Polenta

Polycyclic presentations for matrix groups

A GAP4 Package

by

Björn Assmann

Institut für Geometrie

TU Braunschweig

Pockelsstr. 14
D-38106 Braunschweig

email: bjoerna77@gmx.de

November 2003

Contents

1 Introduction

1.1 The package

This package provides functions for the computation with matrix groups. Let G be a subgroup of $GL(d, R)$ where the ring R is either equal to $\mathbb{Q}, \mathbb{Z}$ or a finite field $\mathbb{F}_q$. Then:

- We can test whether G is solvable.
- If G is polycyclic, then we can determine a polycyclic presentation for G.

A group G which is given by a polycyclic presentation can be largely investigated by algorithms implemented in the GAP-package Polycyclic [EN00]. For example we can determine if G is torsion-free and calculate the torsion subgroup. Further we can compute the derived series and the Hirsch length of the group G. Also various methods for computations with subgroups, factor groups and extensions are available.

In the case that the matrix group G is a subgroup of $GL(d, \mathbb{F}_q)$ or $GL(d, \mathbb{Z})$, the group G is solvable if and only if G is polycyclic (see Chapter 1 in [Seg83]). Therefore, in this case, we can test if G is polycyclic.

As a by-product, the Polenta package provides some functionality to compute certain module series for modules of solvable groups. For example, if G is a rational polycyclic matrix group, then we can compute the radical series of the natural $\mathbb{Q}[G]$-module $\mathbb{Q}^d$.

1.2 Polycyclic groups

A group G is called polycyclic if it has a finite subnormal series with cyclic factors. It is a well-known fact that every polycyclic group is finitely presented by a so-called polycyclic presentation (see for example Chapter 9 in [Sim94] or Chapter 2 in [EN00]). In GAP groups which are defined by polycyclic presentations are called polycyclically presented groups, short PcpGroups.

The overall idea of the algorithm implemented in this package have first been introduced by Ostheimer in 1996 [Ost96]. In 2001 Eick presented a more detailed version [Eic01]. This package contains an implementation of this algorithm. A description of this implementation together with some refinements and extensions can be found in [Ass03].

2 Methods for matrix groups

2.1 Polycyclic presentations of matrix groups

Groups defined by polycyclic presentations are called PcpGroups in GAP. We refer to the Polycyclic manual [EN00] for further background.

Suppose that a collection of matrices of $GL(d, R)$ is given, where the ring R is either $\mathbb{Q}, \mathbb{Z}$ or a finite field. Let G be the group which is generated by these matrices. If the group G is polycyclic, then the following functions determine a PcpGroup isomorphic to G.

1 ▶ `PcpGroupByMatGroup(` *G* `)`
▶ `IsomorphismPcpGroup(` *G* `)`

G is a subgroup of $GL(d, R)$ where $R = \mathbb{Q}, \mathbb{Z}$ or $\mathbb{F}_q$. If G is polycyclic, then these functions determine a PcpGroup isomorphic to G and an isomorphism onto this group. If G is not polycyclic, then there are two cases: If $R = \mathbb{Z}$ or $\mathbb{F}_q$, then the algorithm returns 'fail'. In case that $R = \mathbb{Q}$ the algorithm may return 'fail' or may not terminate.

Note, that the method `IsomorphismPcpGroup`, installed in this package, cannot be applied directly to a group given by the function `AlmostCrystallographicGroup`. Please use `POL_AlmostCrystallographicGroup` (with the same parameters as `AlmostCrystallographicGroup`) instead.

2 ▶ `Image(` *map* `)`
▶ `ImageElm(` *map*`,` *elm* `)`
▶ `ImagesSet(` *map*`,` *elms* `)`
▶ `PreImagesRepresentative(` *map*`,` *pcpelm* `)`

Here *map* is an isomorphism from a polycyclic matrix group G onto a PcpGroup H calculated by `IsomorphismPcpGroup(`G`)`. These functions can be used to compute with such an isomorphism. If the input *elm* is an element of G, then the function `ImageElm` can be used to compute the image of *elm* under *map*. If *elm* is not contained in G then the function `ImageElm` returns 'fail'. The input *pcpelm* is an element of H.

3 ▶ `IsSolvableGroup(` *G* `)`
▶ `IsSolvableMatGroup(` *G* `)`

G is a subgroup of $GL(d, R)$ where $R = \mathbb{Q}, \mathbb{Z}$ or $\mathbb{F}_q$. This function tests if G is solvable and returns 'true' or 'false'.

4 ▶ `IsPolycyclicMatGroup(` *G* `)`

G is a subgroup of $GL(d, R)$ where $R = \mathbb{Q}, \mathbb{Z}$ or $\mathbb{F}_q$. This function tries to test if G is polycyclic. If G is polycyclic, then it returns 'true'. If G is not polycyclic, then there are two cases: If $R = \mathbb{Z}$ or $\mathbb{F}_q$, then the function returns 'false'. In case that $R = \mathbb{Q}$ the algorithm may return 'false' or may not terminate.

2.2 Module series

Let G be a finitely generated solvable subgroup of $GL(d, \mathbb{Q})$. The vectors space $\mathbb{Q}^d$ is a module for the algebra $\mathbb{Q}[G]$. The following functions provide the possiblity to compute certain module series of $\mathbb{Q}^d$. Recall that the radical $Rad_G(\mathbb{Q}^d)$ is definied to be the intersection of maximal $\mathbb{Q}[G]$-submodules of $\mathbb{Q}^d$. Further the radical series

$$0 = R_n < R_{n-1} < \ldots < R_1 < R_0 = \mathbb{Q}^d$$

is definied by $R_{i+1} := Rad_G(R_i)$.

1 ▶ `RadicalSeriesSolvableMatGroup( G )`

return the radical series for the solvable rational matrix group G.

A module is said to homogeneous if it is the direct sum of irreducible and isomorphic submodules. A radical series of $\mathbb{Q}^d$ can be refinied to a homogeneous series. That is a submodule series such that the factors are homogeneous.

2 ▶ `HomogeneousSeriesAbelianMatGroup( G )`

returns the homogeneous series for the abelian rational matrix group G.

Further a homogeneous series can refined to a composition series. That is a submodule series such that the factors are irreducible.

3 ▶ `CompositionSeriesAbelianMatGroup( G )`

returns the composition series for the abelian rational matrix group G.

2.3 Examples

1 ▶ `PolExamples( l )`

Returns some examples for polycyclic rational matrix groups, where l is an integer between 1 and 24. These can be used to test the functions in this package.

3 An example application

In this section we outline two example computations with the functions of the previous chapter.

3.1 Presentation for rational matrix groups

```
gap> G := MatExamples(2);
<matrix group with 6 generators>

gap> mats := GeneratorsOfGroup( G );
[ [ [ 2, -1, -1, 3 ], [ -1, 0, 1, -4 ], [ 0, 1, 2, -2 ], [ -1, 1, 2, -4 ] ],
  [ [ 1, 1, 1, -2 ], [ 1, 1, -1, 1 ], [ -2, 1, 0, -3 ], [ -1, 1, 0, -2 ] ],
  [ [ 1, 0, -1, 0 ], [ -2, 5, 5, -11 ], [ -1, 0, 2, 0 ], [ -1, 1, 2, -2 ] ],
  [ [ 0, 3, 2, -7 ], [ -1, 6, 4, -13 ], [ -1, 2, 1, -5 ], [ -1, 3, 2, -7 ] ],
  [ [ 0, 4, 5, -12 ], [ 2, -4, 1, 5 ], [ 1, -4, -1, 8 ], [ 1, -3, 0, 5 ] ],
  [ [ 1, 0, -1, 0 ], [ -2, 5, 5, -11 ], [ -1, 0, 2, 0 ], [ -1, 1, 2, -2 ] ] ]

# calculate an isomorphism from G to a pcp-group
gap> nat := IsomorphismPcpGroup( G );;

gap> H := Image( nat );
Pcp-group with orders [ 2, 2, 3, 3, 4, 0, 0, 0 ]

gap> h := GeneratorsOfGroup( H );
[ g1, g2, g3, g4, g5, g6, g7, g8 ]

gap> mats2 := List( h, x -> PreImage( nat, x ) );;

# take a random element of G
gap> exp :=  [ 1, 1, 1, 1, 0, 0, 0, 0 ];;
gap> g := MappedVector( exp, mats2 );
[ [ 229793843, -345584045, -503782245, 823202280 ],
  [ 397912065, -598518263, -872506665, 1425593295 ],
  [ 141954212, -213549855, -311309508, 508615375 ],
  [ 189806315, -285510521, -416211500, 680033928 ] ]

# map g into the image of nat
gap> i := ImageElm( nat, g );
g1*g2*g3*g4

# exponent vector
gap> Exponents( i );
[ 1, 1, 1, 1, 0, 0, 0, 0 ]
```

```
# compare the preimage with g
gap> PreImagesRepresentative( nat, i );
[ [ 229793843, -345584045, -503782245, 823202280 ],
  [ 397912065, -598518263, -872506665, 1425593295 ],
  [ 141954212, -213549855, -311309508, 508615375 ],
  [ 189806315, -285510521, -416211500, 680033928 ] ]
gap> last = g;
true
```

3.2 Modules series

```
gap> gens :=
[ [ [ 1746/1405, 524/7025, 418/1405, -77/2810 ],
    [ 815/843, 899/843, -1675/843, 415/281 ],
    [ -3358/4215, -3512/21075, 4631/4215, -629/1405 ],
    [ 258/1405, 792/7025, 1404/1405, 832/1405 ] ],
  [ [ -2389/2810, 3664/21075, 8942/4215, -35851/16860 ],
    [ 395/281, 2498/2529, -5105/5058, 3260/2529 ],
    [ 3539/2810, -13832/63225, -12001/12645, 87053/50580 ],
    [ 5359/1405, -3128/21075, -13984/4215, 40561/8430 ] ] ];

gap> H := Group( gens );
<matrix group with 2 generators>

gap> RadicalSeriesSolvableMatGroup( H );
[ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 0, 0, 1, -197/414 ], [ 1, 0, -3, 2 ], [ 0, 1, 55/4, -55/8 ] ],
  [ [ 0, 1, 55/4, -55/8 ], [ 1, 4/15, 2/3, 1/6 ] ],
  [ [ 1, 4/15, 2/3, 1/6 ] ],
  [  ] ]
```

4 Installing and loading the Polenta package

4.1 Installing the Polenta package

The installation of the Polenta package follows standard GAP rules. So the standard method is to unpack the package into the `pkg` directory of your GAP distribution. This will create a `polenta` subdirectory.

For other non-standard options please see Chapter 74.1 in the GAP Reference Manual.

Note that the GAP-Packages Alnuth and Polycyclic are needed for this package. They can be obtained at

`http://cayley.math.nat.tu-bs.de/software/content.html`

4.2 Loading the Polenta package

If the Polenta Package is not already loaded then you have to request it explicitly. This can be done by `LoadPackage("polenta")`. The `LoadPackage` command is described in Section 74.2.1 in the GAP Reference Manual.

Bibliography

[Ass03] Bjoern Assmann. Polycyclic presentations for matrix groups. Diplomarbeit, TU Braunschweig, 2003. Forthcoming.

[Eic01] Bettina Eick. Algorithms for polycyclic groups. Habilitationsschrift, Gesamthochschule Kassel, 2001.

[EN00] Bettina Eick and Werner Nickel. *Polycyclic*, 2000. A GAP 4 package, see [GAP].

[Ost96] Gretchen Ostheimer. *Algorithms for Polycyclic-by-finite groups*. PhD thesis, Rutgers University, 1996.

[Seg83] Daniel Segal. *Polycyclic groups*. Cambridge University Press, 1983.

[Sim94] Charles C. Sims. *Computation with finitely presented groups*. Cambridge University Press, 1994.

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".