Circle

Adjoint groups of finite radical algebras

Version 1.1

August 2006

Alexander Konovalov
Panagiotis Soules

Alexander Konovalov — Email: konovalov@member.ams.org
— Homepage: http://homepages.vub.ac.be/~okonoval/
— Address: Department of Mathematics
Zaporozhye National University
Zaporozhye, 69063 Ukraine
Current address:
Department of Mathematics
Vrije Universiteit Brussel
Pleinlaan 2, Brussels
B-1050 Belgium

Panagiotis Soules — Email: psoules@math.uoa.gr
— Address: Department of Mathematics
National and Capodistrian University of Athens
Panepistimioupolis, GR-15784, Athens, Greece

Abstract

The GAP4 package Circle extends the GAP functionality for computations in adjoint groups of associative rings. It provides functionality to construct circle objects that will respect the circle multiplication $r \cdot s = r + s + rs$, create multiplicative groups, generated by such objects, and compute groups of elements, invertible with respect to this operation, for finite radical algebras and finite associative rings without one.

Copyright

Acknowledgements

We acknowledge very much Alexander Hulpke for helpful comments and advise.

Contents

Chapter 1

Introduction

1.1 General aims

Let R be an associative ring, not necessarily with one. The set of all elements of R forms a monoid with the neutral element 0 from R under the operation $r \cdot s = r + s + rs$ defined for all r and s of R. This operation is called the *circle multiplication*, and it is also known as the *star multiplication*. The monoid of elements of R under the circle multiplication is called the adjoint semigroup of R and is denoted by R^{ad}. The group of all invertible elements of this monoid is called the adjoint group of R and is denoted by R^*.

These notions naturally lead to a number of questions about the connection between a ring and its adjoint group, for example, how the ring properties will determine properties of the adjoint group; which groups can appear as adjoint groups of rings; which rings can have adjoint groups with prescribed properties, etc.

For example, V. O. Gorlov in [Gor95] gives a full list of finite nilpotent algebras R, such that $R^2 \neq 0$ and the adjoint group of R is metacyclic (but not cyclic).

S. V. Popovich and Ya. P. Sysak in [PS97] characterize all quasiregular algebras such that all subgroups of their adjoint group are their subalgebras. In particular, they show that all algebras of such type are nilpotent with nilpotency index at most three.

Various connections between properties of a ring and its adjoint group were considered by O. D. Artemovych and Yu. B. Ishchuk in [AI97].

B. Amberg and L. S. Kazarin in [AK00] give the description of all nonisomorphic finite p-groups that can occur as the adjoint group of some nilpotent p-algebra of the dimension at most 5.

In [AS01] B. Amberg and Ya. P. Sysak give a survey of results on adjoint groups of radical rings, including such topics as subgroups of the adjoint group; nilpotent groups which are isomorphic to the adjoint group of some radical ring; adjoint groups of finite nilpotent p-algebras. The authors continued their investigations in further papers [AS02] and [AS04].

In [KS04] L. S. Kazarin and P. Soules study associative nilpotent algebras over a field of positive characteristic whose adjoint group has a small number of generators.

The main objective of the proposed GAP4 package Circle is to extend the GAP functionality for computations in adjoint groups of associative rings to make it possible to use the GAP system for the investigation of the above described questions.

Circle provides functionality to construct circle objects that will respect the circle multiplication $r \cdot s = r + s + rs$, create multiplicative groups, generated by such objects, and compute groups of elements, invertible with respect to this operation, for finite radical algebras and finite associative rings without one.

Also we hope that the package will be useful as an example of extending the GAP system with new multiplicative objects.

1.2 Installation and system requirements

Circle does not use external binaries and, therefore, works without restrictions on the type of the operating system. It is designed for GAP4.4 and no compatibility with previous releases of GAP4 is guaranteed.

To use the Circle online help it is necessary to install the GAP4 package GAPDoc by Frank Lübeck and Max Neunhöffer, which is available from the GAP site or from `http://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/`.

Circle is distributed in standard formats (`zoo`, `tar.gz`, `tar.bz2`, `-win.zip`) and can be obtained from `http://homepages.vub.ac.be/~okonoval/circle.htm`. To unpack the archive `circle-1.1.zoo` you need the program `unzoo`, which can be obtained from the GAP homepage `http://www.gap-system.org/` (see section 'Distribution'). To install Circle, copy this archive into the `pkg` subdirectory of your GAP4.4 installation. The subdirectory `circle` will be created in the `pkg` directory after the following command:

```
unzoo -x circle-1.1.zoo
```

Chapter 2

Circle functions

To use the Circle package first you need to load it as follows:

Example
```
gap> LoadPackage("circle");
-----------------------------------------------------------------------------
Loading  Circle 1.0 (Adjoint groups of associative rings)
by Alexander Konovalov (http://homepages.vub.ac.be/~okonoval/) and
   Panagiotis Soules (psoules@math.uoa.gr).
-----------------------------------------------------------------------------
true
gap>
```

2.1 Circle objects

Because for elements of the ring R the ordinary multiplication is already denoted by `*`, for the implementation of the circle multiplication in the adjoint semigroup we need to wrap up ring elements as CircleObjects, for which `*` is defined to be the circle multiplication.

2.1.1 CircleObject

◊ `CircleObject( x )` (attribute)

Let `x` be a ring element. Then `CircleObject(x)` returns the corresponding circle object. If `x` lies in the family `fam`, then `CircleObject(x)` lies in the family `CircleFamily` (2.1.5), corresponding to the family `fam`.

Example
```
gap> x := 2;
2
gap> a := CircleObject( x );
CircleObject( 2 )
gap> FamilyObj( a ) = CircleFamily( FamilyObj( x ) );
true
```

2.1.2 UnderlyingRingElement

◊ UnderlyingRingElement(x) (operation)

Let x be a circle object. Then UnderlyingRingElement(x) returns the corresponding ring element.

Example
```
gap> x := 2;
2
gap> a := CircleObject( x );
CircleObject( 2 )
gap> UnderlyingRingElement( a );
2
gap> FamilyObj( a ) = CircleFamily( FamilyObj( x ) );
true
```

2.1.3 IsCircleObject

◊ IsCircleObject(x) (Category)
◊ IsCircleObjectCollection(x) (Category)

An object x lies in the category IsCircleObject if and only if it lies in a family constructed by CircleFamily (2.1.5). Since circle objects can be multiplied via * with elements in their family, and we want to implement operations One and Inverse to deal with groups they generate, circle objects are implemented in the category IsMultiplicativeElementWithInverse.

Example
```
gap> IsCircleObject( 2 );
false
gap> IsCircleObject( CircleObject( 2 ) );
true
gap> IsMultiplicativeElementWithInverse( CircleObject( 2 ) );
true
gap> One( CircleObject( 2 ) );
CircleObject( 0 )
gap> CircleObject( -2 )^-1;
CircleObject( -2 )
```

2.1.4 IsPositionalObjectOneSlotRep

◊ IsPositionalObjectOneSlotRep(x) (Representation)
◊ IsDefaultCircleObject(x) (Representation)

To store the corresponding circle object, we need only to store the underlying ring element. Since this is quite common situation, we defined the representation IsPositionalObjectOneSlotRep for a more general case. Then we defined IsDefaultCircleObject as a synonym of IsPositionalObjectOneSlotRep for objects in IsCircleObject (2.1.3).

Example

```
gap> IsPositionalObjectOneSlotRep( CircleObject( 2 ) );
true
gap> IsDefaultCircleObject( CircleObject( 2 ) );
true
```

2.1.5 CircleFamily

◊ `CircleFamily( fam )` (attribute)

`CircleFamily(fam)` is a family, elements of which are in one-to-one correspondence with elements of the family `fam`, but with the circle multiplication as an infix multiplication. That is, for x, y in `fam`, the product of their images in the `CircleFamily(fam)` will be the image of $x+y+xy$.

Example

```
gap> x:=CircleObject(2)*CircleObject(3);
CircleObject( 11 )
gap> y:=CircleObject(2+3+2*3);
CircleObject( 11 )
gap> x=y;
true
gap> FamilyObj(x)=FamilyObj(y);
true
```

2.2 Operations with circle objects

2.2.1 OneOp

◊ `OneOp( x )` (operation)

This operation returns the multiplicative neutral element for the circle object `x`. The result is the circle object corresponding to the additive neutral element of the appropriate ring.

Example

```
gap> One( CircleObject( 5 ) );
CircleObject( 0 )
gap> One( CircleObject( 5 ) ) = CircleObject( Zero( 5 ) );
true
gap> One( CircleObject( [ [ 1, 1 ],[ 0, 1 ] ] ) );
CircleObject( [ [ 0, 0 ], [ 0, 0 ] ] )
```

2.2.2 InverseOp

◊ InverseOp(x) (operation)

For a circle object x, returns the multiplicative inverse of x with respect to the circle multiplication; if such one does not exist then fail is returned.

Example
```
gap> CircleObject( -2 )^-1;
CircleObject( -2 )
gap> CircleObject( 2 )^-1;
CircleObject( -2/3 )
gap> CircleObject( -2 )*CircleObject( -2 )^-1;
CircleObject( 0 )
gap> m := CircleObject( [ [ 1, 1 ], [ 0, 1 ] ] );
CircleObject( [ [ 1, 1 ], [ 0, 1 ] ] )
gap> m^-1;
CircleObject( [ [ -1/2, -1/4 ], [ 0, -1/2 ] ] )
gap> m * m^-1;
CircleObject( [ [ 0, 0 ], [ 0, 0 ] ] )
gap> CircleObject( [ [ 0, 1 ], [ 1, 0 ] ] )^-1;
fail
```

2.2.3 IsUnit

◊ IsUnit(R, x) (operation)
◊ IsUnit(x) (operation)

Let x be a circle object corresponding to an element of the ring R. Then the operation IsUnit returns true, if x is invertible in R with respect to the circle multiplication, and false otherwise.

Example
```
gap> IsUnit( Integers, CircleObject( -2 ) );
true
gap> IsUnit( Integers, CircleObject( 2 ) );
false
gap> IsUnit( Rationals, CircleObject( 2 ) );
true
gap> IsUnit( ZmodnZ(8), CircleObject( ZmodnZObj(2,8) ) );
true
gap> m := CircleObject( [ [ 1, 1 ],[ 0, 1 ] ] );
CircleObject( [ [ 1, 1 ], [ 0, 1 ] ] )
gap> IsUnit( FullMatrixAlgebra( Rationals, 2 ), m );
true
```

In the second form the result will be returned with respect to the default ring of the circle object x.

Example
```
gap> IsUnit( CircleObject( -2 ) );
true
gap> IsUnit( CircleObject( 2 ) );
false
gap> IsUnit( CircleObject( ZmodnZObj(2,8) ) );
true
gap> IsUnit( CircleObject( [ [ 1, 1 ],[ 0, 1 ] ] ) );
true
```

2.2.4 IsCircleUnit

◊ IsCircleUnit(R, x) (operation)
◊ IsCircleUnit(x) (operation)

Let x be an element of the ring R. Then IsCircleUnit(R, x) determines whether x is invertible in R with respect to the circle multilpication. This is equivalent to the condition that 1+x is a unit in R with respect to the ordinary multiplication.

Example
```
gap> IsCircleUnit( Integers, -2 );
true
gap> IsCircleUnit( Integers, 2 );
false
gap> IsCircleUnit( Rationals, 2 );
true
gap> IsCircleUnit( ZmodnZ(8), ZmodnZObj(2,8) );
true
gap> m := [ [ 1, 1 ],[ 0, 1 ] ];
[ [ 1, 1 ], [ 0, 1 ] ]
gap> IsCircleUnit( FullMatrixAlgebra(Rationals,2), m );
true
```

In the second form the result will be returned with respect to the default ring of x.

Example
```
gap> IsCircleUnit( -2 );
true
gap> IsCircleUnit( 2 );
false
gap> IsCircleUnit( ZmodnZObj(2,8) );
true
gap> IsCircleUnit( [ [ 1, 1 ],[ 0, 1 ] ] );
true
```

2.3 Construction of the adjoint group

2.3.1 AdjointGroup

◊ `AdjointGroup( R )` (attribute)

If `R` is a finite radical algebra then `AdjointGroup(R)` will return the adjoint group of `R`, given as a group generated by a set of circle objects.

To compute the adjoint group of a finite radical algebra, Circle uses the fact that all elements of a radical algebra form a group with respect to the circle multiplication. Thus, the adjoint group of `R` coincides with `R` elementwise, and we can randomly select an appropriate set of generators for the adjoint group.

The warning is displayed by `IsGeneratorsOfMagmaWithInverses` method defined in `gap4r4/lib/grp.gi` and may be ignored.

WARNINGS:

1. The set of generators of the returned group is not required to be the minimal generating set.

2. `AdjointGroup` is stored as an attribute of `R`, so for the same copy of `R` calling it again you will get the same result. But if you will create another copy of `R` in the future, the output may differ because of the random selection of generators. If you want to have the same generating set, next time you should construct a group immediately specifying circle objects that generate it.

3. In most cases, to investigate some properties of the adjoint group, it is necessary first to convert it to an isomorphic permutation group or to a PcGroup.

For example, we can create the following commutative 2-dimensional radical algebra of order 4 over the field of two elements, and show that its adjoint group is a cyclic group of order 4:

Example
```
gap> x:=[ [ 0, 1, 0 ],
>         [ 0, 0, 1 ],
>         [ 0, 0, 0 ] ];;
gap> R := Algebra( GF(2), [ One(GF(2))*x ] );
<algebra over GF(2), with 1 generators>
gap> RadicalOfAlgebra( R ) = R;
true
gap> Dimension(R);
2
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( [ [ 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) ] ] ) ]
<group of size 4 with 2 generators>
gap> Size( R ) = Size( G );
true
gap> StructureDescription( G );
"C4"
```

In the following example we construct a non-commutative 3-dimensional radical algebra of order 8 over the field of two elements, and demonstrate that its adjoint group is the dihedral group of order 8:

Example

```
gap> x:=[ [ 0, 1, 0 ],
>         [ 0, 0, 0 ],
>         [ 0, 0, 0 ] ];;
gap> y:=[ [ 0, 0, 0 ],
>         [ 0, 0, 1 ],
>         [ 0, 0, 0 ] ];;
gap> R := Algebra( GF(2), One(GF(2))*[x,y] );
<algebra over GF(2), with 2 generators>
gap> RadicalOfAlgebra(R) = R;
true
gap> Dimension(R);
3
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( [ [ 0*Z(2), Z(2)^0, Z(2)^0 ], [ 0*Z(2), 0*Z(2), Z(2)^0 ],
      [ 0*Z(2), 0*Z(2), 0*Z(2) ] ] ) ]
<group of size 8 with 2 generators>
gap> StructureDescription( G );
"D8"
```

If the ring R is not a radical algebra, but also is not a ring with one, then Circle will use another approach. We will enumerate all elements of the ring R and select those that are units with respect to the circle multiplication. Then we will use a random approach similar to the case of the radical algebra, to find some generating set of the adjoint group. Again, all warnings 1-3 above refer also to this case.

Of course, enumeration of all elements of R should be feasible for this computation. In the following example we demonstrate how it works for rings, generated by residue classes:

Example

```
gap> R := Ring( [ ZmodnZObj(2,8) ] );
<ring with 1 generators>
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( ZmodnZObj( 2, 8 ) ) ]
<group of size 4 with 2 generators>
gap> StructureDescription( G );
"C2 x C2"
gap> R := Ring( [ ZmodnZObj(2,256) ] );
<ring with 1 generators>
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( ZmodnZObj( 234, 256 ) ) ]
<group of size 128 with 2 generators>
gap> StructureDescription( G );
"C64 x C2"
```

If R has a unity 1, then the set $1+R^{ad}$, where R^{ad} is the adjoint semigroup of R, coincides with the multiplicative semigroup R^{mult} of R, and the map $r \mapsto (1+r)$ for r in R is an isomorphism from R^{ad} onto R^{mult}.

Similarly, the set $1+R^*$, where R^* is the adjoint group of R, coincides with the unit group of R, which we denote $U(R)$, and the map $r \mapsto (1+r)$ for r in R is an isomorphism from R^* onto $U(R)$.

We demonstrate this isomorphism using the following example.

Example
```
gap> FG := GroupRing( GF(2), DihedralGroup(8) );
<algebra-with-one over GF(2), with 3 generators>
gap> R := AugmentationIdeal( FG );
<two-sided ideal in <algebra-with-one over GF(2), with 3 generators>,
  (dimension 7)>
gap> G := AdjointGroup( R );
#I  default 'IsGeneratorsOfMagmaWithInverses' method returns 'true' for
[ CircleObject( (Z(2)^0)*f2+(Z(2)^0)*f1*f2 ) ]
<group of size 128 with 4 generators>
gap> IdGroup( G );
[ 128, 170 ]
gap> IdGroup( Units( FG ) );
#I  LAGUNA package: Computing the unit group ...
[ 128, 170 ]
```

This is is why we did not implemented in Circle adjoint groups for associative rings with one. If you will try to compute the adjoint group for such a ring, you will get an error message telling that you can investigate the unit group of the ring instead.

If R is infinite, an error message will appear, telling that Circle does not provide methods to deal with infinite rings.

2.4 Service functions

2.4.1 InfoCircle

◊ `InfoCircle` (info class)

`InfoCircle` is a special Info class for Circle algorithms. It has 2 levels: 0 (default) and 1. To change info level to `k`, use command `SetInfoLevel(InfoCircle, k)`.

Example
```
gap> SetInfoLevel( InfoCircle, 1 );
gap> SetInfoLevel(InfoCircle,1);
gap> R := Ring( [ ZmodnZObj(2,8) ]);
<ring with 1 generators>
gap> G := AdjointGroup( R );
#I  Circle : <R> is not a radical algebra, computing circle units ...
#I  Circle : searching generators for adjoint group ...
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( ZmodnZObj( 6, 8 ) ) ]
<group of size 4 with 2 generators>
gap> SetInfoLevel( InfoCircle, 0 );
```

2.4.2 CIRCLEBuildManual

◊ `CIRCLEBuildManual( )` (function)

This function is used to build the manual in the following formats: DVI, PDF, PS, HTML and text for online help. We recommend that the user should have a recent and fairly complete TEX distribution. Since Circle is distributed together with its manual, it is not necessary for the user to use this function. Normally it is intended to be used by the developers only. This is the only function of Circle which requires UNIX/Linux environment.

2.4.3 CIRCLEBuildManualHTML

◊ `CIRCLEBuildManualHTML( )` (function)

This fuction is used to build the manual only in HTML format. This does not depend on the availability of the TEX installation and works under Windows and MacOS as well. Since Circle is distributed together with its manual, it is not necessary for the user to use this function. Normally it is intended to be used by the developers only.

Chapter 3

A sample computation with Circle

Here we give an example to give the reader an idea what Circle is able to compute.

It was proved in [KS04] that if R is a finite nilpotent two-generated algebra over a field of characteristic $p > 3$ whose adjoint group has at most three generators, then the dimension of R is not greater than 9. Also, an example of the 6-dimensional such algebra with the 3-generated adjoint group was given there. We will construct the algebra from this example and investigate it using Circle. First we create two matrices that determine its generators:

Example
```
gap> x:=[ [ 0, 1, 0, 0, 0, 0, 0 ],
>         [ 0, 0, 0, 1, 0, 0, 0 ],
>         [ 0, 0, 0, 0, 1, 0, 0 ],
>         [ 0, 0, 0, 0, 0, 0, 1 ],
>         [ 0, 0, 0, 0, 0, 1, 0 ],
>         [ 0, 0, 0, 0, 0, 0, 0 ],
>         [ 0, 0, 0, 0, 0, 0, 0 ] ];;
gap> y:=[ [ 0, 0, 1, 0, 0, 0, 0 ],
>         [ 0, 0, 0, 0,-1, 0, 0 ],
>         [ 0, 0, 0, 1, 0, 1, 0 ],
>         [ 0, 0, 0, 0, 0, 1, 0 ],
>         [ 0, 0, 0, 0, 0, 0,-1 ],
>         [ 0, 0, 0, 0, 0, 0, 0 ],
>         [ 0, 0, 0, 0, 0, 0, 0 ] ];;
```

Now we construct this algebra in characteristic five and check its basic properties:

Example
```
gap> R := Algebra( GF(5), One(GF(5))*[x,y] );
<algebra over GF(5), with 2 generators>
gap> Dimension( R );
6
gap> Size( R );
15625
gap> RadicalOfAlgebra( R ) = R;
true
```

Then we compute the adjoint group of `R`. During the computation a warning will be displayed. It is caused by the method for `IsGeneratorsOfMagmaWithInverses` defined in the file `gap4r4/lib/grp.gi` from the GAP library, and may be safely ignored.

Example
```
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( [ [ 0*Z(5), Z(5), Z(5), Z(5)^3, Z(5), 0*Z(5), Z(5)^2 ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), Z(5), Z(5)^3, Z(5)^3, Z(5)^3 ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), Z(5), Z(5), 0*Z(5), Z(5) ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), Z(5), Z(5) ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), Z(5), Z(5)^3 ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5) ],
      [ 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5), 0*Z(5) ] ] ) ]
<group of size 15625 with 3 generators>
```

Now we can find the minimal generating set of `G` and check that `G` it is 3-generated. To do this, first we need to convert it to the isomorphic PcGroup:

Example
```
gap> f := IsomorphismPcGroup( G );;
gap> H := Image( f );
Group([ f1, f2, f3, f4, f5, f6 ])
gap> gens := MinimalGeneratingSet( H );
[ f1, f2, f5 ]
gap> gens:=List( gens, x -> UnderlyingRingElement(PreImage(f,x)));;
gap> Perform(gens,Display);
 . 3 3 4 4 . 1
 . . . 3 2 1 4
 . . . 3 3 2 4
 . . . . . 3 3
 . . . . . 3 2
 . . . . . . .
 . . . . . . .
 . 3 1 1 . . .
 . . . 3 4 . 1
 . . . 1 3 2 .
 . . . . . 1 3
 . . . . . 3 4
 . . . . . . .
 . . . . . . .
 . 2 2 3 2 . 4
 . . . 2 3 3 3
 . . . 2 2 . 2
 . . . . . 2 2
 . . . . . 2 3
 . . . . . . .
 . . . . . . .
```

It appears that the adjoint group of the algebra from example will be 3-generated in characteristic three as well:

Example

```
gap> R := Algebra( GF(3), One(GF(3))*[x,y] );
<algebra over GF(3), with 2 generators>
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( [ [ 0*Z(3), 0*Z(3), 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), Z(3), Z(3)^0, Z(3)^0 ],
      [ 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3), Z(3), 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), 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), 0*Z(3), 0*Z(3) ] ] ) ]
<group of size 729 with 3 generators>
gap> H := Image( IsomorphismPcGroup( G ) );
Group([ f1, f2, f3, f4, f5, f6 ])
gap> MinimalGeneratingSet( H );
[ f1, f2, f4 ]
```

But this is not the case in characteristic two, where the adjoint group is 4-generated:

Example

```
gap> R := Algebra( GF(2), One(GF(2))*[x,y] );
<algebra over GF(2), with 2 generators>
gap> G := AdjointGroup( R );
#I  default `IsGeneratorsOfMagmaWithInverses' method returns `true' for
[ CircleObject( [ [ 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, 0*Z(2) ],
      [ 0*Z(2), 0*Z(2), 0*Z(2), 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), Z(2)^0, Z(2)^0 ],
      [ 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), 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) ] ] ) ]
<group of size 64 with 4 generators>
gap> H := Image( IsomorphismPcGroup( G ) );
Group([ f1, f2, f3, f4, f5, f6 ])
gap> MinimalGeneratingSet( H );
[ f1, f2, f4, f5 ]
```

References

[AI97] O. D. Artemovych and Yu. B. Ishchuk. On semiperfect rings determined by adjoint groups. *Mat. Stud.*, 8(2):162–170, 237, 1997. 4

[AK00] B. Amberg and L. S. Kazarin. On the adjoint group of a finite nilpotent p-algebra. *J. Math. Sci. (New York)*, 102(3):3979–3997, 2000. 4

[AS01] B. Amberg and Ya. P. Sysak. Radical rings and their adjoint groups. In *Topics in infinite groups*, volume 8 of *Quad. Mat.*, pages 21–43. Dept. Math., Seconda Univ. Napoli, Caserta, 2001. 4

[AS02] B. Amberg and Ya. P. Sysak. Radical rings with soluble adjoint groups. *J. Algebra*, 247(2):692–702, 2002. 4

[AS04] B. Amberg and Ya. P. Sysak. Associative rings with metabelian adjoint group. *J. Algebra*, 277(2):456–473, 2004. 4

[Gor95] V. O. Gorlov. Finite nilpotent algebras with a metacyclic quasiregular group. *Ukraïn. Mat. Zh.*, 47(10):1426–1431, 1995. 4

[KS04] L. S. Kazarin and P. Soules. Finite nilpotent p-algebras whose adjoint group has three generators. *JP J. Algebra Number Theory Appl.*, 4(1):113–127, 2004. 4, 15

[PS97] S. V. Popovich and Ya. P. Sysak. Radical algebras whose subgroups of adjoint groups are subalgebras. *Ukraïn. Mat. Zh.*, 49(12):1646–1652, 1997. 4

Index