# matgrp

**A GAP Package**

**by**

**Alexander Hulpke**

**Department of Mathematics**
**Colorado State University**

# Contents

# 1 matgrp

This manual describes the matgrp package. This package for GAP provides an interface for Solvable Radical (or Fitting-free) algorithms for matrix groups, using a recognition tree. The underlying rings supported are finite fields and residue class rings of the integers modulo a (possibly composite) number.

The package also implements routines that make this data structure work, notably routines for the epimorphism on the radical factor, data structures for subgroups, as well as for polyclic generating sets for solvable matrix (sub)groups.

The code is still experimental, but should be as stable as the underlying packages.



If you were able to install the prerequisite package recog, you will have no problems in installing matgrp, which is plain GAP code.

## 1.1 Usage

There is basically just one operation of interest. The package implements a method for `FittingFreeLiftSetup` that applies to matrix groups over finite fields, or over residue class rings of the integers.

```
gap> gens:=[ [ [ -28, 3, 9 ], [ 0, -1, -6 ], [ 3, 0, 1 ] ],
>   [ [ -1, 0, 0 ], [ -9, 1, 3 ], [ -3, 0, -1 ] ],
>   [ [ 0, 0, 1 ], [ 1, 0, 9 ], [ 0, 1, 0 ] ] ];
gap> g:=Group(List(gens,x->x*One(Integers mod 89)));
<matrix group with 3 generators>
gap> ffs:=FittingFreeLiftSetup(g);;
gap> Size(Image(ffs.factorhom));
704880
gap> RelativeOrders(ffs.pcgs);
[ 11, 2, 2, 2, 89, 89 ]
gap> h:=Group(List(gens,x->x*One(Integers mod 2403)));
<matrix group with 3 generators>
gap> ffs:=FittingFreeLiftSetup(h);;
gap> Size(Image(ffs.factorhom));
704880
gap> RelativeOrders(ffs.pcgs);
[ 3, 2, 2, 11, 2, 2, 2, 89, 89, 3, 3, 3, 3, 3, 3, 3, 3, 3 ]
```

Once this data structure has been computed, library functionality that uses it (that is newer versions of Solvable Radical/Fitting free algorithms) become available. (Note that method selection does not yet catch all these cases and that in particular `FittingFreeLiftSetup` needs to be called explicitly at the start to avoid falling back on generic library methods.)

```
gap> cl:=ConjugacyClasses(g);;
gap> Collected(List(cl,Size));
[ [ 1, 1 ], [ 7920, 1 ], [ 7921, 87 ], [ 704880, 1 ], [ 712890, 87 ],
  [ 62029440, 1 ], [ 62037272, 3916 ], [ 62734320, 174 ], [ 63447210, 3741 ] ]
gap> RepresentativeAction(g,g.1,g.1^g.2);
[ [ Z(89)^21, Z(89)^60, 0*Z(89) ], [ Z(89)^22, Z(89)^54, Z(89)^80 ],
  [ Z(89)^86, Z(89)^34, Z(89)^35 ] ]
gap> g.1^last=g.1^g.2;
true
gap> m:=MaximalSubgroupClassReps(h);
[ <matrix group of size 38683802512903680 with 20 generators>,
  <matrix group of size 10550127958064640 with 20 generators>,
  <matrix group of size 29012851884677760 with 19 generators>,
  <matrix group of size 14651105610240 with 19 generators>,
  <matrix group of size 4298200279211520 with 18 generators>,
  <matrix group of size 58025703769355520 with 20 generators>,
  <matrix group of size 1289460083763456 with 20 generators>,
  <matrix group of size 29635190893440 with 20 generators>,
  <matrix group of size 28976631095808 with 20 generators> ]
gap> s:=HallViaRadical(g,[2])[1];
<matrix group of size 128 with 7 generators>
gap> n:=NormalizerViaRadical(g,s);
<matrix group of size 1408 with 7 generators>
```

## 1.2 Matrices over Residue class rings

To improve performance, the package provides a rudinemntary data type for matrices over residue class rings (which in the long run ought to be replaced by a better implementation in the main system), that yields a somewhat faster multiplication than using lists of lists.

The command `ZmodnzMat(ring,intmat)` will create a matrix in this compact format, for best performance convert matrices into this format before creating a group.

```
gap> r:=Integers mod 2403;
(Integers mod 2403)
gap> gens:=List(gens,x->ZmodnZMat(r,x));
[ [ [ 2375, 3, 9 ], [ 0, 2402, 2397 ], [ 3, 0, 1 ] ]*ZmodnZObj(1,2403),
  [ [ 2402, 0, 0 ], [ 2394, 1, 3 ], [ 2400, 0, 2402 ] ]*ZmodnZObj(1,2403),
  [ [ 0, 0, 1 ], [ 1, 0, 9 ], [ 0, 1, 0 ] ]*ZmodnZObj(1,2403) ]
gap> h:=Group(gens);
<matrix group with 3 generators>
gap> ffs:=FittingFreeLiftSetup(h);;
```

## 1.3 Generic Treatment of Matrix groups

The ultimate goal of this package surely must be to have GAP work with matrix groups in the same way as with permutation groups, simply using recognition as underlying structure. This is not yet feasible at the moment and will require some larger changes to the system:

-

Currently the `recog` package does not provide a verification routine that would guarantee a correct recognition tree.

- There needs to be a way to override the default treatment of matrix groups using nice monomorphisms, if recognition is available.

- In matrix groups, an initial (safety) test for element membership is expensive. If the element is not contained in the group, it even might require a more robust behaviour of action homomorphisms. In practice such a test ought to be trivial, as the element is known to be contained in the group from creation. However matrices in GAP do not carry "family" matrix group with them to make such a membership test easy.

## 1.4 Error Messages

In certain situations the current version of the code may produce (one-off) error messages. Some reasons for this are:

- The recog package does not yet contain a verify routine.

- Heuristics for numbers of random elements are not yet optimal, and fallback code to treat repeated failure, due to unlucky selection of random elements, is still missing.

- There might be errors left in the required packages used for matrix group recognition.

In general, if a calculation in a `matgrp` routine produces an error, it can be worth trying the command again, more often than not it will run fine. If errors arise repeatedly, please inform the package author.

Some error messages issued by the package are:

`Error, Wrong size -- set doall option`
: The selection of random radical elements was not enough to generate the radical. Set the `doall` option to `true` to enforce use of a proven generating set.

`no vector found`
: When trying a permutation representation for a constituent of the radical factor, action on vectors found no suitable candidate.

`some discrepancy happened`
: Recognized group order and computed group order disagree. Likely due to an error in constructive recognition.

`layerlen`
: Computed and acltual size of a solvable layer disagree. Likely due to an error in the randomized stabilizer chain computation.

# Bibliography

# 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".