The *MAGMA* package *Hecke1*

Gabor Wiese*

April 13, 2003

Contents

1 Introduction

The *MAGMA* ([1]) package *Hecke1* provides functions for creating a module of the Hecke algebra of Katz modular forms of weight 1 with coefficients in a finite field. It is accompanied by

*Supported by the European Research Training Network Contract HPRN-CT-2000-00120 "Arithmetic Algebraic Geometry". Address: Mathematisch Instituut, Universiteit Leiden, Postbus 9512, 2300 RA Leiden, The Netherlands; `http://www.math.leidenuniv.nl/~gabor/`, e-mail: `gabor@math.leidenuniv.nl`

the package *CommMatAlg* ([4]), which contains some tools for computations with commutative matrix algebras.

The algorithms used are based on [2]. See also [3].

2 What is computed?

INPUT:

- M some space of modular symbols, say for field K, level N, weight p and character $\epsilon : (\mathbb{Z}/N)^* \to K^*$.

  The weight p must be a prime, and K must either be a finite field or a number field. See below, for which values the calculations are known to give the desired result.

COMPUTE:

- C the cuspidal subspace of the space of modular symbols M and D the dimension of C.
- $\text{Bound} = B(p+2)$, where $B = \frac{1}{12} N \prod_{l|N, l \text{ prime}} (1 + \frac{1}{l})$.
- If K is a number field, let $\mathcal{O}_K$ be its ring of integers and $\mathfrak{P}$ a prime ideal above p. Set $\phi : \mathcal{O}_K \twoheadrightarrow \mathcal{O}_K/\mathfrak{P} =: \mathbb{F}$.

  If K is a finite field, set $\mathbb{F}$ to be K and take ϕ to be the identity.
- Let T_i stand for either the matrix in $\mathcal{O}_K^{D \times D}$ representing the i-th integral Hecke operator acting on C (generated by the MAGMA command `IntegralHeckeOperator`, only available if $K = \mathbb{Q}$) or the matrix in $K^{D \times D}$ representing the i-th Hecke operator acting on C (generated by the MAGMA command `HeckeOperator`).

  Call $\overline{T_i}$ the image of T_i under ϕ. If in the case of a number field K the matrix does not lie in $\mathcal{O}_K^{D \times D}$, the zero matrix is used instead and a warning is issued.
- Let $\mathcal{A}$ be the sub-$\mathbb{F}$-vector space of $\mathbb{F}^{D \times D}$ generated by all $\overline{T_i}$ for $1 \leq i \leq \text{Bound}$.
- Let $d, \tilde{D}$ be positive integers and b be a sequence of integers $b_1, \ldots, b_d, b_{d+1}, \ldots, b_{\tilde{D}}$ s.t.:

  (i) $p \mid b_i$ for $1 \leq i \leq d$,

  (ii) $p \nmid b_i$ for $d+1 \leq i \leq \tilde{D}$,

  (iii) $\overline{T_{b_i}}$ for $d+1 \leq i \leq \tilde{D}$ form a basis of the span of $\overline{T_n}$ for all $n \leq \text{Bound}$ with $p \nmid n$,

  (iv) $\overline{T_{b_i}}$ for $1 \leq i \leq \tilde{D}$ form a basis of $\mathcal{A}$.
- The weight 1 Hecke module we want to work with is

$$\boxed{\mathcal{H} = \mathcal{A}/\mathcal{R},}$$

where $\mathcal{R}$ is the sub-$\mathbb{F}$-vector space of $\mathcal{A}$ generated by $\overline{T_{b_i}}$ for $d+1 \leq i \leq \tilde{D}$.

$\mathcal{H}$ has dimension d and comes with a natural basis, namely the one given by the images of $\overline{T_{b_i}}$ for $1 \leq i \leq d$.

- The action of the weight 1 Hecke operators $T_n^{(1)}$ is as follows.

  For $l \neq p$ a prime, $T_l^{(1)}$ acts by the natural action of $\overline{T_l}$ on $\mathcal{H}$.

  The operator $T_p^{(1)}$ acts via the natural action of $\overline{T_p} + \phi(\epsilon(p))F$, where F (the Frobenius) by definition sends $\overline{T_{b_i}}$ to $\overline{T_{b_i/p}}$ for all $1 \leq i \leq d$.

  The operator $T_n^{(1)}$ with n composite is calculated by the usual formulae from the prime operators.

3 Mathematical interpretation

We stick to the preceding notations. Let us assume that K is a number field and that M is the full space (or its cuspidal subspace) of modular symbols of weight p (a prime), level N and character $\epsilon : (\mathbb{Z}/N)^* \to K^*$. In order to obtain a good interpretation, we assume that $N \geq 5$ and that p does not divide N. We can also assume that $\epsilon(-1) = (-1)^p$, as we only have the zero space otherwise.

It is known that, with the choice of Bound as above, $\mathcal{A}$ contains all $\overline{T_i}$ for $i \in \mathbb{N}$.

Via the natural surjection $\psi :< T_i \mid i \leq \mathrm{Bound} > \otimes_{\mathcal{O}_K} \mathbb{F} \twoheadrightarrow \mathcal{A}$, it follows from [3], Prop. 2.3.2, that $\mathcal{H}$ is a module for the Hecke algebra of $S_1(\Gamma_1(N), \phi \circ \epsilon, \mathbb{F})'_{\mathrm{Katz}}$.

If $\epsilon = 1$ is the trivial character and if there exists a prime q dividing N such that $q \equiv 3$ modulo 4, it is proved in Cor. 2.3.3 that

$$\big(< T_i \mid i \leq \mathrm{Bound} > \otimes_{\mathcal{O}_K} \mathbb{F} \big) / < T_i \otimes 1 \mid p \nmid i, i \leq \mathrm{Bound} >$$

is a faithful Hecke module. Hence, if ψ is an isomorphism, $\mathcal{H}$ is also faithful and its dimension equals the dimension of $S_1(\Gamma_1(N), \phi \circ \epsilon, \mathbb{F})'_{\mathrm{Katz}}$. In any case a warning is issued, whenever ψ is not an isomorphism.

If K is a finite field, a similar interpretation should be possible.

4 Documentation of *Hecke1*

The basic function is `HeckeAlgebraWt1`. It is called with a space M of modular symbols as specified above. The function creates a record containing the basic data such as $d, \mathcal{A}, b$ etc. as explained before. Already the first Bound Hecke operators of weight 1 acting on $\mathcal{H}$ are computed and stored. Hence calling this function may take some time and memory.

The created record should not be printed. Instead, the properties can be accessed by various functions, such as `Dimension`, `Field`, `HeckeOperatorWt1`, `HeckeAlgebra` and `HeckePropsToString`. Moreover, a rather primitive database handling is provided. Please see below for a precise documentation of the provided functions.

The package *Hecke1* is accompanied by *CommMatAlg*, which is useful for the study of the Hecke algebra (and is required by some of the functions of *Hecke1*).

4.1 Installation

If `PATH` is the directory in which the files `Hecke1.mg` and `CommMatAlg.mg` are stored, type

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
```

in order to use them. By the command

```
> SetVerbose ("Hecke1",i);
```

you can determine whether *MAGMA* informs you on the calculations in progress (`i = 1`) or not (`i = 0`).

4.2 Creation functions

- `intrinsic HeckeAlgebraWt1 ( ms ::  ModSym :`
  `useIntegral := true ) -> Rec`

  This command computes the essential data for the characteristic p Hecke algebra of modular forms of weight 1 of level and character as specified by the space of modular symbols `ms` given. If `useIntegral` is `true`, then the command `IntegralHeckeOperator` is used (if possible) instead of `HeckeOperator`. Please consult section 2 for the precise functionality.

  The record returned by this function can be used to create the Hecke algebra, Hecke operators and to determine their properties. Note that the calculations can take some time and can consume a considerable amount of memory.

4.3 Properties

- `intrinsic Dimension ( H ::  Rec ) -> RngIntElt`

  Returns the dimension of the Hecke algebra of weight 1 specified by the Hecke algebra data `H`.

- `intrinsic Level ( H ::  Rec ) -> RngIntElt`

  Returns the level of the Hecke algebra of weight 1 specified by the Hecke algebra data `H`.

- `intrinsic Characteristic ( H ::  Rec ) -> RngIntElt`

  Returns the characteristic of the coefficients of the Hecke algebra of weight 1 specified by the Hecke algebra data `H`.

- `intrinsic Field ( H :: Rec ) -> Rng`

  Returns the coefficient field of the Hecke algebra of weight 1 specified by the Hecke algebra data `H`.

- `intrinsic Bound ( H :: Rec ) -> RngIntElt`

  Returns the bound up to which the Hecke operators are to be calculated in order to obtain the Hecke algebra of weight 1 specified by the Hecke algebra data `H`.

- `intrinsic OriginalCharacter ( H :: Rec ) -> GrpDrchElt`

  Returns the Dirichlet character of the space of modular symbols, from which the Hecke algebra data `H` have been calculated.

- `intrinsic Error ( H :: Rec ) -> BoolElt`

  Returns whether an error has occured during one of the calculations of Hecke operators. An error can e.g. be due to the fact that matrices representing Hecke operators in weight p need not have coefficients in a ring of integers of a number field. In such a case, the zero operator is used, and thus the results need not be correct.

- `intrinsic HeckePropsToString ( H :: Rec :`
  `format := "TXT") -> MonStgElt`

  Prints some properties of the Hecke algebra in weight 1 to a string. Currently the formats `"TXT"` (human readable) and `"CR"` (computer readable) are supported.

- `intrinsic HeckePropsToFile ( H :: Rec, file :: MonStgElt :`
  `format := "TXT")`

  Same as `HeckePropsToString`. The output string is appended to the specified file.

- `intrinsic CreateFilePropsCR ( file :: MonStgElt )`

  This function creates a file for storing MAGMA readable properties (for use with `HeckePropsToFile` in format `"CR"`).

4.4 Hecke algebra and operators

- `intrinsic MyHeckeOperator ( C :: ModSym, n :: RngIntElt :`
  `useIntegral := true ) -> Any`

  This command calls `IntegralHeckeOperator` if `useIntegral` is `true` and the base field of `C` is $\mathbb{Q}$, and `HeckeOperator` otherwise.

- `intrinsic HeckeOperatorWt1 ( n :: RngIntElt, H :: Rec :`
  `useIntegral := true ) -> Mtrx`

  Computes the `n`th Hecke operator of weight 1 as element of $\mathrm{End}(\mathcal{H})$ for an internal basis, as specified by the Hecke algebra data `H`. For the option `useIntegral` please see above.

- `intrinsic CalcHeckeOperatorsWt1 ( L :: SeqEnum, ~H :: Rec : useIntegral := true )`

  This function computes the Hecke operators of weight 1 (as elements of $\mathrm{End}(\mathcal{H})$ w.r.t. an internal basis) specified by the list `L` of indices and stores them among the Hecke algebra data `H`. For the option `useIntegral` please see above.

- `intrinsic HeckeOperatorsWt1 ( H :: Rec ) -> SeqEnum`

  This function returns the list of all Hecke operators of weight 1 that are stored in the Hecke algebra data `H`. Note that the operators up to `Bound(H)` are already computed by `HeckeAlgebraWt1`.

- `intrinsic HeckeAlgebra ( H :: Rec ) -> AlgMat`

  Creates the Hecke algebra of weight 1 as a matrix algebra. As generators the operators of weight 1 stored in the Hecke algebra data `H` are used.

4.5 Output functions

The command `HeckeAlgebraWt1` computes the essential data of the Hecke algebra in question, which can be extended using `CalcHeckeOperatorsWt1`. In the present section we describe functions for storing these data. We distinguish three subsets: The *information* contains data such as level, weight, dimension etc. It does not need much memory. The *algebra structure* essentially contains a basis of the weight p Hecke algebra, from which the weight 1 operators can be deduced. Its calculations takes most of the time and much memory. Finally, the *Hecke operators of weight* 1 are treated separately.

The stored data can be loaded using

```
> load "file";
```

- `intrinsic SaveHeckeInfoWt1 ( file :: MonStgElt, name :: MonStgElt, H :: Rec )`

  Saves the basic information on the Hecke algebra of weight 1 specified by `H` into the `file`. Neither the calculated Hecke operators of weight 1 nor the algebra structure are saved. The string `name` will be the name of the identifier of the Hecke algebra after reloading.

- `intrinsic SaveHeckeOperatorsWt1 ( file :: MonStgElt, name :: MonStgElt, H :: Rec )`

  Saves the calculated list of Hecke operators of weight 1. The usage is as above.

- `intrinsic SaveHeckeAlgebraWt1 ( file :: MonStgElt, name :: MonStgElt, H :: Rec )`

  Saves the Hecke algebra structure. The usage is as above.

4.6 Database functions

A rather primitive database handling is provided. As above, the data is divided into *information* (`"info"`), *algebra structure* (`"alg"`) and *Hecke operators of weight 1* (`"ops"`). The `"what"`-option can be set to these values or to `"all"`. These functions only work on Unix/Linux operating systems and use `gzip` for compression.

Before using the database, type

```
> H := [];
```

in each session, as the data of level `N` will be stored in `H[N]`. Hence it is also assumed that the coefficients are in the same field and of the same weight, as the algebras are distinguished by their levels only.

- `intrinsic ExistsInDatabase ( N :: RngIntElt,`
  `DatabasePath :: MonStgElt : what := "all" ) -> BoolElt`

  Returns whether the specified data of level N is stored in the database.

- `intrinsic SaveToDatabase ( H :: Rec,`
  `DatabasePath :: MonStgElt : what := "all" )`

  Saves the specified subset of the Hecke algebra data `H` to the database.

- `intrinsic AddToDatabase ( N :: RngIntElt,`
  `DatabasePath :: MonStgElt : what := "all" )`
  `intrinsic AddToDatabase ( L :: SeqEnum,`
  `DatabasePath :: MonStgElt : what := "all" )`

  Computes the Hecke algebra data of level `N` resp. of all levels in the list `L` for the trivial character over the field $\mathbb{F}_2$ and adds them to the database. The coefficient field of the modular symbols used is $\mathbb{Q}$.

- `intrinsic AccessDatabase ( L :: SeqEnum,`
  `DatabasePath :: MonStgElt : what := "all" ) -> MonStgElt`

  Creates and returns the name of a file, which you `load` in order to obtain the specified data for the levels in the list `L`. See the resp. example below.

- `intrinsic SearchDatabase ( L :: SeqEnum,`
  `DatabasePath :: MonStgElt : what := "info" ) -> MonStgElt`

  Please consult the resp. example below for the usage.

5 Examples

5.1 Getting started

First you have to attach the packages *CommMatAlg* and *Hecke1*, which we assume to be stored in the folder `PATH`.

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
```

Now we create the data of a Hecke algebra.

```
> MS := ModularSymbols (283,2);
> h := HeckeAlgebraWt1 (MS);
```

The essential data for the Hecke algebra of weight 1, level 283 with coefficients in $\mathbb{F}_2$ is calculated and stored in the record `h`.

It is then possible to access information on the algebra:

```
> Dimension(h);
3
> Level(h);
283
> Field(h);
Finite field of size 2
```

5.2 Hecke algebra

In this example we show how the Hecke algebra can be obtained as a matrix algebra.

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
> h := HeckeAlgebraWt1(ModularSymbols(491,2));
```

We can create the Hecke algebra as a matrix algebra as follows:

```
> H := HeckeAlgebra(h); H;
Matrix Algebra of degree 6 with 164 generators over GF(2)
```

Let us study its structure by decomposing it into local factors over $\mathbb{F}_2$.

```
> L,S := LocalDecomposition(H);
> #L;
2
```

So there are 2 local factors. We can generate the matrix algebra corresponding to the first factor like this:

```
> H1 := MatrixAlgebra(H,L[1]); H1;
Matrix Algebra of degree 3 with 26 generators over GF(2)
```

Now we take its maximal ideal.

```
> m1 := MaximalIdeals(H1)[1]; m1;
Matrix Algebra [ideal of H1] of degree 3 and dimension 0 over
GF(2)
```

Since it has dimension 0, we know that `H1` is the field with 8 elements.

Let us now also look at the second factor and take its maximal ideal.

```
> H2 := MatrixAlgebra(H,L[2]); H2;
Matrix Algebra of degree 3 with 26 generators over GF(2)
> m2 := MaximalIdeals(H2)[1]; m2;
Matrix Algebra [ideal of H2] of degree 3 and dimension 2 over
GF(2)
```

It follows that the residue field is $\mathbb{F}_2$. We also see that only the third power of `m2` vanishes:

```
> Dimension(m2*m2); Dimension(m2*m2*m2);
1
0
```

5.3 Writing to a database

In this example we show, how data on Hecke algebras can be stored in a database. As path of the database we will use `"data/"`.

We first want to store just one algebra.

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
> h := HeckeAlgebraWt1(ModularSymbols(143,2));
> SaveToDatabase (h,"data/");
Saving Hecke algebra information...
Saving Hecke algebra structure...
Zipping Hecke algebra...
Saving Hecke operators...
Zipping Hecke operators...
```

Alternatively, we could you use

```
> AddToDatabase(143,"data/");
The required data for the level 143 is already in the
database.  Nothing done.
```

If you want to add several algebras, you can use

```
> AddToDatabase([(2*n+1) :  n in [2..50]],"data/");
```

Options are provided to specify which data are to be stored. If for instance the structure is not required (i.e. no further weight 1 Hecke operators are needed), you can only store the information and all calculated weight 1 Hecke operators as follows:

```
> AddToDatabase([(2*n+1) :  n in [100..110]],"data/" :
  what := "ops");
```

5.4 Reading from a database

We now present an example on accessing the database.

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
```

The algebras will be stored in the list `H`, which has to be created first.

```
> H := [];
```

Now we can load the algebra of level 143 as follows:

```
> AccessDatabase ([143],"data/");
data/Access
> load "data/Access";
Loading "data/Access"
Loading information on Hecke algebra of level 143...
Loading "data/info143"
Unzipping Hecke algebra structure of level 143...
Loading Hecke algebra structure of level 143...
Loading "data/LoadIn"
Unzipping Hecke operators of level 143...
Loading Hecke operators of level 143...
Loading "data/LoadIn"
```

The algebra data are now stored in `H[143]`:

```
> Dimension (H[143]);
4
```

Of course, you can also load several algebras at once:

```
> AccessDatabase ([(2*n+1) :  n in [40..50]],"data/");
data/Access
> load "data/Access";
```

5.5 Searching a database

The following example lists the dimensions of the Hecke algebras of weight 1, for the levels $81, 83, 85, \ldots, 101$.

```
> Attach("PATH/CommMatAlg.mg");
> Attach("PATH/Hecke1.mg");
```

Since the data is loaded into the sequence `H`, it has to be created first.

```
> H := [];
```

Next you need to provide a function, named `SEARCH`, which will be called as `SEARCH (H[N])`, for `N` running through the specified levels. In this example, we return a string containing the level and the dimension.

```
> SEARCH := function (h)
function> return "Level = " * Sprint(Level(h)) * " has
          dimension " * Sprint (Dimension(h));
function> end function;
```

Since we only want to see what we print, we set

```
> SetVerbose ("Hecke1",0);
```

Now the search can be performed as follows. We note that all we need to know on the algebras is contained in the *information*.

```
> SearchDatabase ([(2*n+1) :  n in [46..50]],"data/" :
  what := "info" );
data/Search
> load "data/Search";
Loading "data/info93"
Level = 93 has dimension 2
Loading "data/info95"
Level = 95 has dimension 3
Loading "data/info97"
Level = 97 has dimension 0
Loading "data/info99"
Level = 99 has dimension 0
Loading "data/info101"
Level = 101 has dimension 0
```

References

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

[2] Edixhoven, S. J.: *Comparison of integral structures on spaces of modular forms of weight two, and computation of spaces of forms mod* 2 *of weight* 1.

[3] Wiese, G.: *Computing Hecke algebras of weight* 1 *in* MAGMA, Appendix to [2]

[4] Wiese, G.: *The Magma package CommMatAlg*, documentation and source are available from the author's homepage