# Mixing Data and Computation to explore mathematical data sets: Knowledge to the rescue with LMFDB + SageMath+ PARI/GP+ MitM

John Cremona[1] David Lowry-Duda[1]

University of Warwick

**Abstract.** We describe a use case for intercommunication between LMFDB data and either SageMath or PARI/GP in order to investigate a research problem in pure mathematics.

## 1 Introduction

We describe a use case for intercommunication between LMFDB data and either SageMath [1] or PARI/GP [2]. The LMFDB contains many individual instances of several kinds of *modular forms* (which are the "MF" in "LMFDB"), including *classical modular forms*, *Hilbert modular forms* (HMFs), *Bianchi modular forms* (BMFs) and others. Each of these objects has associated to it an algebraic number field called its *Hecke field*. A mathematically interesting project is to study these Hecke fields for the modular forms in each collection, collecting data about them such as their degree or class number (see the next section for definitions) and investigating how these are distributed. Currently it is not even known how many of the Hecke fields are themselves in the LMFDB's own database of number fields; where they are, the associated invariants can simply be looked up in the LMFDB itself, but in cases where the Hecke field is not itself in the database, further computations using SageMath or PARI/GP would become possible.

Being able to access this LMFDB data directly from within a package with the ability to carry out its own computations with number fields would be a useful application.

## 2 Modular forms and their Hecke fields

We will not here give full definitions of the kinds of modular forms to be considered since even for the simplest ones this is a substantial mathematical theory dating back to the 19th century. See http://www.lmfdb.org/knowledge/show/mf and the links there for brief definitions (http://www.lmfdb.org/knowledge/show/mf.hilbert and http://www.lmfdb.org/knowledge/show/mf.bianchi.bianchimodularforms). We limit ourselves to giving just those characteristics which are necessary for the current application, in the case of classical, Hilbert and Bianchi forms.

### 2.1 Base field and level

Each modular form we are considering has a *base field* $F$ which is an algebraic number field (that is, a finite extension of the rational field $\mathbb{Q}$). For classical modular forms we have $F = \mathbb{Q}$, for Hilbert Modular Forms $F$ is a totally real number field (meaning that every embedding of $f$ into $\mathbb{C}$ has image in $\mathbb{R}$, for example $F = \mathbb{Q}(\sqrt{5})$), and for Bianchi Modular Forms, $F$ is an imaginary quadratic field (for example $F = \mathbb{Q}(\sqrt{-1})$). We will mostly restrict to the case of Hilbert Modular Forms; while strictly speaking this includes classical modular forms (since $F = \mathbb{Q}$ is a totally real field), we follow convention in only applying the term "Hilbert Modular Form" when $F \neq \mathbb{Q}$. Then the *degree* of $F$ is an integer $n \geq 2$.

Our reason for excluding Bianchi Modular Forms from now on is simply that at present the LMFDB has almost no BMFs whose Hecke field is not $\mathbb{Q}$. The case of classical modular forms is mathematically very similar, but the means of storing these forms in the LMFDB is different, they are stored in different collections, which are also under current development, and would require a different interface from the HMFs.

After specifying the base field, we must next give the *level* of the modular form. This is an (integral) ideal $\mathcal{N}$ of the ring of integers $\mathcal{O}_F$ of the base field $F$.

In general, modular forms also have an associated *character*, but we only consider those with trivial character here, both for simplicity and also because at present the LMFDB only has HMFs with trivial character. Similarly, modular forms have a *weight*, and everything we will say would apply (with only minor changes) to forms of arbitrary weight; however, the HMFs currently in the LMFDB all have the same weight (parallel weight 2), so we will not mention the weight further.

### 2.2 Hilbert newforms and their Hecke fields

For a fixed base field $F$, level $\mathcal{N}$ (and fixed weight and character), the HMFs with these parameters form a finite-dimensional vector space. In this space there is a finite set of distinguished elements called *newforms*; the objects actually stored in the LMFDB are these newforms. From now on when we refer to an HMF over some base field $F$, will will usually mean a Hilbert newform.

Each newform $f$ is defined by and characterised by additional data in addition to the base field, level, weight and character. This additional data serves to distinguish different newforms with the same base field and level, and also carries with it information of great significance in the study of these newforms. Each newform has a *Fourier expansion* indexed by elements $\alpha \in \mathcal{O}_F$, with Fourier coefficients $c_\alpha$ satisfying various multiplicative relations, including $c_1 = 1$. The *Hecke field* of $f$ is the field $K_f$ generated over $\mathbb{Q}$ by all these coefficients $c_\alpha$, which is always an algebraic number field (a finite extension of $\mathbb{Q}$). The degree of the Hecke field $K_f$ is called the *dimension* of the newform. For example, to say that the Hecke field $K_f = \mathbb{Q}$ is equivalent to the dimension being equal to 1 and to the statement that all Fourier coefficients $c_\alpha$ lie in $\mathbb{Q}$. Moreover the Fourier

coefficients of every newform are all algebraic integers, so in fact all $c_\alpha \in \mathcal{O}_{K_f}$, and when $K_f = \mathbb{Q}$ even $c_\alpha \in \mathbb{Z}$.

The Hecke field has an alternate definition which explains the name, and is also used to establish some of the properties stated above: on the complex vector space of all HMFs of a given level (and fixed weight and character) there is a commutative algebra of linear operators acting, called the *Hecke Algebra*. The newforms are "eigenforms" for the action of this algebra, normalised by scaling so that $c_1 = 1$, and the coefficients $c_\alpha$ are also the eigenvalues of certain operators in the Hecke algebra. The multiplicative relations mentioned above mean that each newform is characterised by its eigenvalues, or coefficients, indexed by *prime* elements of $\mathcal{O}_F$; moreover, the coefficients $c_\alpha$ are unaffected by scaling $\alpha$ by units in $\mathcal{O}_F^*$, so that $f$ is completely determined by Hecke eigenvalues $c_\mathfrak{p}$ for $\mathfrak{p}$ running over the prime ideals of $\mathcal{O}_F$.

To summarize this discussion:

1. Every Hilbert newform $f$ has a base field $F$ and a level $\mathcal{N}$ which is an ideal in $\mathcal{O}_F$.
2. To $f$ is also associated a number field $K_f$ called its Hecke field, containing all its Fourier coefficients (or Hecke eigenvalues).
3. The newform $f$ is uniquely determined by the collection $\{c_\mathfrak{p}\}$ indexed by prime ideals of the base field $F$, where each $c_\mathfrak{p} \in \mathcal{O}_{K_f}$.
4. For fixed $F$ and $\mathcal{N}$ there are only finitely many newforms, which may be distinguished by their coefficients $c_\mathfrak{p}$ for a finite collection of prime ideals $\mathfrak{p}$ of $F$.

## 3 Relevant LMFDB databases and collections

At present the LMFDB data is stored in a `Mongo` database. Under `Mongo` terminology the entire dataset is subdivided first into "databases" and each database consists of a number of "collections". The records in each collection can in principle have widely varying content, but in practice there is little variation and we may consider each record to have the same structure with a fixed set of keys each mapping to a value of a predefined type.

All the Hilbert modular form data in the LMFDB is stored in a database called `hmfs`. Within this database there are three collections: `hmfs.fields`, `hmfs.forms`, and `hmfs.hecke`.

### 3.1 The `hmfs.fields` collection

The collection `hmfs.fields` contains an entry for each base field (denoted $F$ above). An inventory for the records in this collection may be seen at `http://www.lmfdb.org/inventory/hmfs/fields/`. Most relevant for the current discussion are the following keys:

- `degree` (integer): the degree of $F$ over $\mathbb{Q}$ (as an integer, for example 2).

– `label` (string): the LMFDB label of the field, for example `2.2.5.1`. These labels allow cross-referencing to the LMFDB fields database, where most data about the base field is stored.

All fields which occur as base fields for HMFs in the database are themselves in the LMFDB's number field collection. The other keys in `hmfs.fields` are technical and need not concern us. The purpose of this collection is to provide information used in the display of individual Hilbert newform data on the LMFDB website `www.lmfdb.org`, and the display of some statistical data.

The collection `hmfs.fields` contains 400 entries, of base fields of degrees between 2 and 6.

### 3.2 The `hmfs.forms` collection

The collection `hmfs.forms` contains an entry for each Hilbert newform (denoted $f$ above). An inventory for the records in this collection may be seen at `http://www.lmfdb.org/inventory/hmfs/forms/`. The relevant keys for our purposes are as follows:

– Keys relating to the base field $F$:
  • `deg` (integer): the degree $[F : \mathbb{Q}]$;
  • `field_label` (string): the LMFDB label of $F$.
– Keys relating to the level $\mathcal{N}$:
  • `level_ideal` (list): data from which the level may be constructed, consisting of a list of two integers and one element of $\mathcal{O}_F$;
  • `level_norm` (integer): the norm of the level;
  • `level_label` (string): the standardised label for the level.
– Key to specify the newform among others with the same base field and level: `label_suffix` (string).
– Key giving the degree of the Hecke field: `dimension` (integer).
– Other keys give composite labels; `label` (string) is a concatenation of the base field label, the level label and the suffix and completely identifies this newform.

The collection `hmfs.forms` contains 368356 entries. One example is `http://www.lmfdb.org/ModularForm/GL2/TotallyReal/4.4.16357.1/holomorphic/4.4.16357.1-55.1-c`. Here the base field has label `4.4.16357.1`, the level has label `55.1` and the suffix specifying one particular newform at this level is `c`; the full label is then `4.4.16357.1-55.1-c`. This example has dimension 1, meaning that the Hecke field is $\mathbb{Q}$. Another example, which has dimension 286 (the current maximum) is `http://www.lmfdb.org/ModularForm/GL2/TotallyReal/2.2.296.1/holomorphic/2.2.296.1-29.1-c`.

### 3.3 The `hmfs.hecke` collection

The Fourier coefficients (or Hecke eigenvalues) of the newforms are stored in a separate collection `hmfs.hecke` for technical reasons. In the previous section

we saw that the only information about the Hecke field stored in the main `hmfs.forms` collection is the dimension, which is the degree of the Hecke field. Here we specify the Hecke field itself and store the individual eigenvalues for each newform.

There is one record `hmfs.hecke` for each newform record in `hmfs.forms`, containing the following data.

- `label` (string): the full label of the newform, matching the associated record in `hmfs.forms`;
- `hecke_polynomial` (string): a string representing a polynomial $g(x)$ with rational coefficients in the variable $x$.
- `hecke_eigenvalues` (list): a list of the first several Hecke eigenvalues $c_{\mathfrak{p}}$ of the newform, indexed by the primes $\mathfrak{p}$ of $\mathcal{O}_F$. Each $c_{\mathfrak{p}}$ is stored as a string representing the eigenvalue with respect to the power basis for the Hecke field in the generator whose minimal polynomial is $g(x)$.

Note that the stored defining polynomials $g(x)$ are not canonical defining polynomials for the Hecke field, but essential arbitrary. (Often $g(x)$ will be the characteristic polynomial of the first nontrivial eigenvalue.) Hence the same Hecke field may occur for different newforms with different defining polynomials; this will need to be taken into account when comparing Hecke fields: it is not sufficient to just compare the defining polynomials. For example, the Hecke field $\mathbb{Q}(\sqrt{5})$ may occur with $g(x) = x^2 - 5$ and also with $x^2 - x - 1$.

The collection `hmfs.hecke` contains 368356 entries, one for each entry in `hmfs.forms`. The Hecke fields have degrees up to 286.

## 4 Possible investigations

### 4.1 Degrees of Hecke fields

Over all newforms, or just those of each base field degree, describe the distribution of their dimensions, i.e. the degrees of the Hecke fields. This only requires accessing the key `hmfs.forms.dimension`, possibly subdivided by base field degree (`hmfs.forms.deg`) or individual base field (`hmfs.forms.field_label`).

### 4.2 The Hecke fields

Again this could be carried out over all newforms or just those with a fixed base field or fixed base field degree. For each Hecke field degree, determine which distinct Hecke fields occur and with what frequencies. This will require a database query to obtain the values of `hmfs.hecke.hecke_polynomial`, followed by construction of a field with each distinct polynomial found and a field isomorphism test, such as can be done by `SageMath` or `PARI/GP`. A variation of this would be to use `SageMath`'s or `PARI/GP`'s ability to find a unique canonical defining polynomial for every number field: this functionality is provided by the `C` library in the function `polredabs()`, and there is an interface to this library`pari` from `SageMath`.

Note that both the field isomorphism test and the canonicalisation test become very expensive when the degree increases. It may only be possible to carry out this (and subsequent) steps in the investigation for newforms of relatively low dimension, for this reason. However, no-one has yet systematically tried to look at these higher degree fields at all, and would be interesting to have a list of them, especially if it were possible to find simpler representations of them. Moreover there is an entry in the LMFDB wish-list to change the way to represent Hecke eigenvalues to something much more efficient and compact (in terms of storage space and of visual appearance on the web page), and a necessary first step would be to find such optimised representations.

### 4.3 Hecke field invariants

Following from the previous classification, a number of different invariants of each Hecke fields could be computed in `SageMath`: for example the class number.

### 4.4 Hecke eigenvalues and the ring they generate

Previous steps have only concerned the Hecke fields as fields, ignoring the lists of actual Hecke eigenvalues stored for each newform, in the `hmfs.hecke` collection. It would be interesting to study what subring of the ring of integers of the Hecke field they generate, what index it has in the ring of integers, and how many individual Hecke eigenvalues are require to generate this ring. All of these questions could be studied systematically in `SageMath`.

## 5 Code to illustrate shortcomings of existing methods

Here we give `SageMath` code which directly connects to the LMFDB database, extracts some of the data mentioned above, and carries out the first steps in processing this data. We discuss below some notes interspersed in the code, in order to illustrate the inadequacy of this approach. the code given here worked on 18 May 2018 but since it uses a low-level interface to the stored data, there is no guarantee that it will continue to work[1].

The first block of code requires the user's Sage installation to have the `Python` module `pymongo` installed, since this is used to communicate directly with the LMFDB database. The user needs to know the URL of the database, how to use `pymongo`, how to authenticate on to the database in a read-only way, as well as the names of the relevant databases and collections.

```
#  Connect to the LMFDB and authenticate (read-only):

import pymongo
C = pymongo.MongoClient(host='m0.lmfdb.xyz')
```

[1] In fact during 2018 the LMFDB will be converted from `Mongo` to `PostgreSQL`, after which the code give here will certainly no longer work.

```
C.admin.authenticate('lmfdb','lmfdb')

# assign names to the relevant collections:

hmfs = C.hmfs
fields = hmfs.fields
forms = hmfs.forms
hecke = hmfs.hecke
```

We define a utility to convert polynomials in $\mathbb{Q}[x]$, which (for Hecke polynomials) are stored as strings in the LMFDB, into `SageMath` polynomials. Although `SageMath` does have this capability built in, it cannot handle the unicode strings which `Mongo` uses.

```
# Define a utility for converting a polynomial in x stored as a
# string to a Sage polynomial in QQ[x].

Qx = PolynomialRing(QQ,'x') # 'x' is the variable used in the hecke collection
def decode_poly(pol):
    return Qx(str(pol))
```

Next we fetch from the collection `hmfs.forms` a list of the labels of the HMFs (here restricted to those whose base field degree and Hecke field degree are both equal to 2). For this step we need to know the name of the appropriate collection, and the names of the data fields `deg`, `dimension`, `label`, some of which (but not all) are abbreviations. We also need to know that `SageMath` automatically converts integer constants to its own multiprecision integer type, which `pymongo` cannot handle (since they cannot be encoded in `json`), so these parameters need to be manually converted to simple `Python int`s.

```
# Fix a base field degree and a Hecke field degree (dimension).

deg = int(2)
dim = int(2)

# Fetch the Hilbert newforms with this degree and dimension, and extract their labels:

res = forms.find({'deg':deg, 'dimension':dim})
labels = [f['label'] for f in res]
```

Now, for each of these labels, we fetch from the collection `hmfs.hecke` the associated Hecke polynomial. Again, we need to know that these are stored in a different collection, what the relevant keys are (`label` and `hecke_polynomial`), and how the polynomials are encoded so we may successfully convert them.

This step takes some time in practice for technical reasons, including the very large size of some of the records in the collection `hmfs.hecke`; this is somewhat alleviated by using `pymongo`'s parameter `projection` to only retrieve the data we need here and not all the individual Hecke eigenvalues.

```
# Fetch the Hecke polynomials for each of these,
#  converting each from a string to a Sage polynomial:

data = {}
for lab in labels:
    data[lab] = dat = {}
    dat['hecke_pol'] = hecke.find_one({'label':lab},
            projection=['hecke_polynomial'])['hecke_polynomial']
    dat['hecke_pol'] = decode_poly(dat['hecke_pol'])
```

At this point the `Python` dictionary `data` has HMF labels as keys, and their Hecke polynomials as values, and we can use SageMath functions to work with these. As a simple illustration, we merely construct the associated Hecke fields, and record their discriminants, counting how many times each field occurs, reporting on the results.

```
disc_counts = {}
for lab in labels:
    dat = data[lab]
    dat['hecke_field'] = H = NumberField(dat['hecke_pol'],'a')
    dat['hecke_disc'] = D = H.disc()
    disc_counts[D] = disc_counts.get(D,0)+1

# Output a list of which discriminants occur and with what frequency:

Hpols = Set([data[lab]['hecke_pol'] for lab in data])
Hfields = Set([data[lab]['hecke_disc'] for lab in data])
print("The {} newforms have {} different Hecke polynomials,".format(len(data), len(Hpols)))
print(" but only {} different Hecke fields".format(len(Hfields)))
print("Multiplicities of different Hecke field discriminants")
print("disc(H)\tMultiplicity")
for D in sorted(disc_counts.keys()):
    print("{}\t{}".format(D,disc_counts[D]))
```

The output reveals that the 31951 newforms selected have 403 different Hecke polynomials, but only 52 different Hecke fields, with discriminants ranging from $-68$ to $+268$, and that the field which occurs most frequently is $\mathbb{Q}(\sqrt{5})$.

## 6 Conclusions

While it is possible already to interface at a low level with the data in the LMFDB in order to carry out high-level mathematical investigation based on the data stored using exiting software packages, this interface is inadequate, hard to use, requires detailed technical knowledge of the way in which the data is stored, and is fragile: any changes in the database structure, whether minor changes such as the renaming of a key in a record, through medium-level changes such as rearranging the data into collections differently, to major upheavals as a result of switching database engines, would all require any code written in the manor shown here will immediately fail.

## References


1. The SageMath Developers. *Sage Mathematics Software (Version 8.2)*, 2018. `http://www.sagemath.org`.
2. The PARI group. *PARI/GP, version* `2.9.4`, 2017. `http://pari.math.u-bordeaux.fr/`.