Manual of the MAGMA package `ArtinAlgebras`

Gabor Wiese *

8th August 2008

Abstract

This is a short manual for the MAGMA [1] package `ArtinAlgebras`, which can be downloaded from the author's webpage. Most of the package was originally written for computations of weight one modular forms over finite fields [5], [6], [7] and extended for computations of Hecke algebras, also over finite fields, carried out together with L.J.P. Kilford [3].

This package is currently needed by the author's MAGMA packages `Weight1` and `HeckeAlgebra`, which have grown out of the two above topics.

Contents

1 Installation and Example

You need the package `ArtinAlgebras`, which can be downloaded from the author's webpage. For installation, just unpack the tar-file.

Suppose that `PATH` contains `ArtinAlgebras.spec`. Then we attach the package using

> AttachSpec("PATH/ArtinAlgebras.spec");

We first need to create an interesting algebra. Our main source of examples and the motivation for this package are Hecke algebras, i.e. matrix algebras generated by a certain set of commuting matrices. Let us first get some of those.

> M := CuspidalSubspace(ModularSymbols(229,2,1));

> L := [HeckeOperator(M,n) : n in [1..50]];

Now ***L*** contains 50 commuting matrices. We can form the algebra generated by these. It is a matrix algebra over the rationals.

> A := MatrixAlgebra(L);

Any Artin algebra (like matrix algebras over finite fields) is the direct product of its localisations at the (finitely many, necessarily maximal) prime ideals. We compute this decomposition like this:

> D := Decomposition(A);

Let's see how many factors we have.

> #D;

How do we now access an individual factor? This is easily done; for the first one:

> A1 := BaseChange(A,D[1]);

*Institut für Experimentelle Mathematik, Universität Duisburg-Essen, Ellernstr. 29, D-45326 Essen, Germany. `http://maths.pratum.net/`, e-mail: `gabor.wiese@uni-due.de`

> Dimension(A1);

Out of interest, we also computed the dimension of the factor. The same for the second and the third factor:

> A2 := BaseChange(A,D[2]);

> Dimension(A2);

> Basis(A2);

> A3 := BaseChange(A,D[3]);

> Dimension(A3);

We can compute the corresponding idempotents (using a direct algorithm):

> I := CompleteOrthogonalIdempotents(A);

We can also get the idempotents corresponding to the decomposition ***D*** above.

> I1 := Idempotents(D);

I do not know which algorithm is faster.

The same things work over a finite field; actually, some more advanced functions are currently only available over finite fields. So we do the same thing as above again, not over the rationals but over $\mathbb{F}_5$.

> M := CuspidalSubspace(ModularSymbols(229,2,GF(5),1));

> L := [HeckeOperator(M,n) : n in [1..50]];

> A := MatrixAlgebra(L);

> D := Decomposition(A);

> #D;

> A3 := BaseChange(A,D[3]);

> Dimension(A3);

Just out of interest, let's have a look at a basis of ***A3***.

> Basis(A3);

In many cases one would like to store an Artin algebra. Storing it as a matrix algebra can take a lot of memory. The package provides functions for transforming an Artin algebra into an affine algebra, as well as into a tuple describing an affine algebra up to isomorphism. These functions allow to save an Artin algebra in a file by saving the tuple. However, the tuples of isomorphic algebras can be different. All this works as follows.

> AffineAlgebraTup(A3);

This is the tuple attached to the algebra ***A3*** above. See the precise description below for the meaning of the entries of the tuple. We see that the algebra is just a field. Consequently

> AffineAlgebra(A3);

really only is a field. Actually all the seven are:

> [AffineAlgebraTup(BaseChange(A,d)): d in D];

Note that this stops being the case for the corresponding Hecke algebras in characteristic 2. I.e. we do the same as above again, but now over $\mathbb{F}_2$.

> M := CuspidalSubspace(ModularSymbols(229,2,GF(2),1));

> L := [HeckeOperator(M,n) : n in [1..50]];

> A := MatrixAlgebra(L);

> D := Decomposition(A);

> [AffineAlgebraTup(BaseChange(A,d)): d in D];

We can also compute the Gorenstein defect of each factor:

> [GorensteinDefect(BaseChange(A,d)): d in D];

Everything is Gorenstein. Also this may stop if we pass to a different level.

> M := CuspidalSubspace(ModularSymbols(431,2,GF(2),1));

> L := [HeckeOperator(M,n) : n in [1..50]];

> A := MatrixAlgebra(L);

> D := Decomposition(A);

> [AffineAlgebraTup(BaseChange(A,d)): d in D];

> [GorensteinDefect(BaseChange(A,d)): d in D];

That the relation sets are always empty only means that the relations ideal is J^{n+1} (see the description below). Let's have a closer look at the first factor.

> A1 := BaseChange(A,D[1]);

> Basis(A1);

The matrix representation of the algebra is not very nice. We can do better:

> A1T := CommonLowerTriangular(A1);
> Basis(A1T);
Now it's looking much nicer. Note that for the common lower triangular form, the algebra must be defined over its residue field; this was the case. If not, it can be forced; this we explain using the second factor.
> A2 := BaseChange(A,D[2]);
> Basis(A2);
The basis is looking horrible. But, if we change the algebra to the residue field (and take the first of the conjugate factors), it is looking much better:
> B2 := ChangeToResidueField(A2)[1];
> Basis(B2);

2 Mathematical description

The mathematics behind the functions of this package is basic commutative algebra. On this subject many excellent text books exist, to which we refer the reader. Quite a detailed description can also be found in Chapter 2 of the author's lecture notes [4].

3 Functions

3.1 Affine algebras

Let A be a local Artin algebra over a finite field k with maximal ideal $\mathfrak{m}$. The residue field $A/\mathfrak{m}$ is a finite extension of k. By base changing to K and taking one of the conjugate local factors, we assume now that $k = K$. The *embedding dimension* e is the k-dimension of $\mathfrak{m}/\mathfrak{m}^2$. By Nakayama's lemma, this is the minimal number of generators for $\mathfrak{m}$. The name comes from the fact that there is a surjection

$$\pi : k[x_1, \ldots, x_e] \twoheadrightarrow A.$$

Its kernel is called the *relations ideal*. By the *nilpotency order* we mean the maximal integer n such that m^n is not the zero ideal. (As the algebra is local and Artin, its maximal ideal is nilpotent.) We now know that the ideal

$$J^{n+1} \text{ with } J := (x_1, \ldots, x_e)$$

is in the kernel of π. So, in order to store π, we only need to store the kernel R of the linear map between two finite dimensional K-vector spaces

$$\pi_1 : K[x_1, \ldots, x_e]/J^{n+1} \twoheadrightarrow A.$$

From the tuple $< k, e, n, R >$ the algebra can be recreated (up to isomorphism). Let us point out, however, that from the tuple it is not obvious whether two algebras are isomorphic. That would have to be tested after recreating the algebras.
The following functions can be used in order to store such Artin algebras in a way that does not use much memory, but retains the algebra up to isomorphism.

intrinsic AffineAlgebra (A :: AlgMat : try_minimal := true) -> RngMPolRes
intrinsic AffineAlgebra (A :: AlgAss : try_minimal := true) -> RngMPolRes
This function turns the local algebra ***A*** into an affine algebra over its residue field. In fact, the algebra is first base changed to its residue field, then for one of the conjugate local factors an affine presentation is computed. If the option ***try_minimal*** is true, the number of relations will in general be smaller, but the computation time may be longer.

intrinsic AffineAlgebraTup (A :: AlgMat : try_minimal := true) -> Tup
intrinsic AffineAlgebraTup (A :: AlgAss : try_minimal := true) -> Tup
Given a local algebra ***A***, this function returns a tuple ***<k,e,n,R>***, consisting of the residue field ***k*** of ***A***, the embedding dimension ***e***, the nilpotency order ***n*** and relations ***R***. From these data, an affine algebra can be recreated which is isomorphic to one of the local factors of A base changed to its residue field. If the option ***try_minimal*** is true, the number of relations will in general be smaller, but the computation time may be longer.

intrinsic AffineAlgebra (form :: Rec) -> RngMPolRes
Given a modular form record, this function returns the corresponding Hecke algebra as an affine algebra.

intrinsic AffineAlgebra (A :: Tup) -> RngMPolRes
This function turns a tuple ***<k,e,n,R>***, consisting of a field ***k***, two integers ***e***, ***n*** (the embedding dimension and the nilpotency order) and relations ***R***, into an affine algebra.

3.2 Matrix algebra functions

intrinsic MatrixAlgebra (L :: SeqEnum) -> AlgMat
Given a list of matrices ***L***, this function returns the matrix algebra generated by the members of ***L***.

intrinsic RegularRepresentation (A :: AlgMat) -> AlgMat
This function computes the regular representation of the commutative matrix algebra ***A***.

intrinsic CommonLowerTriangular (A :: AlgMat) -> AlgMat
Given a local commutative matrix algebra ***A***, this function returns an isomorphic matrix algebra whose matrices are all lower triangular, provided that ***A*** is defined over its residue field.

Base change

intrinsic BaseChange (S :: Tup, T :: Tup) -> Tup
This function computes the composition of the base change matrices ***T = <C,D>***, followed by those in ***S = <E,F>***.

intrinsic BaseChange (M :: Mtrx, T :: Tup) -> Mtrx
Given a matrix ***M*** and a tuple ***T = <C,D>*** of base change matrices (for a subspace), computes the matrix of ***M*** with respect to the basis corresponding to ***T***.

intrinsic BaseChange (M :: AlgMat, T :: Tup) -> AlgMat
Given a matrix algebra ***M*** and a tuple ***T = <C,D>*** of base change matrices (for a subspace), computes the matrix algebra of ***M*** with respect to the basis corresponding to ***T***.

Decomposition over base field

intrinsic Decomposition (M :: Mtrx) -> Tup
Given a matrix ***M***, this function computes a decomposition of the standard vector space such that the minimal polynomial of ***M*** is a prime power on each summand. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic Decomposition (L :: SeqEnum) -> Tup
Given a sequence ***L*** of commuting matrices, this function computes a decomposition of the standard vector space such that the minimal polynomial of each ***l*** in ***L*** is a prime power on each summand. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic Decomposition (A :: AlgMat) -> Tup
Given a commutative matrix algebra ***A***, this function computes a decomposition of the standard vector space such that each the minimal polynomial of each ***a*** in ***A*** is a prime power on each summand. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic AlgebraDecomposition (A :: AlgMat) -> SeqEnum
Given a matrix algebra ***A*** over a finite field ***k***, returns a list of all local factors of ***A***.

Decomposition over residue field

intrinsic DecompositionOverResidueField (M :: Mtrx : DegBound := 0) -> Tup
Given a matrix ***M***, this function computes a decomposition of the standard vector space such that ***M*** acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic DecompositionUpToConjugation (M :: Mtrx : DegBound := 0) -> Tup

Given a matrix $\boldsymbol{M}$, this function computes a decomposition of the standard vector space such that $\boldsymbol{M}$ acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands. Summands conjugate under the absolute Galois group only appear once.

intrinsic DecompositionOverResidueField (L :: SeqEnum : DegBound := 0) -> Tup

Given a sequence $\boldsymbol{L}$ of commuting matrices, computes a decomposition of the standard vector space such that each $\boldsymbol{l}$ in $\boldsymbol{L}$ acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic DecompositionOverResidueField (A :: AlgMat : DegBound := 0) -> Tup

Given a commutative matrix algebra $\boldsymbol{A}$, this function computes a decomposition of the standard vector space such that each $\boldsymbol{a}$ in $\boldsymbol{A}$ acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands.

intrinsic DecompositionUpToConjugation (L :: SeqEnum : DegBound := 0) -> Tup

Given a sequence $\boldsymbol{L}$ of commuting matrices, this function computes a decomposition of the standard vector space such that each $\boldsymbol{l}$ in $\boldsymbol{L}$ acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands. Summands conjugate under the absolute Galois group only appear once.

intrinsic DecompositionUpToConjugation (A :: AlgMat : DegBound := 0) -> Tup

Given a commutative matrix algebra $\boldsymbol{A}$, this function computes a decomposition of the standard vector space such that each $\boldsymbol{a}$ in $\boldsymbol{A}$ acts as multiplication by a scalar on each summand. For this, each local factor of the algebra is base changed to its residue field. The output is a tuple consisting of base change tuples ***<C,D>*** corresponding to the summands. Summands conjugate under the absolute Galois group only appear once.

intrinsic AlgebraDecompositionUpToConjugation (A :: AlgMat : DegBound := 0) -> SeqEnum
intrinsic ChangeToResidueField (A :: AlgMat : DegBound := 0) -> SeqEnum

Given a matrix algebra $\boldsymbol{A}$ over a finite field $\boldsymbol{k}$, these identical functions return a list consisting for each local factor $\boldsymbol{B}$ of $\boldsymbol{A}$ of one local factor of $\boldsymbol{B}$ tensor $\boldsymbol{K}$ where $\boldsymbol{K}$ is the residue field of $\boldsymbol{B}$.

intrinsic AlgebraDecompositionOverResidueField (A :: AlgMat : DegBound := 0) -> SeqEnum

Given a matrix algebra $\boldsymbol{A}$ over a finite field $\boldsymbol{k}$, this function returns a list of all local factors of $\boldsymbol{A}$ after base change to their residue fields.

Idempotents

intrinsic CompleteOrthogonalIdempotents (L :: SeqEnum) -> SeqEnum

For a list $\boldsymbol{L}$ of commuting algebra elements over a field, this function returns a compelete set of orthogonal idempotents for all the elements in $\boldsymbol{L}$. Note that if $\boldsymbol{L}$ forms a basis of an algebra, then the output is a complete set of orthogonal idempotents of this algebra. If, however, $\boldsymbol{L}$ only forms a generating set, then it is not guaranteed that the output is a complete set.

intrinsic Idempotents (D :: Tup) -> SeqEnum, Tup

Given a decomposition $\boldsymbol{D}$, this function calculates a list of the corresponding idempotents as matrices. The function also returns base change matrices describing the algebra decomposition in terms of block matrices.

Localisations

intrinsic Localisations (L :: SeqEnum) -> Tup, Tup
intrinsic Localisations (A :: AlgMat) -> Tup, Tup

Given a list $\boldsymbol{L}$ of commuting matrices or a commutative Artin matrix algebra $\boldsymbol{A}$, this function computes two tuples $\boldsymbol{C}$, $\boldsymbol{D}$, where $\boldsymbol{C}$ contains a tuple consisting of the localisations of $\boldsymbol{A}$, respectively of the matrix algebra generated by $\boldsymbol{L}$, and $\boldsymbol{D}$ consists of the corresponding base change tuples. The base field is supposed to be a finite field for technical reasons (the MAGMA function ***MaximalIdeals*** is used). Alternatively, and more generally, the function ***Decomposition*** can be used. I do not know which one is faster.

intrinsic Localisations (A :: AlgAss) -> SeqEnum
This function returns a list of all localisations of the Artin algebra A which is assumed to be commutative. The output is a list of associative algebras. This also only works over finite fields; for the same reasons as above.

3.3 Gorenstein defect

Let A be a local Artin algebra over a finite field with unique maximal ideal $\mathfrak{m}$. That the base field is a finite field is only due to technical reasons (the MAGMA function ***MaximalIdeals*** is used). We define the *Gorenstein defect* of A to be $(\dim_{A/\mathfrak{m}} A[\mathfrak{m}]) - 1$, which is equal to the number of A-module generators of the annihilator of the maximal ideal minus one. The algebra is said to be *Gorenstein* if its Gorenstein is equal to 0.

intrinsic GorensteinDefect (A :: RngMPolRes) -> RngIntElt
intrinsic GorensteinDefect (A :: AlgAss) -> RngIntElt
intrinsic GorensteinDefect (A :: AlgMat) -> RngIntElt
These functions return the Gorenstein defect of the local commutative algebra ***A***.

intrinsic IsGorenstein (M :: RngMPolRes) -> BoolElt
intrinsic IsGorenstein (M :: AlgAss) -> BoolElt
intrinsic IsGorenstein (M :: AlgMat) -> BoolElt
These functions test whether the commutative local algebra ***M*** is Gorenstein.

References

[1] W. Bosma, J. J. Cannon, C. Playoust. *The Magma Algebra System I: The User Language.* J. Symbolic Comput. **24** (1997), pp. 235-265

[2] S. J. Edixhoven. *Comparison of integral structures on spaces of modular forms of weight two, and computation of spaces of forms mod* 2 *of weight* 1. Journal of the Inst. of Math. Jussieu (2006) 5(1), 1-34.

[3] L.J.P. Kilford, G. Wiese. *On the failure of the Gorenstein property for Hecke algebras of prime weight.* Experimental Mathematics 17(1), 2008, 37-52.

[4] G. Wiese. *Computational Arithmetic of Modular Forms.* Lecture Notes from a course at Universität Duisburg-Essen. Available from `http://maths.pratum.net/`.

[5] G. Wiese. *Computing Hecke algebras of weight* 1 *in MAGMA.* Appendix B of [2].

[6] G. Wiese. *Modular Forms of Weight One over Finite Fields.* PhD thesis, Universiteit Leiden, 2005. Available from the author's webpage.

[7] G. Wiese. *On the faithfulness of parabolic cohomology as a Hecke module over a finite field.* Journal für die reine und angewandte Mathematik 606 (2007), 79-103.