IRREDSOL

Version 1.1

A library of irreducible solvable matrix groups over finite fields

Burkhard Höfling

Institut für Geometrie, Algebra und Diskrete Mathematik
Technische Universität
Braunschweig, Germany
b.hoefling@tu-bs.de

Contents

1 Overview

The package IRREDSOL provides a library of irreducible solvable subgroups of matrix groups over finite fields and a corresponding library of primitive solvable groups.

Currently, IRREDSOL contains all subgroups, up to conjugacy, of $GL(n,q)$, where n is a positive integer and q is a prime power satisfying $q^n < 2^{16}$. The underlying data base lists 28095 absolutely irreducible groups of degree > 1 and some additional information needed for constructing all irreducible groups. See Section 2.1 for details.

The groups in the IRREDSOL library can be accessed one at a time (see Section 2.2). In addition, there are functions which allow to search the library for groups with given properties (see Section 2.3). Moreover, given an irreducible solvable matrix group G, it is possible to identify the group in the library to which G is conjugate, including a conjugating matrix, if desired. See Section 3.1.

Apart from this, the IRREDSOL package provides additional functionality for matrix groups, such as the computation of imprimitivity systems; see Chapter 4.

It is well-known that there is a bijection between the irreducible solvable subgroups of $GL(n,p)$, where p is a prime, and the conjugacy classes, or equivalently the isomorphism types, of primitive solvable subgroups of $\mathrm{Sym}(p^n)$. The IRREDSOL package contains functions to translate between irreducible solvable matrix groups and primitive groups, to search for primitive solvable groups with given properties, and functions to recognize them, up to isomorphism (or, equivalently, up to conjugacy in $\mathrm{Sym}(p^n)$). See Sections 5.1, 5.3, and 5.4, respectively.

Note that GAP contains another library consisting of all 372 irreducible solvable subgroups of $GL(n,p)$, where $n > 1$, p is a prime, and $p^n < 2^8$. This library was originally created by Mark Short [Sho92], and two omissions in $GL(2,13)$ were added later; see Section 48.11 in the GAP reference manual. All of these groups are, of course, also part of the IRREDSOL data base, and the IRREDSOL package provides functions to identify the groups in the GAP library in IRREDSOL and viceversa. See Section 3.2.

The groups in the IRREDSOL data base were constructed using the methods described by Bettina Eick and the author in [EH03], where the construction of all irreducible solvable subgroups of $GL(n,q)$ with $q^n < 3^8$ is described.

For a historic account of the classification of irreducible matrix groups and primitive permutation groups, the reader is referred to [Sho92] and, for recent developments, to [EH03].

2 Accessing the data library

This chapter describes the design of the IRREDSOL group library (see Section 2.1) and the various ways of accessing groups in the data library. It is possible to construct individual groups in the group library (see Section 2.2), or to search for groups with certain properties (see Section 2.3). Finally, there are functions for loading and unloading group data manually (see Section 2.4).

2.1 Design of the group library

To avoid redundancy, the package IRREDSOL does not actually store lists of irreducible subgroups of $GL(n, q)$ but only has lists $\mathcal{A}_{n,q}$ of subgroups of $GL(n, q)$ such that

- each group in $\mathcal{A}_{n,q}$ is absolutely irreducible and solvable
- $\mathcal{A}_{n,q}$ contains a conjugate of each absolutely irreducible solvable subgroup of $GL(n, q)$
- no two groups in $\mathcal{A}_{n,q}$ are conjugate in $GL(n, q)$
- each group in $\mathcal{A}_{n,q}$ has trace field $\mathbb{F}_q$.

(For $n = 1$, such lists are not actually stored but are computed when required.)

We will briefly say that $\mathcal{A}_{n,q}$ contains, up to conjugacy, all absolutely irreducible solvable subgroups of $GL(n, q)$ with trace field $\mathbb{F}_q$. Here, the *trace field* of a subgroup G of $GL(n, q)$ is the field generated by the traces of the elements of G. By a theorem of Brauer, an irreducible subgroup of $GL(n, q)$ with trace field $\mathbb{F}_{q_0}$ has a conjugate lying in $GL(n, q_0)$. See also `TraceField` (4.4.1) and `ConjugatingMatTraceField` (4.4.2).

Note that by the Deuring-Noether theorem, two subgroups of $GL(n, q_0)$ are conjugate in $GL(n, q_0)$ if, and only if, they are conjugate in $GL(n, q)$. Therefore, we obtain, up to conjugacy, all absolutely irreducible solvable subgroups of $GL(n, q)$ by forming the union of the $\mathcal{A}_{n,q_0}$, where $\mathbb{F}_{q_0}$ runs over all subfields of $\mathbb{F}_q$.

The lists $\mathcal{A}_{n,q}$ are also sufficient to reconstruct lists of all irreducible solvable subgroups of $GL(n, q)$. Let d be a divisor of n, and let G be an absolutely irreducible subgroup of $GL(n/d, q^d)$. By regarding the underlying $\mathbb{F}_{q^d}$-vector space of G as an $\mathbb{F}_q$-vector space, we obtain an irreducible subgroup G^* of $GL(n, q)$ with splitting field $\mathbb{F}_{q^d}$. Up to conjugacy, all irreducible subgroups of $GL(n, q)$ arise in that way. If two subgroups G_1 and G_2 of $GL(n, q)$ are constructed from subgroups G_1^* and $G_2^*$of $GL(n/d, q^d)$, then G_1 and G_2 are conjugate if, and only if, there exists a Galois automophism σ of $\mathbb{F}_{q^d}/\mathbb{F}_q$ such that $(G_1^*)^\sigma$ and G_2^* are conjugate in $GL(n/d, q^d)$. See, e. g., [DH92], Theorem B 5.15. The latter information has been precomputed and is also part of the IRREDSOL library.

Note that all of the arguments above apply to nonsolvable groups as well.

2.2 Low level access functions

The access functions described in this section allow to check for the availability of data and to construct irreducible groups in the IRREDSOL group library.

1 ► `IsAvailableIrreducibleSolvableGroupData(`n`, `q`)` F

This function tests whether the irreducible solvable subgroups of $GL(n, q)$ with trace field $\mathbb{F}_q$ are part of the IRREDSOL library.

2 ► `IndicesIrreducibleSolvableMatrixGroups(`n`, `q`, `d`)` F

Let n and d be positive integers and q a prime power. This function returns a set of integers parametrising the groups in the IRREDSOL library which are subgroups of $GL(n, q)$ with trace field $\mathbb{F}_q$ and splitting field $\mathbb{F}_{q^d}$. This set is empty unless d divides n. An error is raised if the relevant data is not available, cf. 2.2.1.

```
gap> IndicesIrreducibleSolvableMatrixGroups (6, 2, 2);
[ 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12 ]
```

3 ► `IrreducibleSolvableMatrixGroup(`n`, `q`, `d`, `k`)` F

Let n, d and k be positive integers and q a prime power. This function returns the irreducible solvable subgroup of $GL(n, q)$ with trace field $\mathbb{F}_q$, splitting field $\mathbb{F}_{q^d}$, and index k. An error is raised if the relevant data is not available, or if k is not in `IndicesIrreducibleSolvableMatrixGroups(`n, q, d`)`; cf. 2.2.1 and 2.2.2. The groups returned have the attributes and properties described in Chapter 4 set to their appropriate values.

4 ► `IsAvailableAbsolutelyIrreducibleSolvableGroupData(`n`, `q`)` F

This function tests whether the absolutely irreducible solvable subgroups of $GL(n, q)$ with trace field $\mathbb{F}_q$ are in the IRREDSOL library.

5 ► `IndicesMaximalAbsolutelyIrreducibleSolvableMatrixGroups(`n`, `q`)` F

Let n be a positive integer and q a prime power. This function returns a set of integers parametrising those subgroups of $GL(n, q)$ in the IRREDSOL library that are maximal among the absolutely irreducible solvable subgroups with trace field $\mathbb{F}_q$ and that are maximal with respect to being solvable. An error is raised if the relevant data is not available (see 2.2.4 for information how to check this first). An integer k in the list return corresponds to `IrreducibleSolvableMatrixGroup(`n`, `q`, 1, `k`)`, which is the same group as `AbsolutelyIrreducibleSolvableMatrixGroup(`n`, `q`, `k`)`

```
gap> inds := IndicesMaximalAbsolutelyIrreducibleSolvableMatrixGroups (2,3);
[ 2 ]
gap> max := IrreducibleSolvableMatrixGroup (2,3,1,2);
Group([ [ [ Z(3), 0*Z(3) ], [ 0*Z(3), Z(3)^0 ] ], [ [ Z(3)^0, Z(3) ], [ 0*Z(3), Z(3)^0 ] ],
  [ [ Z(3), Z(3)^0 ], [ Z(3)^0, Z(3)^0 ] ], [ [ 0*Z(3), Z(3)^0 ], [ Z(3), 0*Z(3) ] ],
  [ [ Z(3), 0*Z(3) ], [ 0*Z(3), Z(3) ] ] ])
gap> max = GL(2,3); # it is the whole GL
true
```

6 ► `IndicesAbsolutelyIrreducibleSolvableMatrixGroups(`n`, `q`)` F
► `AbsolutelyIrreducibleSolvableMatrixGroup(`n`, `q`, `k`)` F

These functions are not available any longer. Please use `IndicesIrreducibleSolvableMatrixGroups(`n`, `q`, 1)` and `IrreducibleSolvableMatrixGroup(`n`, `q`, 1, `k`)` instead.

2.3 Finding matrix groups with given properties

This section describes three functions (`AllIrreducibleSolvableMatrixGroups`, `OneIrreducibleSolvableMatrixGroup`, `IteratorIrreducibleSolvableMatrixGroups`) which allow to find matrix groups with prescribed properties. Using these functions can be more efficient than to construct each group in the library using the functions in Section 2.2 because they can access additional information about a group in the IRREDSOL library before actually constructing the group. See the discussion following the description of `AllIrreducibleSolvableMatrixGroups` for details.

1 ▶ `AllIrreducibleSolvableMatrixGroups(`*func_1*`,` *arg_1*`,` *func_2*`,` *arg_2*`, ...)` F

This function returns a list of all irreducible solvable matrix groups G in the IRREDSOL library for which the return value of *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions which take a matrix group as their only argument and return a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. The functions `DegreeOfMatrixGroup` and `FieldOfMatrixGroup` (or their equivalents, see below) must be among the *func_i*. `TraceField`). Note that all groups in the data library have the property that `TraceField`(G) = `FieldOfMatrixGroup`(G); see Section 2.1 for details.

Note that there is also a function `IteratorIrreducibleSolvableMatrixGroups` (see 2.3.3) which allows to run through the list produced by `AllIrreducibleSolvableMatrixGroups` without having to store all of the groups simultaneously.

The following functions *func_i* are handled particularly efficiently, because the return values of these functions can be read off the IRREDSOL library without actually constructing the relevant matrix group. For the definitions of these functions, see Chapter 4.

- `DegreeOfMatrixGroup` (or `Degree`, `Dimension`, `DimensionOfMatrixGroup`),
- `CharacteristicOfField` (or `Characteristic`)
- `FieldOfMatrixGroup` (or `Field` or `TraceField`)
- `Order` (or `Size`)
- `IsMaximalAbsolutelyIrreducibleSolvableMatrixGroup`
- `IsAbsolutelyIrreducibleMatrixGroup` (or `IsAbsolutelyIrreducible`)
- `MinimalBlockDimensionOfMatrixGroup` (or `MinimalBlockDimension`)
- `IsPrimitiveMatrixGroup` (or `IsPrimitive`, `IsLinearlyPrimitive`)

The groups G passed to the *func_i* and the groups returned have the attributes and properties described in Chapter 4 set to their appropriate values. Note that you may speed up computations in G by using an isomorphic copy of G, which can be obtained via `RepresentationIsomorphism` (see 4.1.6).

```
# get just those groups with trace field GF(9)
gap> l := AllIrreducibleSolvableMatrixGroups (Degree, 1, Field, GF(9));;
gap> List (l, Order);
[ 4, 8 ]

# get all irreducible subgroups
gap> l := AllIrreducibleSolvableMatrixGroups (Degree, 1, Field, Subfields (GF(9)));;
gap> List (l, Order);
[ 1, 2, 4, 8 ]

# get only maximal absolutely irreducible ones
gap> l := AllIrreducibleSolvableMatrixGroups (Degree, 4, Field, GF(3),
>              IsMaximalAbsolutelyIrreducibleSolvableMatrixGroup, true);;
gap> SortedList (List (l, Order));
[ 320, 640, 2304, 4608 ]
```

```
# get only absolutely irreducible groups
gap> l := AllIrreducibleSolvableMatrixGroups (Degree, 4, Field, GF(3),
> IsAbsolutelyIrreducibleMatrixGroup, true);;
gap> Collected (List (l, Order));
[ [ 20, 1 ], [ 32, 7 ], [ 40, 2 ], [ 64, 10 ], [ 80, 2 ], [ 96, 6 ],
  [ 128, 9 ], [ 160, 3 ], [ 192, 9 ], [ 256, 6 ], [ 288, 1 ], [ 320, 2 ],
  [ 384, 4 ], [ 512, 1 ], [ 576, 3 ], [ 640, 1 ], [ 768, 1 ], [ 1152, 4 ],
  [ 2304, 3 ], [ 4608, 1 ] ]
```

2 ► `OneIrreducibleSolvableMatrixGroup(`*func_1*`,` *arg_1*`,` *func_2*`,` *arg_2*`, ...)` F

This function returns a matrix group G from the IRREDSOL library such that *func_i*(G) lies in *arg_i*, or `fail` if no such group exists. The arguments *func_1*, *func_2*, ..., must be GAP functions taking one argument and returning a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. The functions `DegreeOfMatrixGroup` and `FieldOfMatrixGroup` (or their equivalents, see below) must be among the *func_i*. Note that all groups in the data library have the property that `TraceField`(G) = `FieldOfMatrixGroup`(G); see Section 2.1 for details. The groups passed to the *func_i* and the groups returned have the attributes and properties described in Chapter 4 set to their appropriate values.

To use this function efficiently, please see the comments in 2.3.1.

3 ► `IteratorIrreducibleSolvableMatrixGroups(`*func_1*`,` *arg_1*`,` *func_2*`,` *arg_2*`, ...)` F

This function returns an iterator which runs through the list of all matrix groups G in the IRREDSOL library such that *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions taking one argument and returning a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. The functions `DegreeOfMatrixGroup` and `FieldOfMatrixGroup` (or their equivalents, see below) must be among the *func_i*.

Using

`IteratorIrreducibleSolvableMatrixGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...`))`

is functionally equivalent to

`Iterator(AllIrreducibleSolvableMatrixGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...`))`

(see Section 28.7 in the GAP reference manual for details) but, unlike `AllIrreducibleSolvableMatrixGroups`, does not store all relevant matrix groups at the same time. This may save a considerable amount of memory.

To use this function efficiently, please see the comments in 2.3.1.

2.4 Loading and unloading group data manually

The data required by the IRREDSOL library is loaded into GAP's workspace automatically whenever required, but is never unloaded automatically. The functions described in this and the following section describe how to load and unload this data manually. They are only relevant if timing or conservation of memory is an issue.

1 ► `LoadAbsolutelyIrreducibleSolvableGroupData(`*n*`,` *q*`)` F

This function loads the data for $GL(n, q)$ into the GAP workspace and does some pre-processing. If the data is already loaded, the function does nothing. This function is called automatically when you access the IRREDSOL library, so most users will not need this function.

2 ► `LoadedAbsolutelyIrreducibleSolvableGroupData()` F

This function returns a list. Each entry consists of an integer n and a set l. The set l contains all prime powers q such that the group data for $GL(n, q)$ is currently in memory.

3 ► `UnloadAbsolutelydIrreducibleSolvableGroupData([`n` [, `q`]])` F

This function can be used to delete data for $GL(n, q)$ from the GAP workspace. If no argument is given, all data will be deleted. If only n is given, all data for degree n (and any q) will be deleted. If n and q are given, only the data for $GL(n, q)$ will be deleted from the GAP workspace. Use this function if you run out of GAP workspace. The data is automatically re-loaded when required.

3 Recognition of matrix groups

This chapter describes some functions which, given an irreducible matrix group, identify a group in the IRREDSOL library which is conjugate to that group, see Section 3.1. Moreover, Section 3.2 describes how to translate between groups in the IRREDSOL library and the GAP library of irreducible solvable groups. Section 3.3 describes some functions which allow to load and unload the recognition data in the IRREDSOL package manually.

3.1 Identification of irreducible groups

1 ► `IsAvailableIdIrreducibleSolvableMatrixGroup(`G`)` F

This function returns `true` if `IdIrreducibleSolvableMatrixGroup` (see 3.1.3) will work for the irreducible matrix group G, and `false` otherwise.

2 ► `IsAvailableIdAbsolutelyIrreducibleSolvableMatrixGroup(`G`)` F

This function returns `true` if `IdIrreducibleSolvableMatrixGroup` (see 3.1.3) will work for the absolutely irreducible matrix group G, and `false` otherwise.

3 ► `IdIrreducibleSolvableMatrixGroup(`G`)` A

If the matrix group G is solvable and irreducible over $F =$ `FieldOfMatrixGroup`(G), (see 42.1.3 in the GAP reference manual), and a conjugate in $GL(n, F)$ of G belongs to the data base of irreducible solvable groups in IRREDSOL, this function returns a list `[`n`, `q`, `d`, `k`]` such that G is conjugate to `IrreducibleSolvableMatrixGroup(`n, q, d, k`)` (see 2.2.3).

```
gap> G := IrreducibleSolvableMatrixGroup (12, 2, 3, 52)^RandomInvertibleMat (12, GF(8));
<matrix group of size 2340 with 6 generators>
gap> IdIrreducibleSolvableMatrixGroup (G);
[ 12, 2, 3, 52 ]
```

4 ► `RecognitionIrreducibleSolvableMatrixGroup(`G`, `*wantmat*`, `*wantgroup*`)` F
► `RecognitionIrreducibleSolvableMatrixGroupNC(`G`, `*wantmat*`,`*wantgroup*`)` F

Let G be an irreducible solvable matrix group over a finite field, and let *wantmat* and *wantmat* be `true` or `false`. These functions identify a conjugate H of G group in the library. They return a record which has the following entries:

`id`
: contains the id of H (and thus of G); cf. `IdIrreducibleSolvableMatrixGroup` (3.1.3)

`mat` (present if `wantmat` is `true`)
: a matrix x such that $G^x = H$

`group` (present if `wantmat` is `true`)
: the group H

Note that in most cases, `RecognitionIrreducibleSolvableMatrixGroup` and `RecognitionIrreducibleSolvableMatrixGroupNC` are much slower if *wantmat* is set to true.

`RecognitionIrreducibleSolvableMatrixGroupNC` does not check its arguments. If the group G is beyond the scope of the IRREDSOL library (see 3.1.1), `RecognitionIrreducibleSolvableMatrixGroupNC` returns `fail`, while `RecognitionIrreducibleSolvableMatrixGroup` raises an error.

```
gap> G := IrreducibleSolvableMatrixGroup (6, 2, 3, 5) ^
>         RandomInvertibleMat (6, GF(4));
<matrix group of size 42 with 3 generators>
gap> r := RecognitionIrreducibleSolvableMatrixGroup (G, true, false);;
gap> r.id;
[ 6, 2, 3, 5 ]
gap> G^r.mat = CallFuncList (IrreducibleSolvableMatrixGroup, r.id);
true
```

5 ► `IdAbsolutelyIrreducibleSolvableMatrixGroup(`G`)` A
► `RecognitionAbsolutelyIrreducibleSolvableMatrixGroup(`G`,` *wantmat*`,` *wantgroup*`)` F
► `RecognitionAbsolutelyIrreducibleSolvableMatrixGroupNC(`G`,` *wantmat*`,`*wantgroup*`)` F

These functions are no longer available. These functions have been replaced by the functions `IdIrreducibleSolvableMatrixGroup` (3.1.3), `RecognitionIrreducibleSolvableMatrixGroup` (3.1.4), or `RecognitionIrreducibleSolvableMatrixGroupNC` (3.1.4).

Note that the ids returned by the functions for absolutely irreducible groups was a triple `[`n`,` d`,` k`]`, while the replacement functions use ids of the form `[`n`,` d`,` d`,` k`]`, where $d = 1$ in the absolutely irreducible case.

3.2 Compatibility with other data libraries

A library of irreducible solvable subgroups of $GL(n, p)$, where p is a prime and $p^n \leq 255$ already exists in GAP, see Section 48.11 in the GAP reference manual. The following functions allow one to translate between between that library and the IRREDSOL library.

1 ► `IdIrreducibleSolvableMatrixGroupIndexMS(`n`,` p`,` k`)` F

This function returns the id (see 3.1.3) of G, where G is `IrreducibleSolvableGroupMS(`n, p, k`)` (see 48.11.1 in the GAP reference manual).

```
gap> IdIrreducibleSolvableMatrixGroupIndexMS (6, 2, 5);
[ 6, 2, 2, 4 ]
gap> G := IrreducibleSolvableGroupMS (6,2,5);
<matrix group of size 27 with 2 generators>
gap> H := IrreducibleSolvableMatrixGroup (6, 2, 2, 4);
<matrix group of size 27 with 3 generators>
gap> G = H;
false # groups in the libraries need not be the same
gap> r := RecognitionIrreducibleSolvableMatrixGroup (G, true, false);;
gap> G^r.mat = H;
true
```

2 ► `IndexMSIdIrreducibleSolvableMatrixGroup(`n`,` q`,` d`,` k`)` F

This function returns a triple $[n, p, l]$ such that `IrreducibleSolvableGroupMS(`n, p, l`)` (see 48.11.1 in the GAP reference manual) is conjugate to `IrreducibleSolvableMatrixGroup(`n, q, d, k`)` (see 2.2.3).

```
gap> IndexMSIdIrreducibleSolvableMatrixGroup (6, 2, 2, 7);
[ 6, 2, 13 ]
gap> G := IrreducibleSolvableGroupMS (6,2,13);
<matrix group of size 27 with 2 generators>
gap> H := IrreducibleSolvableMatrixGroup (6, 2, 2, 7);
<matrix group of size 27 with 3 generators>
gap> G = H;
false # groups in the libraries need not be the same
gap> r := RecognitionIrreducibleSolvableMatrixGroup (G, true, false);;
gap> G^r.mat = H;
true
```

3.3 Loading and unloading recognition data manually

The data required by the IRREDSOL library is loaded into GAP's workspace automatically whenever required, but is never unloaded automatically. The functions described in this and the previous section describe how to load and unload this data manually. They are only relevant if timing or conservation of memory is an issue.

1 ► `LoadAbsolutelyIrreducibleSolvableGroupFingerprints(`n`,` q`)` FThis function loads the fingerprint data required for the recognition of absolutely irreducible solvable subgroups of $GL(n, q)$.

2 ► `LoadedAbsolutelyIrreducibleSolvableGroupFingerprints()` F

This function returns a list. Each entry consists of an integer n and a set l. The set l contains all prime powers q such that the recognition data for $GL(n, q)$ is currently in memory.

3 ► `UnloadAbsolutelyIrreducibleSolvableGroupFingerprints([n [,q]])` F

This function can be used to delete recognition data for irreducible groups from the GAP workspace. If no argument is given, all data will be deleted. If only n is given, all data for degree n (and any q) will be deleted. If n and q are given, only the data for $GL(n, q)$ will be deleted from the GAP workspace. Use this function if you run out of GAP workspace. The data is automatically re-loaded when required.

4 Additional functionality for matrix groups

This chapter explains some attributes, properties, and operations which may be useful for working with matrix groups. Some of these are part of the GAP library and are listed for the sake of completeness, and some are provided by the package IRREDSOL. Note that groups constructed by functions in IRREDSOL already have the appropriate properties and attributes.

4.1 Basic attributes for matrix groups

1 ► `DegreeOfMatrixGroup(`G`)` A
► `Degree(`G`)` O
► `DimensionOfMatrixGroup(`G`)` A
► `Dimension(`G`)` A

This is the degree of the matrix group or, equivalently, the dimension of the natural underlying vector space. See also 42.1.1 in the GAP reference manual.

2 ► `FieldOfMatrixGroup(`G`)` A

This is the field generated by the matrix entries of the elements of G. See also 42.1.3 in the GAP reference manual.

3 ► `DefaultFieldOfMatrixGroup(`G`)` A

This is a field containing all matrix entries of the elements of G. See also 42.1.2 in the GAP reference manual.

4 ► `SplittingField(`G`)` A

Let G be an irreducible subgroup of $GL(n, F)$, where $F =$ `FieldOfMatrixGroup(`G`)` is a finite field. This attribute stores the splitting field E for G, that is, the (unique) smallest field E containing F such that the natural EG-module E^n is the direct sum of absolutely irreducible EG- submodules. The number of these absolutely irreducible summands equals the dimension of E as an F-vector space.

5 ► `CharacteristicOfField(`G`)` A
► `Characteristic(`G`)` O

This is the characteristic of `FieldOfMatrixGroup(`G`)` (see 4.1.2).

6 ► `RepresentationIsomorphism(`G`)` A

This attribute stores an isomorphism $H \rightarrow G$, where H is a group in which computations can be carried out more efficiently than in G, and the isomorphism can be evaluated easily. Every group in the IRREDSOL library has such a representation isomorphism from a pc group H to G.

In this way, computations which only depend on the isomorphism type of G can be carried out in the group H and translated back to the group G via the representation isomorphism. Possible applications are the conjugacy classes of G, Sylow subgroups, composition and chief series, normal subgroups, group theoretical properties of G, and many more.

The concept of a representation isomorphism is related to nice monomorphisms; see Section 38.5 in the GAP reference manual. However, unlike nice monomorphisms, `RepresentationIsomorphism` need not be efficient for computing preimages (and, indeed, will not usually be, in the case of the groups in the IRREDSOL library).

4.2 Irreducibility and maximality of matrix groups

1 ▶ `IsIrreducibleMatrixGroup(`G`)` P
▶ `IsIrreducibleMatrixGroup(`G`, `F`)` O
▶ `IsIrreducible(`G` [, `F`])` O

The matrix group G of degree d is irreducible over the field F if no subspace of F^d is invariant under the action of G. If F is not specified, $F =$ `FieldOfMatrixGroup`(G) is assumed.

```
gap> G := IrreducibleSolvableMatrixGroup (4, 2, 2, 3);
<matrix group of size 10 with 2 generators>
gap> IsIrreducibleMatrixGroup (G);
true
gap> IsIrreducibleMatrixGroup (G, GF(2));
true
gap> IsIrreducibleMatrixGroup (G, GF(4));
false
```

2 ▶ `IsAbsolutelyIrreducibleMatrixGroup(`G`)` P
▶ `IsAbsolutelyIrreducible(`G`)` O

If present, this operation returns true if G is absolutely irreducible, i. e., irreducible over any extension field of `FieldOfMatrixGroup`(G).

```
gap> G := IrreducibleSolvableMatrixGroup (4, 2, 2, 3);
<matrix group of size 10 with 2 generators>
gap> IsAbsolutelyIrreducibleMatrixGroup(G);
false;
```

3 ▶ `IsMaximalAbsolutelyIrreducibleSolvableMatrixGroup(`G`)` P

This property, if present, is `true` if, and only if, G is absolutely irreducible and maximal among the solvable subgroups of $GL(d, F)$, where $d =$ `DegreeOfMatrixGroup`(G) and $F =$ `FieldOfMatrixGroup`(G).

4.3 Primitivity of matrix groups

1 ▶ `MinimalBlockDimensionOfMatrixGroup(`G`)` A
▶ `MinimalBlockDimensionOfMatrixGroup(`G`, `F`)` O
▶ `MinimalBlockDimension(`G` [, `F`])` O

Let G be a matrix group of degree d over the field F. A decomposition $V_1 \oplus \cdots \oplus V_k$ of F^d into F-subspaces V_i is a block system of G if the V_i are permuted by the natural action of G. Obviously, all V_i have the same dimension; this is the dimension of the block system $V_1 \oplus \cdots \oplus V_k$. The function `MinimalBlockDimensionOfMatrixGroup` returns the minimum of the dimensions of all block systems of G. If F is not specified, $F =$ `FieldOfMatrixGroup`(G) is assumed. At present, only methods for groups which are irreducible over F are available.

```
gap> G := AbsolutelyIrreducibleSolvableMatrixGroup (2,3,4);;
gap> MinimalBlockDimension (G, GF(3));
2
gap> MinimalBlockDimension (G, GF(9));
1
```

2 ► `IsPrimitiveMatrixGroup(`G`)` P
► `IsPrimitiveMatrixGroup(`G`, `F`)` O
► `IsPrimitive(`G` [, `F`])` O
► `IsLinearlyPrimitive(`G` [, `F`])` O

An irreducible matrix group G of degree d is primitive over the field F if it only has the trivial block system F^d or, equivalently, if `MinimalBlockDimensionOfMatrixGroup(`G, F`)` $= d$. If F is not specified, $F =$ `FieldOfMatrixGroup(`G`)` is assumed.

```
gap> G := IrreducibleSolvableMatrixGroup (2,2,1,1);
Group([ <an immutable 2x2 matrix over GF2>, <an immutable 2x2 matrix over GF2> ])
gap> IsPrimitiveMatrixGroup (G, GF(2));
true
gap> IsIrreducibleMatrixGroup (G, GF(4));
true
gap> IsPrimitiveMatrixGroup (G, GF(4));
false
```

3 ► `ImprimitivitySystems(`G` [, `F`])` O

This function returns the list of all imprimitivity systems of the irreducible matrix group G over the field F. If F is not given, `FieldOfMatrixGroup(`G`)` is used. Each imprimitivity system is given by a record with the following entries:

`bases`
: a list of the bases of the subspaces which form the imprimitivity system. Note that a basis here is just a list of vectors, not a basis in the sense of GAP (see 59.4.2 in the GAP reference manual). Each basis is in Hermite normal form so that the action of G on the imprimitivity system can be determined by `OnSubspacesByCanonicalBasis`

`stab1`
: the subgroup of G stabilizing the subspace W spanned by `bases[1]`

`min`
: is true if the imprimitivity system is minimal, that is, if `stab1` acts primitively on W, and false otherwise

```
gap> G := IrreducibleSolvableMatrixGroup (6, 2, 1, 9);
<matrix group of size 54 with 4 generators>
gap> impr := ImprimitivitySystems(G, GF(2));;
gap> List (ImprimitivitySystems(G, GF(2)), r -> Length (r.bases));
[ 3, 3, 1 ]
gap> List (ImprimitivitySystems(G, GF(4)),
>        r -> Action (G, r.bases, OnSubspacesByCanonicalBasis));
[ Group([ (), (1,2)(3,6)(4,5), (1,3,4)(2,5,6), (1,4,3)(2,6,5) ]),
  Group([ (1,2,4)(3,5,6), (1,3)(2,5)(4,6), (), () ]),
  Group([ (1,2,4)(3,5,6), (1,3)(2,5)(4,6), (1,2,4)(3,6,5), (1,4,2)(3,5,6) ]),
  Group([ (1,2,4)(3,5,6), (1,3)(2,5)(4,6), (1,4,2)(3,5,6), (1,2,4)(3,6,5) ]),
  Group([ (), (1,2), (), () ]), Group([ (1,2,3), (), (), () ]),
  Group([ (), (2,3), (1,2,3), (1,3,2) ]),
  Group([ (), (2,3), (1,2,3), (1,3,2) ]),
  Group([ (), (2,3), (1,2,3), (1,3,2) ]), Group(()) ]
```

4.4 Conjugating matrix groups into smaller fields

1 ► `TraceField(`G`)` A

This is the field generated by the traces of the elements of the matrix group G. If G is an irreducible matrix group over a finite field then, by a theorem of Brauer, G has a conjugate which is a matrix group over `TraceField(`G`)`.

```
gap> repeat
>        G := IrreducibleSolvableMatrixGroup (8, 2, 2, 7)^RandomInvertibleMat (8, GF(8));
>    until FieldOfMatrixGroup (G) = GF(8);
gap> TraceField (G);
GF(2)
```

2 ► `ConjugatingMatTraceField(`G`)` A

If bound, this is a matrix x over `FieldOfMatrixGroup(`G`)` such that G^x is a matrix group over `Trace-Field(`G`)`. Currently, there are only methods available for irreducible matrix groups G over finite fields and certain trivial cases. The method for absolutely irreducible groups is described in [GH97]. Note that, for matrix groups over infinite fields, such a matrix x need not exist.

```
gap> repeat
>       G := IrreducibleSolvableMatrixGroup (8, 2, 2, 7) ^
>                RandomInvertibleMat (8, GF(8));
>    until FieldOfMatrixGroup(G) = GF(8);
gap> FieldOfMatrixGroup (G^ConjugatingMatTraceField (G));
GF(2)
```

5 Primitive solvable groups

A finite group G is called *primitive* if it has a maximal subgroup M with trivial core; the group acts faithfully and primitively on the cosets of such a maximal subgroup.

Now assume that G is primitive and solvable. Then there exists a unique conjugacy class of such maximal subgroups; the index of M in G is called the degree of G. Moreover, M complements the socle N of G. THe socle N coincides with the Fitting subgroup of G; it is the unique minimal normal subgroup N of G. Therefore, the index of M in G is a prime power, p^n, say. Regarding N as a $\mathbb{F}_p$-vector space, M acts as an irreducible subgroup of $GL(n,p)$ on N. Conversely, if M is an irreducible solvable subgroup of $GL(n,p)$, and $V = \mathbb{F}_p^n$, then the split extension of V by M is a primitive solvable group. This establishes a well known bijection between the isomorphism types (or, equivalently, the $Sym(p^n)$-conjugacy classes) of primitive solvable groups of degree p^n and the conjugacy classes of irreducible solvable subgroups of $GL(n,p)$.

The IRREDSOL package provides functions for translating between primitive solvable groups and irreducible solvable matrix groups, which are described in Section 5.1. Moreover, there are functions for finding primitive solvable groups with given properties, see Section 5.2 and 5.3.

5.1 Translating between irreducible solvable matrix groups and primitive solvable groups

1 ► `PrimitivePcGroupIrreducibleMatrixGroup(`G`)` F
► `PrimitivePcGroupIrreducibleMatrixGroupNC(`G`)` F

For a given irreducible solvable matrix group G over a prime field, this function returns a primitive pc group H which is the split extension of G with its natural underlying vector space V. The `NC` version does not check whether G is over a prime field, or whether G is irreducible. The group H has an attribute `Socle` (see 37.12.10 in the GAP reference manual, corresponding to V. If the package CRISP is loaded, then the attribute `SocleComplement` (see 4.3.2 in the CRISP manual) is set to a subgroup of H isomorphic with G.

```
gap> PrimitivePcGroupIrreducibleMatrixGroup (
>       IrreducibleSolvableMatrixGroup (4,2,2,3));
<pc group of size 160 with 6 generators>
```

2 ► `PrimitivePermutationGroupIrreducibleMatrixGroup(`G`)` F
► `PrimitivePermutationGroupIrreducibleMatrixGroupNC(`G`)` F

For a given irreducible solvable matrix group G over a prime field, this function returns a primitive permutation group H, representing the affine action of G on its natural vector space V. The `NC` version does not check whether G is over a prime field, or whether G is irreducible. The group H has an attribute `Socle` (see 37.12.10 in the GAP reference manual, corresponding to V. If the package CRISP is loaded, then the attribute `SocleComplement` (see 4.3.2 in the CRISP manual) is set to a subgroup of H isomorphic with G.

```
gap> PrimitivePermutationGroupIrreducibleMatrixGroup (
>       IrreducibleSolvableMatrixGroup (4,2,2,3));
<permutation group of size 160 with 6 generators>
```

3 ▶ `IrreducibleMatrixGroupPrimitiveSolvableGroup(`*G*`)` F
▶ `IrreducibleMatrixGroupPrimitiveSolvableGroupNC(`*G*`)` F

For a given primitive solvable group G, this function returns a matrix group obtained from the conjugation action of G on its unique minimal normal subgroup N, regarded as a vector space over $\mathbb{F}_p$, where p is the exponent of N. The $\mathbb{F}_p$-basis of N is chosen arbitrarily, so that the matrix group returned is unique only up to conjugacy in the relevant $GL(n, p)$. The NC version does not check whether G is primitive and solvable.

```
gap> IrreducibleMatrixGroupPrimitiveSolvableGroup (SymmetricGroup (4));
Group([ <an immutable 2x2 matrix over GF2>, <an immutable 2x2 matrix over GF2>
    , <an immutable 2x2 matrix over GF2>, <an immutable 2x2 matrix over GF2>
 ])
```

5.2 Finding primitive pc groups with given properties

1 ▶ `AllPrimitivePcGroups(`*func_1* `,` *arg_1* `,` *func_2* `,` *arg_2* `, ...)` F

This function returns a list of all primitive solvable pc groups G in the IRREDSOL library for which the return value of *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions which take a pc group as their only argument and return a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its equivalents, see below.

The following functions *func_i* are handled particularly efficiently.

- `Degree`, `NrMovedPoints`, `LargestMovedPoint`
- `Order`, `Size`

Note that there is also a function `IteratorPrimitivePcGroups` (see 5.2.3) which allows one to run through the list produced by `AllPrimitivePcGroups` without having to store all the groups in the list simultaneously.

```
gap> AllPrimitivePcGroups (Degree, [1..255], Order, [168]);
[ <pc group of size 168 with 5 generators> ]
```

2 ▶ `OnePrimitivePcGroup(`*func_1* `,` *arg_1* `,` *func_2* `,` *arg_2* `, ...)` F

This function returns one primitive solvable pc group G in the IRREDSOL library for which the return value of *func_i*(G) lies in *arg_i*, or `fail` if no such group exists. The arguments *func_1*, *func_2*, ..., must be GAP functions which take a pc group as their only argument and return a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its, equivalents, `NrMovedPoints` or `LargestMovedPoint`.

For a list of functions which are handled particularly efficiently, see 5.2.1.

```
gap> OnePrimitivePcGroup (Degree, [256], Order, [256*255]);
<pc group of size 65280 with 11 generators>
```

3 ▶ `IteratorPrimitivePcGroups(`*func_1* `,` *arg_1* `,` *func_2* `,` *arg_2* `, ...)` F

This function returns an iterator which runs through the list of all primitive solvable pc groups G in the IRREDSOL library such that *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions taking a pc group as their only argument and returning a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its, equivalents, `NrMovedPoints` or `LargestMovedPoint`. For a list of functions which are handled particularly efficiently, see 5.2.1.

Using

`IteratorPrimitivePcGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...))

is functionally equivalent to

`Iterator(AllPrimitivePcGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...))

(see 28.7 in the GAP reference manual for details) but does not compute all relevant pc groups at the same time. This may save some memory.

5.3 Finding primitive solvable permutation groups with given properties

1 ► `AllPrimitiveSolvablePermutationGroups(`*func_1* , *arg_1* , *func_2* , *arg_2* , ...`)` F

This function returns a list of all primitive solvable permutation groups G corresponding to irreducible matrix groups in the IRREDSOL library for which the return value of *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions which take a permutation group as their only argument and return a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its equivalents, see below.

The following functions *func_i* are handled particularly efficiently.

- `Degree`, `NrMovedPoints`, `LargestMovedPoint`
- `Order`, `Size`

Note that there is also a function `IteratorPrimitivePermutationGroups` (see 5.3.3) which allows one to run through the list produced by `AllPrimitivePcGroups` without having to store all of the groups simultaneously.

```
gap> AllPrimitiveSolvablePermutationGroups (Degree, [1..100], Order, [72]);
[ Group([ (1,4,7)(2,5,8)(3,6,9), (1,2,3)(4,5,6)(7,8,9), (2,4)(3,7)(6,8),
      (2,3)(5,6)(8,9), (4,7)(5,8)(6,9) ]),
  Group([ (1,4,7)(2,5,8)(3,6,9), (1,2,3)(4,5,6)(7,8,9), (2,5,3,9)(4,8,7,6),
      (2,7,3,4)(5,8,9,6), (2,3)(4,7)(5,9)(6,8) ]),
  Group([ (1,4,7)(2,5,8)(3,6,9), (1,2,3)(4,5,6)(7,8,9), (2,5,6,7,3,9,8,4) ]) ]
gap> List (last, IdGroup);
[ [ 72, 40 ], [ 72, 41 ], [ 72, 39 ] ]
```

2 ► `OnePrimitiveSolvablePermutationGroup(`*func_1* , *arg_1* , *func_2* , *arg_2* , ...`)` F

This function returns one primitive solvable permutation group G corresponding to irreducible matrix groups in the IRREDSOL library for which the return value of *func_i*(G) lies in *arg_i*, or `fail` if no such group exists. The arguments *func_1*, *func_2*, ..., must be GAP functions which take a permutation group as their only argument and return a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its, equivalents, `NrMovedPoints` or `LargestMovedPoint`.

For a list of functions which are handled particularly efficiently, see 5.3.1.

```
gap> OnePrimitiveSolvablePermutationGroup (Degree, [1..100], Size, [123321]);
fail
```

3 ► `IteratorPrimitivePermutationGroups(`*func_1* `,` *arg_1* `,` *func_2* `,` *arg_2* `,` ... `)` F

This function returns an iterator which runs through the list of all primitive solvable groups G in the IRREDSOL library such that *func_i*(G) lies in *arg_i*. The arguments *func_1*, *func_2*, ..., must be GAP functions taking a pc group as their only argument and returning a value, and *arg_1*, *arg_2*, ..., must be lists. If *arg_i* is not a list, *arg_i* is replaced by the list `[`*arg_i*`]`. One of the functions must be `Degree` or one of its, equivalents, `NrMovedPoints` or `LargestMovedPoint`. For a list of functions which are handled particularly efficiently, see 5.3.1.

Using

`IteratorPrimitiveSolvablePermutationGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...))

is functionally equivalent to

`Iterator(AllPrimitiveSolvablePermutationGroups(`*func_1*, *arg_1*, *func_2*, *arg_2*, ...))

(see 28.7 in the GAP reference manual for details) but does not compute all relevant pc groups at the same time. This may save some memory.

5.4 Recognizing primitive solvable groups

1 ► `IdPrimitiveSolvableGroup(`*G*`)` F
► `IdPrimitiveSolvableGroupNC(`*G*`)` F

returns the id of the primitive solvable group G. This is the same as the id of `IrreducibleMatrixGroup-PrimitiveSolvableGroup(`*G*`)`, see 5.1.3 and 3.1.3. Note that two primitive solvable groups are isomorphic if, and only if, their ids returned by `IdPrimitivePcGroup` are the same. The NC version does not check whether G is primitive and solvable.

```
gap> G := PrimitivePcGroupIrreducibleMatrixGroup (\
>              IrreducibleSolvableMatrixGroup (6,2,3,3));
<pc group of size 8064 with 10 generators>
gap> IdPrimitiveSolvableGroup (G);
[ 6, 2, 3, 3 ]
```

Bibliography

[DH92] K. Doerk and T. Hawkes. *Finite soluble groups.* DeGruyter, Berlin, New York, 1992.

[EH03] Bettina Eick and Burkhard Höfling. The solvable primitive permutation groups of degree at most 6560. *LMS J. Comput. Math.*, 6:29–39, 2003.

[GH97] S. P. Glasby and R. B. Howlett. Writing representations over minimal fields. *Comm. Alg.*, 25:1703–1711, 1997.

[Sho92] Mark W. Short. *The Primitive Soluble Permutation Groups of Degree less than 256*, volume 1519 of *Lecture Notes in Math.* Springer, 1992.

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".