Computing classical modular forms for the LMFDB

John Cremona

University of Warwick

—

joint work with Andrew Sutherland, John Voight, Andy Booker, Jonathan Bober, Edgar Costa, David Roe and others

17 January 2019
Atelier PARI/GP 2019, Bordeaux

EPSRC
Engineering and Physical Sciences Research Council

WARWICK
THE UNIVERSITY OF WARWICK

Overview

1. The LMFDB
2. Tables of modular forms
3. Software for computing modular forms
4. How the new data was computed
5. What the new data looks like

The LMFDB

I could give a whole talk about the LMFDB but will not!

The **L**-Functions and **M**odular **F**orms **D**ata**B**ase is a large collection of L-functions and mathematical objects which give rise to them: number fields, modular forms (of various kinds), elliptic curves (over $\mathbb{Q}$ and other fields) and curves of higher genus, Galois representations, and more.

See the LMFDB website `www.lmfdb.org`.

The LMFDB classical modular form collection

In this talk I will only concern myself with the collection of "classical modular forms" in the LMFDB, i.e. standard holomorphic modular forms of integer **level** $N \geq 1$, integer **weight** $k \geq 1$ and arbitrary **character** χ (a Dirichlet character modulo N).

The space $S_k(N, \chi)$ of these is a finite-dimensional complex vector space. The **new subspace** $S_k(N, \chi)^{\mathrm{new}}$ of forms not induced from lower levels has a distinguished basis of so-called **newforms**.

The computational task is to compute $S_k(N, \chi)^{\mathrm{new}}$ and its newforms for as wide a range of (N, k, χ) as is practical, including for each newform f all additional quantities of interest to researchers, storing them in a database in a way which is compact, yet also allows flexible searching.

Previous collections of classical modular forms

There have been several earlier collections of modular forms, including some with particular restrictions on weight and character. These include:

- The tables of Cohen, Skoruppa and Zagier: $N \leq 1000$, various weights and characters. c.1990, using Pari/GP.
- Weight 2 and trivial character, especially of dimension 1 because of the link with elliptic curves: Tingley 1972, myself 1986-2016.
- William Stein's tables: c.1998-2005, using C++ and then Magma.

Classical modular forms in the LMFDB

The LMFDB modular forms collection has evolved roughly as follows:

- 0.0: (pre 2015) database contained Stein's data with a web interface;
- 1.0: (2015–2018) new data computed by Stromberg and Ehlen using Stein's Sage code, same web interface;
- 2.0: (2018–, to be released in early 2019) new data computed by many using Pari/GP, Magma, and/or C, new database, new web interface: this talk.

Here is a comparison of the extent of these, provided by Andrew Sutherland.

Classical modular forms in the LMFDB

	Stein DB	LMFDB old	LMFDB new
Nonzero $S_k^{\mathrm{new}}(\Gamma_1(N))$	315	756	7045
Nonzero $S_k^{\mathrm{new}}(N,\chi)$	8,579	3,416	45,444
Newforms (Galois orbits)	84,407	8,659	236,555
Rational newforms	25,806	2,292	48,324
Complex embeddings	1,095,619	77,434	1,3710,564
Eigenvalues	≈3,000,000	2,418,750	585,223,000
Weight 1 newforms	0	0	19,306
Weight 2 newforms	76,896	2,914	155,759
Newforms on $\Gamma_0(N)$	83,694	5,731	171,238
Max dimension	340	208	39690
Max weight	78	200	316
Max level	7248	549	10,000

Computing the new data set has taken 6 months, and of the order of a few CPU-decades.

Computing the new collection: the team

After some preparatory work, during the week of 27–31 August 2018, a workshop was held at MIT, with:

Alex Best, Jon Bober, Andy Booker, Edar Costa, me, David Lowry-Duda, David Roe, Andrew Sutherland, John Voight,

funded by the US Simons Foundation, through the Simons Collaboration on Arithmetic Geometry, Number Theory, and Computation, and the UK EPSRC, through a "Programme grant", LMF.

Work has continued since the workshop and a release is imminent. Here is a sneak preview: `http://cmfs.lmfdb.xyz/ModularForm/GL2/Q/holomorphic/`

Computing the new collection: the methods

We did **not** develop any new methods for computing modular forms, and neither did we write any new implementations, except for a considerable amount of code to “post-process” the data computed by available packages.

We used

- Magma (code originally by Stein with later work by Donnelly): uses modular symbols; used only for weights $k \geq 2$
- Pari/GP version 2.11.1 (code by Cohen, Belabas et al.): uses trace formulas; all weights especially $k = 1$
- `C` code of Bober and Booker: numerical and rigorous, using `arb`

Computing the new collection: the strategy

One aim was to make sure that in as many cases as possible everything would be computed by at least two of the above methods. We agreed on the format of text data files (see `https://github.com/JohnCremona/CMFs#cmfs`) and wrote code to compare these for consistency.

For example, to compare the Pari and Magma output for fixed (N, k, χ), after checking that the irreducible pieces have the same dimensions, it was necessary to find an isomorphism from each Hecke field in the Magma output to the Hecke field in the Pari output and check that the q-expansion coefficients matched. Using `polredabs` wherever possible, and iterated `polredbest` otherwise, helped.

The largest dimension in the database currently is 39690 (for form 983.2.c.a). For such large dimensions we do not compute or store exact algebraic q-expansions, only the traces and all complex embeddings of a_n for several n.

Storing and displaying the new collection

Each newform $f \in S_k(N, \chi)^{\mathrm{new}}$ has a q-expansion

$$\sum_{n=1}^{\infty} a_n q^n \qquad \text{where } q = \exp(2\pi i \tau)$$

so is determined by the sequence of algebraic integers a_n which generate an algebraic number field $K_f = \mathbb{Q}(a_1, a_1, \dots)$ whose degree is the **dimension** of f. We have $K_k \supseteq \mathbb{Q}(\chi)$, and both Pari and Magma give the a_n as elements of a relative extension $K_f = \mathbb{Q}(\chi)(\nu)$.

This for each newform f we wish to store, and possibly display, the field K_f and the sequence (a_n) of algebraic integers in K.

Storing and displaying q-expansions

We use an **absolute** and **integral** representation for the a_n: instead of writing each a_n as a polynomial in ν with coefficients in $\mathbb{Q}$ or $\mathbb{Q}(\chi)$, we compute an integral basis $\{\beta_j\}$ for the **ring** $R_f = \mathbb{Z}[a_1, a_2, \dots]$ which is an order in K_f. Now each a_n is a $\mathbb{Z}$-linear combination of the β_j, and using LLL-reduction we make these integers small. There are three steps in this:

1. choice of the best polynomial to define the field K_f
2. choice of an integral basis for the Hecke ring $\mathbb{Z}[\dots a_n \dots]$
3. each a_n is now stored as a list of d integers

Using LLL-reduction the integral basis is chosen so that the coefficients in (3) are small.

This reduces considerably the volume of data stored, and also makes it possible to display exact algebraic coefficients for much higher dimensions than otherwise.

Implementation notes

We use the `cypari` package in Sage, so that we can do all high-level coding using Python, calling Pari's `mf⋆` functions to do the real work.