Polenta

Polycyclic presentations for matrix groups

1.3.10

29 March 2022

Björn Assmann

Copyright

Acknowledgements

We appreciate very much all past and future comments, suggestions and contributions to this package and its documentation provided by GAP users and developers.

Contents

Chapter 1

Introduction

1.1 The package

This package provides functions for 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.
- We can test whether G is polycyclic.
- 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.

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, abbreviated PcpGroups.

The overall idea of the algorithm implemented in this package was first introduced by Ostheimer in 1996 [Ost96]. In 2001 Eick presented a more detailed version [Eic01]. This package contains an implementation of Eick's algorithm. A description of this implementation together with some refinements and extensions can be found in [AE05] and [Ass03].

Chapter 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 X of matrices of $GL(d,R)$ is given, where the ring R is either $\mathbb{Q},\mathbb{Z}$ or a finite field. Let $G = \langle X \rangle$. If the group G is polycyclic, then the following functions determine a PcpGroup isomorphic to G.

2.1.1 PcpGroupByMatGroup

▷ `PcpGroupByMatGroup(G)` (operation)

G is a subgroup of $GL(d,R)$ where $R = \mathbb{Q},\mathbb{Z}$ or $\mathbb{F}_q$. If *G* is polycyclic, then this function determines a PcpGroup isomorphic to *G*. If *G* is not polycyclic, then this function returns `fail`.

2.1.2 IsomorphismPcpGroup

▷ `IsomorphismPcpGroup(G)` (method)

G is a subgroup of $GL(d,R)$ where $R = \mathbb{Q},\mathbb{Z}$ or $\mathbb{F}_q$. If *G* is polycyclic, then this function determines an isomorphism onto a PcpGroup. If *G* is not polycyclic, then this function returns `fail`.

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.

2.1.3 ImagesRepresentative

▷ `ImagesRepresentative(map, elm)` (method)

▷ `ImageElm(map, elm)` (method)

▷ `ImagesSet(map, elms)` (method)

Here *map* is an isomorphism from a polycyclic matrix group *G* onto a PcpGroup *H* calculated by `IsomorphismPcpGroup` (2.1.2). These methods 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*.

2.1.4 IsSolvableGroup

▷ `IsSolvableGroup(G)` (method)

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`.

2.1.5 IsTriangularizableMatGroup

▷ `IsTriangularizableMatGroup(G)` (property)

G is a subgroup of $GL(d,\mathbb{Q})$. This function tests if *G* is triangularizable (possibly over a finite field extension) and returns `true` or `false`.

2.1.6 IsPolycyclicGroup

▷ `IsPolycyclicGroup(G)` (method)

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

2.2 Module series

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

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

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

2.2.1 RadicalSeriesSolvableMatGroup

▷ `RadicalSeriesSolvableMatGroup(G)` (operation)

This function returns a radical series for the $\mathbb{Q}[G]$-module $\mathbb{Q}^d$, where *G* is a solvable subgroup of $GL(d,\mathbb{Q})$.

A radical series of $\mathbb{Q}^d$ can be refined to a homogeneous series.

2.2.2 HomogeneousSeriesAbelianMatGroup

▷ `HomogeneousSeriesAbelianMatGroup(G)` (function)

A module is said to be homogeneous if it is the direct sum of pairwise irreducible isomorphic submodules. A homogeneous series of a module is a submodule series such that the factors are homogeneous. This function returns a homogeneous series for the $\mathbb{Q}[G]$-module $\mathbb{Q}^d$, where `G` is an abelian subgroup of $GL(d,\mathbb{Q})$.

2.2.3 HomogeneousSeriesTriangularizableMatGroup

▷ `HomogeneousSeriesTriangularizableMatGroup(G)` (function)

A module is said to be homogeneous if it is the direct sum of pairwise irreducible isomorphic submodules. A homogeneous series of a module is a submodule series such that the factors are homogeneous. This function returns a homogeneous series for the $\mathbb{Q}[G]$-module $\mathbb{Q}^d$, where `G` is a triangularizable subgroup of $GL(d,\mathbb{Q})$.

A homogeneous series can be refined to a composition series.

2.2.4 CompositionSeriesAbelianMatGroup

▷ `CompositionSeriesAbelianMatGroup(G)` (function)

A composition series of a module is a submodule series such that the factors are irreducible. This function returns a composition series for the $\mathbb{Q}[G]$-module $\mathbb{Q}^d$, where `G` is an abelian subgroup of $GL(d,\mathbb{Q})$.

2.2.5 CompositionSeriesTriangularizableMatGroup

▷ `CompositionSeriesTriangularizableMatGroup(G)` (function)

A composition series of a module is a submodule series such that the factors are irreducible. This function returns a composition series for the $\mathbb{Q}[G]$-module $\mathbb{Q}^d$, where `G` is a triangularizable subgroup of $GL(d,\mathbb{Q})$.

2.3 Subgroups

2.3.1 SubgroupsUnipotentByAbelianByFinite

▷ `SubgroupsUnipotentByAbelianByFinite(G)` (operation)

`G` is a subgroup of $GL(d,R)$ where $R = \mathbb{Q}$ or $\mathbb{Z}$. If `G` is polycyclic, then this function returns a record containing two normal subgroups T and U of G. The group T is unipotent-by-abelian (and thus triangularizable) and of finite index in `G`. The group U is unipotent and is such that T/U is abelian. If `G` is not polycyclic, then the algorithm returns `fail`.

2.4 Examples

2.4.1 PolExamples

▷ `PolExamples(l)` (function)

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. Some of the properties of the examples are summarised in the following table.

```
PolExamples      number generators      subgroup of      Hirsch length
          1                      3          GL(4,Z)                  6
          2                      2          GL(5,Z)                  6
          3                      2          GL(4,Q)                  4
          4                      2          GL(5,Q)                  6
          5                      9         GL(16,Z)                  3
          6                      6          GL(4,Z)                  3
          7                      6          GL(4,Z)                  3
          8                      7          GL(4,Z)                  3
          9                      5          GL(4,Q)                  3
         10                      4          GL(4,Q)                  3
         11                      5          GL(4,Q)                  3
         12                      5          GL(4,Q)                  3
         13                      5          GL(5,Q)                  4
         14                      6          GL(5,Q)                  4
         15                      6          GL(5,Q)                  4
         16                      5          GL(5,Q)                  4
         17                      5          GL(5,Q)                  4
         18                      5          GL(5,Q)                  4
         19                      5          GL(5,Q)                  4
         20                      7         GL(16,Z)                  3
         21                      5         GL(16,Q)                  3
         22                      4         GL(16,Q)                  3
         23                      5         GL(16,Q)                  3
         24                      5         GL(16,Q)                  3
```

Chapter 3

An example application

In this section we outline three example computations with functions from the previous chapter.

3.1 Presentation for rational matrix groups

Example

```
gap> mats :=
[ [ [ 1, 0, -1/2, 0 ], [ 0, 1, 0, 1 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 1/2, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 1 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 0, 0, 1 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -1/2, -3, 7/6 ], [ 0, 1, -1, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ -1, 3, 3, 0 ], [ 0, 0, 1, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 0, 1 ] ] ];

gap> G := Group( mats );
<matrix group with 5 generators>

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

gap> H := Image( nat );
Pcp-group with orders [ 2, 2, 3, 5, 5, 5, 0, 0, 0 ]

gap> h := GeneratorsOfGroup( H );
[ g1, g2, g3, g4, g5, g6, g7, g8, g9]

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

# take a random element of G
gap> exp :=  [ 1, 1, 1, 1, 0, 0, 0, 0, 1 ];;
gap> g := MappedVector( exp, mats2 );
[ [ -1, 17/2, -1, 233/6 ],
  [ 0, 1, 0, -2 ],
  [ 0, 1, -1, 2 ],
  [ 0, 0, 0, 1 ] ]

# map g into the image of nat
gap> i := ImageElm( nat, g );
g1*g2*g3*g4*g9
```

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

# compare the preimage with g
gap> PreImagesRepresentative( nat, i );
[ [ -1, 17/2, -1, 233/6 ],
  [ 0, 1, 0, -2 ],
  [ 0, 1, -1, 2 ],
  [ 0, 0, 0, 1 ] ]


gap> last = g;
true
```

3.2 Modules series

Example

```
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 ] ],
  [ [ 1, 0, 0, 79/138 ], [ 0, 1, 0, -275/828 ], [ 0, 0, 1, -197/414 ] ],
  [ [ 1, 0, -3, 2 ], [ 0, 1, 55/4, -55/8 ] ],
  [ [ 1, 4/15, 2/3, 1/6 ] ],
  [  ] ]
```

3.3 Triangularizable subgroups

Example

```
gap> G := PolExamples(3);
<matrix group with 2 generators>

gap> GeneratorsOfGroup( G );
[ [ [ 73/10, -35/2, 42/5, 63/2 ],
    [ 27/20, -11/4, 9/5, 27/4 ],
    [ -3/5, 1, -4/5, -9 ],
    [ -11/20, 7/4, -2/5, 1/4 ] ],
  [ [ -42/5, 423/10, 27/5, 479/10 ],
```

```
    [ -23/10, 227/20, 13/10, 231/20 ],
    [ 14/5, -63/5, -4/5, -79/5 ],
    [ -1/10, 9/20, 1/10, 37/20 ] ] ]

gap> subgroups := SubgroupsUnipotentByAbelianByFinite( G );
rec( T := <matrix group with 2 generators>,
  U := <matrix group with 4 generators> )

gap> GeneratorsOfGroup( subgroups.T );
[ [ [ 73/10, -35/2, 42/5, 63/2 ],
    [ 27/20, -11/4, 9/5, 27/4 ],
    [ -3/5, 1, -4/5, -9 ],
    [ -11/20, 7/4, -2/5, 1/4 ] ],
  [ [ -42/5, 423/10, 27/5, 479/10 ],
    [ -23/10, 227/20, 13/10, 231/20 ],
    [ 14/5, -63/5, -4/5, -79/5 ],
    [ -1/10, 9/20, 1/10, 37/20 ] ] ]

# so G is triangularizable!
```

Chapter 4

Installation

4.1 Installing this package

The Polenta package is part of the standard distribution of GAP and so normally there should be no need to install it separately. If by any chance it is not part of your GAP distribution, then 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 (**Reference: Installing a GAP Package**) of the GAP Reference Manual.

Note that the GAP-Packages Alnuth and Polycyclic are needed for this package. Normally they should be contained in your distribution. If not, they can be obtained at http://www.gap-system.org/Packages/packages.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 via the `LoadPackage` (**Reference: LoadPackage**) command.

4.3 Running the test suite

Once the package is installed, it is possible to check the correct installation by running the test suite of the package.

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

For more details on Test Files see Section (**Reference: Test Files**) of the GAP Reference Manual.

If the test suite runs into an error, even though the packages Polycyclic and Alnuth and their depdendencies have been correctly installed, then please send a message to `horn@mathematik.uni-kl.de` including the error message.

Chapter 5

Information Messages

It is possible to get informations about the status of the computation of the functions of Chapter 2 of this manual.

5.1 Info Class

5.1.1 InfoPolenta

▷ `InfoPolenta` (info class)

is the Info class of the Polenta package (for more details on the Info mechanism see Section (**Reference: Info Functions**) of the GAP Reference Manual). With the help of the function `SetInfoLevel(InfoPolenta,`*`level`*`)` you can change the info level of `InfoPolenta`.

- If `InfoLevel( InfoPolenta )` is equal to 0 then no information messages are displayed.
- If `InfoLevel( InfoPolenta )` is equal to 1 then basic informations about the process are provided. For further background on the displayed informations we refer to [Ass03] (publicly available via the Internet address `http://www.icm.tu-bs.de/ag_algebra/software/assmann/diploma.pdf`).
- If `InfoLevel( InfoPolenta )` is equal to 2 then, in addition to the basic information, the generators of computed subgroups and module series are displayed.

5.2 Example

Example

```
gap> SetInfoLevel( InfoPolenta, 1 );

gap> PcpGroupByMatGroup( PolExamples(11) );
#I  Determine a constructive polycyclic sequence
    for the input group ...
#I
#I  Chosen admissible prime: 3
#I
#I  Determine a constructive polycyclic sequence
    for the image under the p-congruence homomorphism ...
```

```
#I  finished.
#I  Finite image has relative orders [ 3, 2, 3, 3, 3 ].
#I
#I  Compute normal subgroup generators for the kernel
    of the p-congruence homomorphism ...
#I  finished.
#I
#I  Compute the radical series ...
#I  finished.
#I  The radical series has length 4.
#I
#I  Compute the composition series ...
#I  finished.
#I  The composition series has length 5.
#I
#I  Compute a constructive polycyclic sequence
    for the induced action of the kernel to the composition series ...
#I  finished.
#I  This polycyclic sequence has relative orders [  ].
#I
#I  Calculate normal subgroup generators for the
    unipotent part ...
#I  finished.
#I
#I  Determine a constructive polycyclic  sequence
    for the unipotent part ...
#I  finished.
#I  The unipotent part has relative orders
#I  [ 0, 0, 0 ].
#I
#I  ... computation of a constructive
    polycyclic sequence for the whole group finished.
#I
#I  Compute the relations of the polycyclic
    presentation of the group ...
#I  Compute power relations ...
#I  ... finished.
#I  Compute conjugation relations ...
#I  ... finished.
#I  Update polycyclic collector ...
#I  ... finished.
#I  finished.
#I
#I  Construct the polycyclic presented group ...
#I  finished.
#I
Pcp-group with orders [ 3, 2, 3, 3, 3, 0, 0, 0 ]


gap> SetInfoLevel( InfoPolenta, 2 );

gap> PcpGroupByMatGroup( PolExamples(11) );
```

```
#I  Determine a constructive polycyclic sequence
    for the input group ...
#I
#I  Chosen admissible prime: 3
#I
#I  Determine a constructive polycyclic sequence
    for the image under the p-congruence homomorphism ...
#I  finished.
#I  Finite image has relative orders [ 3, 2, 3, 3, 3 ].
#I
#I  Compute normal subgroup generators for the kernel
    of the p-congruence homomorphism ...
#I  finished.
#I  The normal subgroup generators are
#I  [ [ [ 1, -3/2, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 0, 0, 24 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, 3, 15 ], [ 0, 1, 0, 6 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, 3, 9 ], [ 0, 1, 0, 6 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3/2, 3/2, 3/2 ], [ 0, 1, 0, 3 ], [ 0, 0, 1, -3 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3/2, 9/2, -69/2 ], [ 0, 1, 0, 9 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ] ]
    , [ [ 1, 0, 0, -24 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -9 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -15 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, 0, 9 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -9 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, 0, 9 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3/2, -3/2, -9/2 ], [ 0, 1, 0, -3 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ]
     ],
  [ [ 1, -3, -3, -12 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, -3/2, -21 ], [ 0, 1, 0, -3 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3/2, 3/2, 9/2 ], [ 0, 1, 0, 3 ], [ 0, 0, 1, -3 ], [ 0, 0, 0, 1 ] ] ]
#I
#I  Compute the radical series ...
#I  finished.
#I  The radical series has length 4.
#I  The radical series is
#I  [ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ], [ [ 0, 0, 0, 1 ] ],
  [  ] ]
#I
#I  Compute the composition series ...
#I  finished.
#I  The composition series has length 5.
#I  The composition series is
#I  [ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ], [ [ 0, 0, 0, 1 ] ], [  ] ]
#I
#I  Compute a constructive polycyclic sequence
    for the induced action of the kernel to the composition series ...
#I  finished.
#I  This polycyclic sequence has relative orders [  ].
```

```
#I
#I  Calculate normal subgroup generators for the
    unipotent part ...
#I  finished.
#I  The normal subgroup generators for the unipotent part are
#I  [ [ [ 1, -3/2, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 0, 0, 24 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, 3, 15 ], [ 0, 1, 0, 6 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, 3, 9 ], [ 0, 1, 0, 6 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3/2, 3/2, 3/2 ], [ 0, 1, 0, 3 ], [ 0, 0, 1, -3 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3/2, 9/2, -69/2 ], [ 0, 1, 0, 9 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ] ]
    , [ [ 1, 0, 0, -24 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -9 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -15 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, 0, 9 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, -3, -9 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3, 0, 9 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, -3/2, -3/2, -9/2 ], [ 0, 1, 0, -3 ], [ 0, 0, 1, 3 ], [ 0, 0, 0, 1 ]
     ],
  [ [ 1, -3, -3, -12 ], [ 0, 1, 0, -6 ], [ 0, 0, 1, 6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3, -3/2, -21 ], [ 0, 1, 0, -3 ], [ 0, 0, 1, -6 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 3/2, 3/2, 9/2 ], [ 0, 1, 0, 3 ], [ 0, 0, 1, -3 ], [ 0, 0, 0, 1 ] ] ]
#I
#I  Determine a constructive polycyclic  sequence
    for the unipotent part ...
#I  finished.
#I  The unipotent part has relative orders
#I  [ 0, 0, 0 ].
#I
#I  ... computation of a constructive
    polycyclic sequence for the whole group finished.
#I
#I  Compute the relations of the polycyclic
    presentation of the group ...
#I  Compute power relations ...
.....
#I  ... finished.
#I  Compute conjugation relations ...
..............................................
#I  ... finished.
#I  Update polycyclic collector ...
#I  ... finished.
#I  finished.
#I
#I  Construct the polycyclic presented group ...
#I  finished.
#I
Pcp-group with orders [ 3, 2, 3, 3, 3, 0, 0, 0 ]
```

References

[AE05] B. Assmann and B. Eick. Computing polycyclic presentations for polycyclic rational matrix groups. *J. Symbolic Comput.*, 40(6):1269--1284, 2005. 4

[Ass03] B. Assmann. Polycyclic presentations for matrix groups. Diplomarbeit, TU Braunschweig, 2003. http://www.icm.tu-bs.de/ag_algebra/software/assmann. 4, 13

[Eic01] B. Eick. Algorithms for polycyclic groups. Habilitationsschrift, Gesamthochschule Kassel, 2001. 4

[EN00] B. Eick and W. Nickel. *Polycyclic*, 2000. GAP package. 4, 5

[Ost96] G. Ostheimer. *Algorithms for Polycyclic-by-finite groups*. PhD thesis, Rutgers University, 1996. 4

[Sim94] C. C. Sims. *Computation with finitely presented groups*. Cambridge University Press, 1994. 4

Index