Manual of the MAGMA package HeckeAlgebra

Gabor Wiese *

8th August 2008

Abstract

This is a short manual for the MAGMA [1] package HeckeAlgebra. It is designed for the computation of Hecke algebras of modular forms over finite fields of characteristic p in the following cases:

- all eigenforms for a given level N, weight k and Dirichlet character $\epsilon : (\mathbb{Z}/N\mathbb{Z})^\times \to \overline{\mathbb{F}}_p^\times$,
- dihedral modular forms coming from the Hilbert class field of a quadratic number field,
- icosahedral modular forms (currently only in characteristic $p = 2$) given by an A_5-polynomial.

The package needs the author's MAGMA package ArtinAlgebras. Both packages can be downloaded from the author's webpage.

The package was originally written in 2006 and has only been slightly updated since. It was used for the computations in [2]. The author would like to thank Lloyd Kilford for very helpful suggestions.

Contents

1 Installation and Example

Note that you need the packages ArtinAlgebras and HeckeAlgebra. They can be downloaded from the author's webpage. For installation, just unpack the tar-files.
Suppose that PATH1 contains ArtinAlgebras.spec and that PATH2 contains HeckeAlgebra.spec. Then we attach the packages using

> AttachSpec("PATH1/ArtinAlgebras.spec");

> AttachSpec("PATH2/HeckeAlgebra.spec");

The following example explains the main functions of the package. We want the package to be silent, so we put

> SetVerbose ("HeckeAlgebra",0);

If we would like more information on the computations being performed, we should have put the value ***1***. Since we want to store the data to be computed in a file, we already create the file.

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

> my_file := "datafile";

> CreateStorageFile(my_file);

Next, we would like to compute the Hecke algebras of the dihedral eigenforms of level 2039 over extensions of $\mathbb{F}_2$. First, we create a list of such forms.

> dih := DihedralForms(2039 : ListOfPrimes := [2], completely_split := false);

Now, we compute the corresponding Hecke algebras, print part of the computed data in a human readable format, and finally save the data to our file.

> for f in dih do

for> ha := HeckeAlgebras(f);

for> HeckeAlgebraPrint1(ha);

for> StoreData(my_file, ha);

for> end for;

The function ***DihedralForms*** also allows to compute only representations that are completely split in the characteristic. The default is ***completely_split := true***. The following example illustrates this.

> dih1 := DihedralForms (431 : bound := 20);

> for f in dih1 do

for> ha := HeckeAlgebras(f);

for> HeckeAlgebraPrint1(ha);

for> StoreData(my_file, ha);

for> end for;

One can also compute icosahedral modular forms over extensions of $\mathbb{F}_2$, starting from an integer polynomial with Galois group A_5, as follows.

> R<x> := PolynomialRing(Integers());

> pol := x^5-x^4-780*x^3-1795*x^2+3106*x+344;

> f := A5Form(pol);

With this kind of icosahedral examples one has to pay attention to the conductor, as it can be huge. This polynomial has prime conductor. But, conductors need not be square-free, in general.

> print Modulus(f`Character);

It's 1951, so it's reasonable. Do the computation.

> ha := HeckeAlgebras(f);

> HeckeAlgebraPrint1(ha);

There are two forms, which is okay, since they come from a weight one form in two different ways and this case is not exceptional. We now save them, as always.

> StoreData(my_file, ha);

It is also possible to compute all forms at a given character and weight.

> eps := DirichletGroup(229,GF(2)).1;

> ha := HeckeAlgebras(eps,2);

> HeckeAlgebraPrint1(ha);

> StoreData(my_file,ha);

Next, we illustrate how one reloads what has been saved. One would like to type: ***load my_file;*** But that does not work. One has to do it as follows.

> load "datafile";

> mf := RecoverData(LoadIn,LoadInRel);

Now, ***mf*** contains a list of all algebra data computed before. There's a rather concise printing function, displaying part of the information.

> HeckeAlgebraPrint(mf);

One can also create a LaTeX longtable. The entries can be chosen in quite a flexible way, but there's also a standard choice of entries given by ***StandardLatexList()***, which is used in our example.

> HeckeAlgebraLaTeX(mf,StandardLaTeXList(),"table.tex");

A short LaTeX file displaying the table is the following:

```
\documentclass[11pt]{article}
\begin{document}
\input{table.tex}
\end{document}
```

The table we created is this one:

Level	Wt	Char	ResD	Dim	GorDef	EmbDim	NilO	largest	HB	Gp
2039	2	2	2	1	0	0	0	5	340	D_3
2039	2	2	2	6	2	3	2	7	340	D_5
2039	2	2	6	1	0	0	0	5	340	D_9
2039	2	2	4	1	0	0	0	5	340	D_{15}
2039	2	2	4	1	0	0	0	5	340	D_{15}
2039	2	2	12	1	0	0	0	5	340	D_{45}
2039	2	2	12	1	0	0	0	5	340	D_{45}
431	2	2	1	4	2	3	1	13	72	D_3
431	11	11	3	4	2	3	1	11	396	D_7
1951	2	2	4	3	0	1	2	5	326	A_5
1951	2	2	4	6	0	2	3	5	326	A_5
229	2	2	1	1	0	0	0	37	39	
229	2	2	2	2	0	1	1	37	39	
229	2	2	1	4	0	1	3	37	39	
229	2	2	5	2	0	1	1	37	39	

In the examples of level 229 the image of the Galois representations as an abstract group is not known. That is due to the fact that we created these examples without specifying the Galois representation in advance.
It is possible to compute arbitrary Hecke operators on the local Hecke factors generated by ***HeckeAlgebras(·)***, as the following example illustrates.
> A,B,M,C := HeckeAlgebras(DirichletGroup(253,GF(2)).1,2 : over_residue_field := true);
Suppose that we want to know the Hecke operator T_{17} on the 4th local factor.
> i := 4;
> T := BaseChange(HeckeOperator(M,17),C[i]);
The coefficients are the eigenvalues (only one):
> Eigenvalues(T);
Let us remember the eigenvalue.
> e := SetToSequence(Eigenvalues(T))[1][1];
In order to illustrate the option ***over_residue_field***, we also compute the following:
> A1,B1,M1,C1 := HeckeAlgebras(DirichletGroup(253,GF(2)).1,2 : over_residue_field := false);
> T1 := BaseChange(HeckeOperator(M1,17),C1[i]);
> Eigenvalues(T1);
The base field is strictly smaller than the residue field in this example and the operator ***T1*** cannot be diagonalised over the base field. We check, that ***e*** is nevertheless a zero of the minimal polynomial of ***T1***.
> Evaluate(MinimalPolynomial(T1),e);

The precise usage of the package is described in the following sections.

2 Mathematical description

For a description of the algorithms and the background we refer to the paper [2]. Some more details can be found in [3]. Moreover, we include some mathematical background in the description of the functions in the following section.

3 Functions

3.1 The modular form format

In the package, modular forms are often represented by the following record.
ModularFormFormat := recformat <

Character **: GrpDrchElt,**
Weight **: RngIntElt,**
CoefficientFunction **: Map,**
ImageName **: MonStgElt,**
Polynomial **: RngUPolElt**

>;

The fields ***Character*** and ***Weight*** have the obvious meaning. Sometimes, the image of the associated Galois representation is known as an abstract group. Then that name is recorded in ***ImageName***, e.g. ***A_5*** or ***D_3***. In some cases, a polynomial is known whose splitting field is the number field cut out by the Galois representation. Then the polynomial is stored in ***Polynomial***. The cases in which polynomials are known are usually icosahedral ones. The ***CoefficientFunction*** is a function from the integers to a polynomial ring. For all primes l different from the characteristic and not dividing the level of the modular form (i.e. the modulus of the ***Character***), the coefficient function should return the minimal polynomial of the l-th coefficient in the q-expansion of the modular form in question.

3.2 Dihedral modular forms

Eigenforms whose associated Galois representation take dihedral groups as images provide an important source of examples, in many contexts. These eigenforms are called *dihedral*. The big advantage is that their Galois representation, and hence their q-coefficients, can be computed using class field theory. That enables one to exhibit Galois representations in the context of modular forms with certain number theoretic properties. The property for which these functions were initially created is that the representations should be unramified in the characteristic, say p, and that p is completely split in the number field cut out by the representation.

Any dihedral representation in our context arises by induction of a character of a quadratic field. The Dirichlet character of the associated modular form is the Legendre symbol of the quadratic field.

intrinsic GetLegendre (N :: RngIntElt, K :: FldFin) -> GrpDrchElt
For an odd integer ***N***, this function returns the element of ***DirichletGroup(Abs(N),K)*** (with ***K*** a finite field of characteristic different from 2) which corresponds to the Legendre symbol $p \mapsto \left(\frac{\pm N}{p}\right)$. The sign in front of ***N*** is chosen so that the number is congruent to 1 mod 4.

intrinsic DihedralForms (N :: RngIntElt :
bound := 100, ListOfPrimes := [], completely_split := true) -> Rec

This function computes all modular forms over a finite field of characteristic p that come from dihedral representations which arise from the quadratic field $K = \mathbb{Q}(\sqrt{\pm N})$ by induction of an unramified character of K. The sign in front of ***N*** is chosen so that the number is congruent to 1 mod 4. If the option ***completely_split*** is set, only those representations are returned which are completely split at p. If the option ***ListOfPrimes*** is assigned a non-empty list of primes, only those primes are considered as the characteristic. If it is the empty set, all primes up to the ***bound*** are taken into consideration.

3.3 Icosahedral modular forms

Eigenforms whose attached Galois representation takes the group A_5 as projective image are called *icosahedral*. Since extensive tables of A_5-extensions of the rationals are available, one disposes of icoshaedral Galois representations which one knows very well. That allows one to test certain conjectures concerning modular forms on icosahedral ones.

We note the isomorphism $A_5 \cong \mathrm{SL}_2(\mathbb{F}_4)$. Thus, A_5-extenstions of the rationals give rise to icosahedral Galois representations in characteristic 2 which (should) come from modular forms mod 2. It would also be possible to use other primes, but this has not been implemented.

intrinsic A5Form (f :: RngUPolElt) -> Rec
Returns the icosahedral form in characteristic 2 and weight 2 of smallest predicted level corresponding to the polynomial ***f***. No checks about ***f*** are performed.

3.4 The Hecke algebra format

The data concerning the Hecke algebra of one eigenform that is computed by the function ***HeckeAlgebras*** is a record of the following form.

AlgebraData := recformat <

Level	***: RngIntElt,***
Weight	***: RngIntElt,***
Characteristic	***: RngIntElt,***
BaseFieldDegree	***: RngIntElt,***
CharacterOrder	***: RngIntElt,***
CharacterConductor	***: RngIntElt,***
CharacterIndex	***: RngIntElt,***
AlgebraFieldDegree	***: RngIntElt,***
ResidueDegree	***: RngIntElt,***
Dimension	***: RngIntElt,***
GorensteinDefect	***: RngIntElt,***
EmbeddingDimension	***: RngIntElt,***
NilpotencyOrder	***: RngIntElt,***
Relations	***: Tup,***
NumberGenUsed	***: RngIntElt,***
ImageName	***: MonStgElt,***
Polynomial	***: RngUPolElt***

>;

Level and ***Weight*** have the obvious meaning. Let K be the base field for the space of modular symbols used. Then ***Characteristic*** is the characteristic of K and ***BaseFieldDegree*** is the degree of K over its prime field. The entries ***CharacterOrder***, ***CharacterConductor*** and ***CharacterIndex*** concern the Dirichlet character for which the modular symbols have been computed. The latter field is the index of the character in ***Elements(DirichletGroup(·))***. Note that that might change between different versions of Magma. The fields ***ResidueDegree*** (over the prime field), ***Dimension*** and ***GorensteinDefect*** have their obvious meaning for the Hecke algebra in question. The tuple

<AlgebraFieldDegree, EmbeddingDimension, NilpotencyOrder, Relations>

are the data from which ***AffineAlgebra*** can recreate the Hecke algebra up to isomorphism. ***NumberGenUsed*** indicates the number of generators used by the package for the computation of the Hecke algebra. This number is usually much smaller than the Sturm bound. ***ImageName*** and ***Polynomial*** have the same meaning as in the record ***ModularFormFormat***.

3.5 Hecke algebras

intrinsic HeckeAlgebras (eps :: GrpDrchElt, weight :: RngIntElt :
first_test := 3, test_interval := 1, when_test_p := 3, dimension_factor := 2,
ms_space := 0, cuspidal := true, DegreeBound := 0, OperatorList := [],
ms_space := 0, over_residue_field := true, force_local := false,
expected_forms := 0 ms_space := 0,
) -> SeqEnum, SeqEnum, ModSym, Tup, Tup
intrinsic HeckeAlgebras (t :: Rec :
first_test := 3, test_interval := 1, when_test_p := 3, dimension_factor := 2,
ms_space := 0, cuspidal := true, DegreeBound := 0, OperatorList := [],
ms_space := 0, over_residue_field := true, force_local := false,
expected_forms := 0 ms_space := 0,
) -> SeqEnum, SeqEnum, ModSym, Tup, Tup

These functions compute all local Hecke algebras (up to Galois conjugacy) in the specified ***weight*** for the given

Dirichlet character ***eps***, respectively those corresponding to the modular form ***t*** given by a record of type ***ModularFormFormat***. The functions return 5 values ***A,B,C,D,E***. ***A*** contains a list of records of type ***AlgebraData*** describing the local Hecke algebra factors. ***B*** is a list containing the local Hecke algebra factors as matrix algebras. ***C*** is the space of modular symbols used in the computations. ***D*** is a tuple containing the base change tuples describing the local Hecke factors. Its knowledge is necessary in order to compute matrices representing Hecke operators in the local factor. Finally, ***E*** contains a tuple consisting of all computed Hecke operators for each local factor of the Hecke algebra.

The usage in practice is described in the example at the beginning of this note. We now explaint the different options in detail.

The modular symbols space to be used in the computations can be determined as follows. The option ***ms_space*** can be set to the values 1 (the plus-space), -1 (the minus-space) and 0 (the full space). Whether the restriction to the cuspidal subspace is taken, is determined by ***cuspidal***. It is not necessary to pass to the cuspidal subspace, for example, if a cusp is given by a coefficient function (see the description of the record ***ModularFormFormat***).

In some cases, a list of Hecke operators on the modular symbols space in question may already have been computed. In order to prevent Magma from redoing their computations, they may be passed on to the function using the option ***OperatorList***.

Often, one wants to compute the local Hecke algebra of a modular form whose degree of the coefficient field over its prime field is known, e.g. in the case of an icosahedral form in characteristic 2 for the trivial Dirichlet character the coefficient field is $\mathbb{F}_4$. By assigning a positive value to the option ***DegreeBound*** the function will automatically discard any systems of eigenvalues beyond that bound, which speeds up the computations. One must be a bit careful with this option, as there may be cases when the bound may not be respected at "bad primes". Hence, one better adds 2 to the degree of the coefficient field, e.g. one chooses ***DegreeBound := 4*** in the icosahedral example just described. If no system of eigenvalues should be discarded for degree reasons, one must set ***DegreeBound := 0***.

The options ***dimension_factor***, ***first_test***, ***test_interval***, ***when_test_p***, ***force_local*** and ***expected_forms*** concern the stop criterion. Theoretically, the Sturm bound (see ***HeckeBound***) tells us up to which bound Hecke operators must be computed in order to be sure that they generate the whole Hecke algebra. In practice, however, the algorithm can often determine itself when enough Hecke operators have been computed to generate the algebra. That number is usually much smaller than the Sturm bound. The stop criterion is the following. Let M be the modular symbols space used and S the set of Hecke operators computed so far. Then $M = \bigoplus_{i=1}^{r} M_i$ (for some r) such that each M_i is respected by the Hecke operators and the minimal polynomial of each $T \in S$ restricted to M_i is a prime power (i.e. each M_i is a primary space for the action of the algebra generated by all elements of S). Let A_i be the algebra generated by $T|_{M_i}$ for all $T \in S$. One knows (in many cases, and in all cases of interest) that A_i is equal to a direct product of local Hecke algebras if one has the equality

$$f \times \dim(A_i) = \text{ degree of } M_i.$$

Here, f is given by ***dimension_factor*** and should be 1 if the plus-space or the minus space of modular symbols are used, and 2 otherwise. The correct assignment of ***dimension_factor*** must be made by hand, whence experimentations are possible. If the stop criterion is not reached, the algorithm terminates at the Hecke bound.

It may happen that when the stop criterion is reached, one A_i is isomorphic to a proper direct product of local Hecke algebras. If in that case the option ***force_local*** is ***true***, the computation of Hecke operators is continued until each A_i is isomorphic to a single Hecke factor. If ***force_local*** is ***false***, then a fast localisation algorithm is applied to each A_i. Hence, the cases in which one may wish ***force_local*** to be ***true*** are rare.

In many cases of interest the Hecke operator T_p with p the characteristic is needed in order to generate the whole Hecke algebra. The option ***when_test_p*** tells the algorithm at which step to compute T_p. It is very advisable to chose a small number. In practice, the stop criterion is reached after very few steps, e.g. 5 steps, when T_p is computed early. Otherwise, the algorithm often has to continue until T_p is computed, although most of the operators before did not change the generated algebra.

The option ***first_test*** tells the algorithm at which step the first test for the stop criterion is to be performed. The next test is then carried out after ***test_interval*** many steps, and so on. These numbers should be chosen small, too, unless the dimension test takes much time, which is rare, so that wants to perform is less often, meaning that possibly more Hecke operators than necessary are computed (time consuming).

If ***expected_forms := n*** is non-zero, the algorithm only stops when both the stop criterion is reached and exactly ***n*** local algebra factors have been computed.

Finally, ***over_residue_field*** tells the algorithm whether at the end of the computation the local Hecke factors should be base changed to their residue field. If that is done, only one of the conjugate local factors of the base changed algebra is retained.

3.6 Storage functions

The package provides functions to store a list whose elements are records of type ***AlgebraData*** in a file, and to re-read it. The usage of these functions is explained in the example at the beginning of this note.

intrinsic CreateStorageFile (filename :: MonStgElt)
This function prepares the file ***filename*** for storing the data.

intrinsic StoreData (filename :: MonStgElt, forms :: SeqEnum)
This functions appends the list ***forms*** of Hecke algebra data to the file ***filename***. That file must have been created by ***CreateStorageFile***.

intrinsic StoreData (filename :: MonStgElt, form :: Rec)
This function appends the Hecke algebra data ***form*** to the file ***filename***. That file must have been created by ***CreateStorageFile***.

intrinsic RecoverData (LoadIn :: SeqEnum, LoadInRel :: Tup) -> SeqEnum
In order to read Hecke algebra data from file ***filename***, proceed as follows:

> load filename;

> readData := RecoverData(LoadIn,LoadInRel).

Then ***readData*** will contain a list whose elements are records of type ***AlgebraData***.

3.7 Output functions

intrinsic HeckeAlgebraPrint (ha :: SeqEnum)
intrinsic HeckeAlgebraPrint1 (ha :: SeqEnum)
These functions print part of the data stored in the list ***ha*** of records of type ***AlgebraData*** in a human readable format.

intrinsic GetLevel (a :: Rec) -> Any
intrinsic GetWeight (a :: Rec) -> Any
intrinsic GetCharacteristic (a :: Rec) -> Any
intrinsic GetResidueDegree (a :: Rec) -> Any
intrinsic GetDimension (a :: Rec) -> Any
intrinsic GetGorensteinDefect (a :: Rec) -> Any
intrinsic GetEmbeddingDimension (a :: Rec) -> Any
intrinsic GetNilpotencyOrder (a :: Rec) -> Any
intrinsic GetLargestOperator (a :: Rec) -> Any
intrinsic GetHeckeBound (a :: Rec) -> Any
intrinsic GetPolynomial (a :: Rec) -> Any
intrinsic GetImageName (a :: Rec) -> Any
These functions return the property of the record ***a*** of type ***AlgebraData*** specified by the name of the function. If the corresponding attribute is not assigned, the empty string is returned.

intrinsic StandardLaTeXList () -> SeqEnum
This function creates a standard format for writing Hecke algebras to a LaTeX table. That format is a list of tuples ***<f, name>***. Here ***f*** is a function that evaluates a record of type ***AlgebraData*** to some Magma object which is afterwards transformed into a string using ***Sprint***. Examples for ***f*** are the functions ***GetLevel*** etc., which are described above. The ***name*** will appear in the table header.

intrinsic HeckeAlgebraLaTeX (ha :: SeqEnum, which :: SeqEnum, filename :: MonStgElt)
This function creates the LaTeX file ***filename*** containing a longtable consisting of certain properties of the objects

in ***ha*** which are supposed to be records of type ***AlgebraData***. The properties to be written are indicated by the list ***which*** consisting of tuples ***<f, name>***. Here ***f*** is a function that evaluates a record of type ***AlgebraData*** to some Magma object which is afterwards transformed into a string using ***Sprint***. Examples for ***f*** are the functions ***GetLevel*** etc., which are described above. The ***name*** will appear in the table header. For a sample usage, see the example at the beginning of this note.

3.8 Other functions

intrinsic HeckeBound (N :: RngIntElt, k :: RngIntElt) -> RngIntElt
intrinsic HeckeBound (eps :: GrpDrchElt, k :: RngIntElt) -> RngIntElt
These functions compute the Hecke bound for weight ***k*** and level ***N***, respectively Dirichelt character ***eps***. Note that the Hecke bound is also often called Sturm bound.

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] 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.

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