```

4.2 Kernel size

An idea to study the Modular Isomorphism Problem is to define maps on certain quotients of the augmentation ideal A and count the number of elements which map to 0 under this map. The map most typically used for this is a p-power map $A^n/A^{n+m} \to A^{n\cdot p^f}/A^{n\cdot p^f+m}$. This can be done in the package using the function

1 ► `KernelSizePowerMap(T, n, m, l, [f])` F

where `T` is a table as returned by `ModIsomTable` and `n, m, l` are as just described and the calculation is performed over the field $\mathbb{F}_{p^f}$. If `f` is not given, then it is set to 1. We can check for instance the first calculation in [HS06](Section 4.1).

```
gap> G := SmallGroup(64, 20);
<pc group of size 64 with 6 generators>
gap> H := SmallGroup(64, 22);
<pc group of size 64 with 6 generators>
gap> TG := ModIsomTable(G, 5);;
gap> TH := ModIsomTable(H, 5);;
gap> KernelSizePowerMap(TG, 1, 1, 2);
3
gap> KernelSizePowerMap(TH, 1, 1, 2);
1
```

This shows that the group algebras over $\mathbb{F}_2$ are not isomorphic. This argument does however not work over $\mathbb{F}_4$:

```
gap> TG := ModIsomTable(G, 5, 2);;
gap> TH := ModIsomTable(H, 5, 2);;
gap> KernelSizePowerMap(TG, 1, 1, 2);
7
gap> KernelSizePowerMap(TH, 1, 1, 2);
7
```

4.3 The group theoretical invariants

We document here which group-theoretical invariants are used in `BinsByGT` and similar functions.

1 ► `GroupInfo(G)` F

This is an auxiliary function used in other group-theoretical invariants. If `IdGroup` is available in `GAP` for the order of G it returns `IdGroup(G)`. Otherwise it returns `[Size(G), AbelianInvariants(G)]`.

We now describe the invariants in the order they appear in `BinsByGT`. First the isomorphism types of G/G' and $Z(G)$, the abelianization and the center of G are used. These are very classical invariants [San85](Theorems 6.12, 6.7). We next list the other functions which are applied, which are all small functions written for the package:

2 ► `CenterDerivedInfo(G)` F

calculates the isomorphism types of $Z(G) \cap G'$ and $Z(G)/Z(G) \cap G'$ [San85](Theorem 6.11).

3 ► `SandlingInfo(G)` F

calculates several invariants coming from the small groups algebra which was first used to study the Modular Isomorphism Problem in [San89]. Namely, for $\gamma_i(G)$ the i-th term of the lower central series of G it returns the `GroupInfo` for $G/\gamma_2(G)^p\gamma_3(G)$ [San89], the `GroupInfo` of $G/\gamma_2(G)^p\gamma_4(G)$, if G is 2-generated (mentioned in [Bag99], proved in [MM22]) and if the derived subgroup of G is elementary abelian and the Jennings series of G has length at most $2p$, it returns `GroupInfo` for the Frattini subgroup of G [HS06](p.16). This function is not applied in `BinsByGTAllFields` and its variations.

4 ► `JenningsInfo(G)` F

Denoting by $D_i(G)$ the i-th member of the Jennings series, this function returns `GroupInfo` for the quotients $D_i(G)/D_{i+1}(G)$, $D_i(G)/D_{i+2}(G)$, $D_i(G)D_{2i+1}(G)$ for meaningful values of i (results [Jen41] [PS72] [RS83]) and if p is odd also for $G/D_4(G)$ [Her07]. For `BinsByGTAllFields` and its variations only the quotients $D_i(G)/D_{i+1}(G)$ are computed.

5 ► `JenningsDerivedInfo(G)` F

computes $D_i(G')/D_{i+1}(G')$ for all i [San85](Lemma 6.26).

6 ► `BaginskiInfo(G)` F

For $N = C_G(G'/\Phi(G'))$, where $\Phi(G)$ denotes the Frattini subgroup, if G/N is cyclic, it returns the `GroupInfo` for $N/\Phi(G')$ and $G/\Phi(N)$ [Bag99]. This function is not applied in `BinsByGTAllFields` and its variations.

7 ► `BaginskiCarantiInfo(G)` F

returns the nilpotency class of $G/\Phi(G')$ [BC88](Proposition 2.1). This function is not applied in `BinsByGTAll-Fields` and its variations.

8 ► `NilpotencyClassInfo(G)` F

returns the nilpotency class of G, when the exponent of G is p or the class equals 2 or the derived subgroup is cyclic [BK07].

9 ► `Theorem41MS22(G)` F

If p is odd, G is 2-generated, the nilpotency class of G is 3 and $\gamma_3(G)$ has exponent p, it returns the isomorphism types of $\gamma_2(G)$ and $\gamma_3(G)$ [MS22](Theorem 4.1). This function is not applied in `BinsByGTAllFields` and its variations.

10 ► `CyclicDerivedInfo(G)` F

If p is odd and G' is cyclic this returns several invariants contained in [GLdRS22] [GLdRS23a]. Namely, $D_i(C_G(G'))/D_{i+1}(C_G(G'))$ for all i, the exponent of $C_G(G')$, the isomorphism type of $C_G(G')/G'$ and the `GroupInfo` for $G/R_1(\gamma_3(G))$ and $G/R_3(C_G(G'))$. Here $R_i(G)$ denotes the subgroup of G generated by the p^i-th powers in G. If G is additionally 2-generated, it also computes the `GroupInfo` for $C_G(G')$ and the type invariants of G (cf. [GLdRS22] for the definition). For `BinsByGTAllFields` and its variations, the `GroupInfo` for $G/R_1(\gamma_3(G))$ and $G/R_3(C_G(G'))$ is not computed.

11 ► `MaximalAbelianDirectFactor(G)` F

computes the maximal abelian direct factor of G [GL24]. In "BinsByGTAllFields" and its variations it computes the maximal elementary abelian direct factor instead [MSS23].

12 ► `NormalSubgroupsInfo(G)` F

computes some of the sections of G which are canonical following [GL24](Lemma 3.6). The starting canonical group is the derived subgroup.

13 ▶ `IsCoveredByTheory(G)` F

determines, whether G belongs to certain classes for which the Modular Isomorphism Problem has been solved positively. Namely if G is a 2-group of maximal class [Bag92], p is odd, $G \leq p^{p+1}$, G has a maximal abelian subgroup and G is of maximal class [BC88], $[G : Z(G)] = p^2$ [Dre89], G is a 2-generated 2-group of nilpotency class 2 [BdR21], p is odd, G' is elementary abelian, the nilpotency class of G is 3 and either $C_G(G')$ is a maximal subgroup of G and abelian [MS22](Theorem 3.3) or a G is 3-generated and a certain condition holds on the commutator map modulo the second center of G [MS22](Theorem 3.5), G is a 2-group with cyclic center and dihedral central quotient [MSS23], G is a 2-group of nilpotency class 2 with cyclic center [GLM24], p is odd, G' is cyclic and $R_2(G/G')$ is cyclic [GLdRS23a](Proposition 3.7). For `BinsByGTAllFields` and its variations it only checks, if G is a 2-group of maximal class, $[G : Z(G)] = p^2$ or G is a 2-group with cyclic center and dihedral central quotient.

14 ▶ `DimensionTwoCohomology(G)` F

computes the dimensions of the second cohomology group $H^2(FG, F)$ and of the second Hochschild cohomology group $HH^2(FG) = H^2(FG, FG)$.

15 ▶ `ConjugacyClassInfo(G)` F

Computes the number of conjugacy classes which are p^i-th powers, for all i (Kuelshammer) and the number of conjugacy classes of p^i-th powers which come from conjugacy classes of the same order (Parmenter-Polcino Milies) [HS06](Section 2.2) and the dimension of the first Hochschild cohomology group which equals $\sum_{g^G} \log_p(C_G(g)/\Phi(C_G(g))$ where the sum runs over all the conjugacy classes of G [HS06](Section 2.6).

16 ▶ `SubgroupsInfo(G)` F

Computes the number of conjugacy classes of maximal elementary abelian subgroups of rank 1,2... The return is a list of integers which contains this number for all ranks until the maximal possible. This is based on results of Quillen [HS06](Section 2.5).

5 Nilpotent Quotients

This chapter contains a description of the nilpotent quotient algorithm for associative finitely presented algebras. We refer to [Eic11] for background on the algorithms used in this Chapter.

5.1 Computing nilpotent quotients

Let A be a finitely presented algebra in the GAP sense. The following function can be used to determine the class-c nilpotent quotient of A. The quotient is described by a nilpotent table.

1 ▶ `NilpotentQuotientOfFpAlgebra( A, c )` F

The output of this function is a nilpotent table with some additional entries. In particular, there is the additional entry *img* which describes the images of the generators of A in the nilpotent table.

5.2 Example of nilpotent quotient computation

```
gap> F := FreeAssociativeAlgebra(GF(2), 2);;
gap> g := GeneratorsOfAlgebra(F);;
gap> r := [g[1]^2, g[2]^2];;
gap> A := F/r;;
gap> NilpotentQuotientOfFpAlgebra(A,3);
rec( def := [ 1, 2 ], dim := 8, fld := GF(2),
  img := [ <a GF2 vector of length 8>, <a GF2 vector of length 8> ],
  mat := [ [  ], [  ] ], rnk := 2,
  tab :=
    [ [<a GF2 vector of length 8>, <a GF2 vector of length 8>,
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0 ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
      [ <a GF2 vector of length 8>, <a GF2 vector of length 8>,
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ],
      [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ] ],
      [ [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2), 0*Z(2) ],
        [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2) ] ]],
  wds := [ ,, [ 2, 1 ], [ 1, 2 ], [ 1, 3 ], [ 2, 4 ], [ 2, 5 ], [ 1, 6 ] ],
  wgs := [ 1, 1, 2, 2, 3, 3, 4, 4 ] )
```

6 Relatively free Algebras

As described in [Eic11], the nilpotent quotient algorithm also allows to determine certain relatively free algebras; that is, algebras that are free within a variety.

6.1 Computing Kurosh Algebras

1 ▶ `KuroshAlgebra( d, n, F )` F

determines a nilpotent table for the largest associative algebra on d generators over the field F so that every element a of the algebra satisfies $a^n = 0$.

2 ▶ `ExpandExponentLaw( T, n )`

suppose that T is the nilpotent table of a Kurosh algebra of exponent n defined over a prime field. This function determines polynomials describing the corresponding Kurosh algebras over all fields with the same characteristic as the prime field.

6.2 A Library of Kurosh Algebras

The package contains a library of Kurosh algebras. This can be accessed as follows.

1 ▶ `KuroshAlgebraByLib(d, n, F)` F

At current, the library contains the Kurosh algebras for $n = 2$, $(d,n) = (2,3)$, $(d,n) = (3,3)$ and $F = \mathbb{Q}$ or $|F| \in \{2,3,4\}$, $(d,n) = (4,3)$ and $F = \mathbb{Q}$ or $|F| \in \{2,3,4\}$, $(d,n) = (2,4)$ and $F = \mathbb{Q}$ or $|F| \in \{2,3,4,9\}$, $(d,n) = (2,5)$ and $F = \mathbb{Q}$ or $|F| \in \{2,3,4,5,8,9\}$.

6.3 Example of accessing the library of Kurosh algebras

```
gap> KuroshAlgebra(2,2,Rationals);
... some printout ..
rec( bas := [ [ 1, 0, 0, 0 ], [ 0, 1, 1, 0 ], [ 0, 0, 0, 1 ], [ 0, 1, 0, 0 ] ]
    , com := false, dim := 3, fld := Rationals, rnk := 2,
  tab := [ [ [ 0, 0, 0 ], [ 0, 0, -1 ], [ 0, 0, 0 ] ],
      [ [ 0, 0, 1 ], [ 0, 0, 0 ], [ 0, 0, 0 ] ] ], wds := [ ,, [ 2, 1 ] ],
  wgs := [ 1, 1, 2 ] )
```

Bibliography

[Bag92] C. Bagiński. Modular group algebras of 2-groups of maximal class. *Comm. Algebra*, 20(5):1229–1241, 1992.

[Bag99] C. Bagiński. On the isomorphism problem for modular group algebras of elementary abelian-by-cyclic p-groups. *Colloq. Math.*, 82(1):125–136, 1999.

[BC88] C. Bagiński and A. Caranti. The modular group algebras of p-groups of maximal class. *Canad. J. Math.*, 40(6):1422–1435, 1988.

[BdR21] O. Broche and Á. del Río. The modular isomorphism problem for two generated groups of class two. *Indian J. Pure Appl. Math.*, 52(3):721–728, 2021.

[BK07] C. Bagiński and A. Konovalov. The modular isomorphism problem for finite p-groups with a cyclic subgroup of index p^2. In *Groups St. Andrews 2005. Vol. 1*, volume 339 of *London Math. Soc. Lecture Note Ser.*, pages 186–193. Cambridge Univ. Press, Cambridge, 2007.

[BKRW99] Frauke M. Bleher, Wolfgang Kimmerle, Klaus W. Roggenkamp, and Martin Wursthorn. Computational aspects of the isomorphism problem. In *Algorithmic algebra and number theory (Heidelberg, 1997)*, pages 313–329. Springer, Berlin, 1999.

[Dre89] V. Drensky. The isomorphism problem for modular group algebras of groups with large centres. In *Representation theory, group rings, and coding theory*, volume 93 of *Contemp. Math.*, pages 145–153. Amer. Math. Soc., Providence, RI, 1989.

[Eic08] Bettina Eick. Computing automorphism groups and testing isomorphisms for modular group algebras. *J. Algebra*, 320(11):3895–3910, 2008.

[Eic11] Bettina Eick. Computing nilpotent quotients of associative algebras and algebras satisfying a polynomial identity. *Internat. J. Algebra Comput.*, 21(8):1339–1355, 2011.

[EK11] Bettina Eick and Alexander Konovalov. The modular isomorphism problem for the groups of order 512. In *Groups St Andrews 2009 in Bath. Volume 2*, volume 388 of *London Math. Soc. Lecture Note Ser.*, pages 375–383. Cambridge Univ. Press, Cambridge, 2011.

[GL24] D. García-Lucas. The modular isomorphism problem and abelian direct factors. *Med. J. of Math.*, 21, article number 18, 2024

[GLM24] D. García-Lucas and L. Margolis On the modular isomorphism problem for groups of nilpotency class 2 with cyclic center *Forum Math.*, 2024

[GLdRS22] D. García-Lucas, Á. del Río, and M. Stanojkovski. On group invariants determined by modular group algebras: Even versus odd characteristic. *Alg. Rep. Th.*, pages 1–25, 2022. https://doi.org/10.1007/s10468-022-10182-x.

[GLdRS23a] D. García-Lucas, Á. del Río, and M. Stanojkovski. On the modular isomorphism problem for 2-generated groups with cyclic derived subgroup. arXiv:2310.02627, 2023.

[GLdRS23b] D. García-Lucas, Á. del Río, and M. Stanojkovski. On the modular isomorphism problem for groups of nilpotency class 2 with cyclic center. *Forum Mat.*, pages 1–19, 2023. to appear, arXiv:2306.06957.

[GLMdR22] D. García-Lucas, L. Margolis, and Á. del Río. Non-isomorphic 2-groups with isomorphic modular group algebras. *J. Reine Angew. Math.*, 783:269–274, 2022.

[Her07] M. Hertweck. A note on the modular group algebras of odd p-groups of M-length three. *Publ. Math. Debrecen*, 71(1-2):83–93, 2007.

[HS06] M. Hertweck and M. Soriano. On the modular isomorphism problem: groups of order 2^6. In *Groups, rings and algebras*, volume 420 of *Contemp. Math.*, pages 177–213. Amer. Math. Soc., Providence, RI, 2006.

[Jen41] S. A. Jennings. The structure of the group ring of a p-group over a modular field. *Trans. Amer. Math. Soc.*, 50:175–185, 1941.

[MM22] L. Margolis and T. Moede. The Modular Isomorphism Problem for small groups – revisiting Eick's algorithm. *Journal of Computational Algebra*, 1-2:100001, 2022.

[MS22] L. Margolis and M. Stanojkovski. On the modular isomorphism problem for groups of class 3 and obelisks. *J. Group Theory*, 25(1):163–206, 2022.

[MSS23] Leo Margolis, Taro Sakurai, and Mima Stanojkovski. Abelian invariants and a reduction theorem for the modular isomorphism problem. *J. Algebra*, 636:1–27, 2023.

[PS72] Inder Bir S. Passi and Sudarshan K. Sehgal. Isomorphism of modular group algebras. *Math. Z.*, 129:65–73, 1972.

[RS83] J. Ritter and S. K. Sehgal. Isomorphism of group rings. *Arch. Math. (Basel)*, 40(1):32–39, 1983.

[RS93] K. W. Roggenkamp and L. L. Scott. Automorphisms and nonabelian cohomology: an algorithm. *Linear Algebra Appl.*, 192:355–382, 1993.

[San85] R. Sandling. The isomorphism problem for group rings: a survey. In *Orders and their applications (Oberwolfach, 1984)*, pages 256–288. Springer, Berlin, 1985.

[San89] R. Sandling. The modular group algebra of a central-elementary-by-abelian p-group. *Arch. Math. (Basel)*, 52(1):22–27, 1989.

[Wur93] Martin Wursthorn. Isomorphisms of modular group algebras: an algorithm and its application to groups of order 2^6. *J. Symbolic Comput.*, 15(2):211–227, 1993.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".