Tutorial for Modular Forms in Pari/GP

Henri Cohen

October 12, 2026

1 Introduction

Three packages are available to work with modular forms and related functions in `Pari/GP`. The first one is the L-function package, which has been available since 2.9.0 (2015), and computes with general motivic L-functions, and in particular with L-functions attached to Dirichlet characters, Hecke characters, Artin representations, and modular forms. The name of most functions in this package begins with `lfun`, such as `lfuninit`.

The second is the modular symbol package, whose primary aim is not so much to compute modular form spaces and modular forms, but to compute p-adic L-functions attached to modular forms. The name of most functions in this package begins with `ms`, such as `msinit`.

The third package is the modular forms package, whose aim is to compute in the standard spaces $M_k(\Gamma_0(N),\chi)$ with k integral or half-integral, both with modular form *spaces* and individual modular *forms*. The name of most functions in this package begins with `mf`, such as `mfinit`. The goal of the present manual is to describe this package in view of a guide for a new user, so will essentially be a tutorial, although we include a reference guide at the end.

We can work on five subspaces of $M_k(\Gamma_0(N),\chi)$, through a corresponding *space flag* in the commands: the cuspidal *new space* $S_k^{\rm new}(\Gamma_0(N),\chi)$ (flag = 0), the full cuspidal space $S_k(\Gamma_0(N),\chi)$ (flag = 1), the old space $S_k^{\rm old}(\Gamma_0(N),\chi)$ (flag = 2, probably of little use), the space generated by all Eisenstein series $\mathcal{E}_k(\Gamma_0(N),\chi)$ (flag = 3), and finally the full space including the Eisenstein part $M_k(\Gamma_0(N),\chi)$ (flag = 4, which can be omitted since it is the default). Note that although it can be defined, we have not included the space $M_k^{\rm new}$, nor the "certain space" of Zagier–Skoruppa. In

the half-integral weight case, we have included only the full cuspidal space and the full space (flags 1 and 4), as well as the Kohnen's +-space and the corresponding newspace and eigenforms when N is squarefree.

Note in particular that the package includes the computation of modular forms of weight $k = 1$ and of half-integral weight.

The modular forms themselves are represented in a special internal format which the user need not worry about and which basically is a recipe to compute successive Fourier coefficients at infinity: if F is a GP modular form, `mfcoefs`$(F, 10)$ will give you the Fourier coefficients at infinity from $a(0)$ to $a(10)$ of the modular form corresponding to F as a row *vector* (if you want a power series expansion, use the GP function `Ser`, see below). Many operations are available on such objects, but the most important thing the user needs to know is that the number of Fourier coefficients need not be specified in advance: the command `mfcoefs`(F, n) is valid for any integer n. We will of course explain the details of this below.

Finally, note that we may roughly divide the complexity of available functions into three levels:

1. The first level includes all the basic modular form and modular spaces creation and operations. Many functions are very fast, but some are quite time-consuming for very different reasons; first those dealing with forms and spaces involving Dirichlet characters of large order; second finding eigenforms when the splitting using the Hecke algebra is difficult; and third modular spaces of weight 1 when there exist "exotic" forms. Reasonable levels (for low weight) can go up to a few thousands. The critical parameter is actually the dimension of the underlying modular form space where linear algebra needs to be performed, so the time complexity is at least proportional to $(N \times k)^3$, and in fact more than that due to coefficient explosion in the base field $\mathbb{Q}(\chi)$.

2. The second level needs technical information about spaces generated by products of two Eisenstein series, and is quite expensive. But it allows to perform computations which would be almost impossible otherwise, such as Fourier expansions of $f|_k\gamma$ (hence at any cusp), numerical evaluation of modular forms at any point in the upper half-plane (even close to the real axis) or L-functions attached to an arbitrary form. We have included a caching method, so that once a single such computation is performed in a given space, the needed technical data is stored and no longer needs to be computed so that all subsequent

calls are much faster. Reasonable levels (for low weight) can go up to one thousand, say. Again, the critical parameter is the space dimension but this time the linear algebra is performed in large degree cyclotomic fields, even when the Nebentypus is trivial. Compared to the first level, we lose at least a factor $\Lambda(N)$ (the exponent of the multiplicative group $(\mathbb{Z}/N\mathbb{Z})^*$) in the time complexity, which gets as large as $N-1$ if N is prime.

3. The third level, which uses the second level functions, allows more numerical computations such as period polynomials, modular symbols, Petersson products, etc... The time complexity does not increase much since it is dominated by the second level.

2 Creation of Modular Forms

In `Pari/GP` modular forms can be created in three different ways:

- As *basic modular forms*, i.e., forms attached (or not) to different mathematical objects, and which are of so frequent use that we have implemented them so that the user has them at his disposal. Examples: `mfDelta` (Ramanujan's delta), `mfEk` (Eisenstein series of weight k on the full modular group; of course we also have more general Eisenstein series), `mffrometaquo` (eta quotients), `mffromell` (modular form attached to an elliptic curve over $\mathbb{Q}$), `mffromqf` (modular form attached to a lattice).

- From existing forms by applying *operations*. Examples: multiplication/division, linear combination, derivation and integration, Serre derivative, RC-brackets, Hecke and Atkin–Lehner operations, expansion and diamond operators, etc...

- Through the creation of the modular form *spaces*: typically, if only `mf=mfinit` is applied, then a basis of forms is obtained by the command `mfbasis(mf)`. The command `mfeigenbasis(mf)` produces the canonical basis of eigenforms.

3 A First Session: working with Leaves

This is now a tutorial session. We will see sample commands as we go along.

```
? D = mfDelta(); V = mfcoefs(D, 8)
% = [0, 1, -24, 252, -1472, 4830, -6048, -16744, 84480]
```

The command `mfcoefs(D,n)` gives the vector of Fourier coefficients (at infinity) $[a(0), a(1), \ldots, a(n)]$ (note that there are $n+1$ coefficients). This is a compact representation, but if you prefer power series you can use `Ser(V,q)` (convert a vector into a power series).

```
% = q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6\
      - 16744*q^7 + 84480*q^8 + O(q^9)
```

(This simple-minded recipe only works when the form has rational coefficients. Make sure to use `q = varhigher("q")` first if the form has non-rational algebraic coefficients to avoid problems with variable priorities.) Similarly

```
? E4 = mfEk(4); E6 = mfEk(6); apply(f->mfcoefs(f,3),[E4,E6])
% = [[1, 240, 2160, 6720], [1, -504, -16632, -122976]]
? E43 = mfpow(E4, 3); E62 = mfpow(E6, 2);
? DP = mflinear([E43, E62], [1, -1]/1728);
? mfcoefs(DP, 6)
% = [0, 1, -24, 252, -1472, 4830, -6048]
? mfisequal(D, DP)
% = 1
```

Self-explanatory. Note that there is a command `mfcoef(F, n)` (without the final "s") which simply outputs the coefficient $a(n)$. A final example of the same type:

```
? F = mffrometaquo([1,2; 11,2]); mfcoefs(F,10)
% = [0, 1, -2, -1, 2, 1, 2, -2, 0, -2, -2]
? G = mffromell(ellinit("11a1"))[2];
? mfisequal(F, G)
% = 1
```

Here, `mffrometaquo` takes as argument a matrix representing an *eta quotient*, here $\eta(1 \times \tau)^2 \eta(11 \times \tau)^2$.

The second component of the `mffromell` output is the modular form associated to the elliptic curve by modularity.

4 A Second Session: Modular Form Spaces

In the first session, we have seen a few preinstalled modular forms (that we can call *leaves*), and a number of operations on them. All reasonable operations have been implemented (if some are missing, please tell us). We are now going to work with *spaces* of modular forms.

```
? mf = mfinit([1,12]); L = mfbasis(mf); #L
% = 2
? mfdim(mf)
% = 2
```

This creates the full space of modular forms of level 1 and weight 12. This space is created thanks to an almost random basis that one can obtain using `mfbasis`, and we see either by asking for the number of elements of `L` or by using the command `mfdim`, that it has dimension 2, not surprising. We can see it better by writing:

```
? mfcoefs(L[1],6)
% = [691/65520, 1, 2049, 177148, 4196353, 48828126, 362976252]
? mfcoefs(L[2],6)
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

or simply

```
? mfcoefs(mf,6)    \\ apply mfcoefs to mfbasis elements
% =
[691/65520     0]
[         1     1]
[      2049   -24]
[    177148   252]
[   4196353 -1472]
[  48828126  4830]
[362976252 -6048]
```

Note two things: first, the Eisenstein series are given before the cusp forms (this may change, but for now this is the case), and second, the Eisenstein series is normalized so that it is the coefficient $a(1)$ which is equal to 1, and not $a(0)$. In particular, here at least, it is a normalized Hecke eigenform.

If we want to work only in the cuspidal space $S_{12}(\Gamma)$, we simply use the flag 1, such as:

```
? mf = mfinit([1,12], 1); L = mfbasis(mf); #L
% = 1
? mfcoefs(L[1],6)
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

Let us now look at higher dimensional cases. In the following example, we consider the *new space* (flag = 0), although in the present case this is the same as the cuspidal space:

```
? mf = mfinit([35,2], 0); L = mfbasis(mf); #L
% = 3
? for (i = 1, 3, print(mfcoefs(L[i], 10)))
[0, 3, -1, 0, 3, 1, -8, -1, -9, 1, -1]
[0, -1, 9, -8, -11, -1, 4, 1, 13, 7, 9]
[0, 0, -8, 10, 4, -2, 4, 2, -4, -12, -8]
```

These are essentially random cusp forms. Usually, you want the eigenforms: this is obtained by the function `mfeigenbasis` (note in passing that `B=mfeigenbasis(mf)` adds components to `mf`, so that the next call is instantaneous). You can ask for the defining number fields with the command `mffields`. Note that these commands act only on the new space, but the package also accepts the spaces that contain it (such as the cuspidal space or the full space, but not the old space), although the result is only about the new space.

```
? mffields(mf)
% = [y, y^2 - y - 4]
? L = mfeigenbasis(mf); #L
% = 2
? mfcoefs(L[1],10)
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
? mfcoefs(L[2],4)
% = [Mod(0, y^2 - y - 4), Mod(1, y^2 - y - 4),\
     Mod(-y, y^2 - y - 4),Mod(y - 1, y^2 - y - 4),\
     Mod(y + 2, y^2 - y - 4)]
? lift(mfcoefs(L[2],10))
% = [0, 1, -y, y - 1, y + 2, 1, -4, -1, -y - 4, -y + 2, -y]
```

The command `mffields` gives the polynomials in the variable y defining the number field extensions on which the eigenforms are defined. Here, one of the fields is $\mathbb{Q}$, the other is $\mathbb{Q}(\sqrt{17})$. To obtain the eigenforms, we use

`mfeigenbasis`, and there are only two and not three, since the one defined on $\mathbb{Q}(\sqrt{17})$ goes together with its conjugate. Asking directly `mfcoefs(L[2],4)` gives the coefficients as `polmods`, not easy to read, so it is usually preferable to *lift* them, giving the last command, where in the output we must of course remember that y stands for *one* of the two roots of $y^2 - y - 4 = 0$, i.e., $(1 \pm \sqrt{17})/2$.

In fact, for some numerical computations, we really need the coefficients of the eigenform embedded in $\mathbb{C}$, and not just as abstract algebraic numbers (in our case of trivial character, they will be in $\mathbb{R}$). This is why a few functions (most notably `mfeval` and `lfunmf`) will return a *vector* of results and not a scalar when called on such a form.

For instance, here is a little GP script which computes the numerical expansion of a modular form instead of the expansion in `polmods`:

```
mfcoefsembed(F,n) = mfembed(F, mfcoefs(F,n));
```

Note that this produces a vector of expansions when the eigenforms are defined over an extension, i.e. $[\mathbb{Q}(F) : \mathbb{Q}(\chi)] > 1$, one per conjugate form.

```
? mfcoefsembed(L[2],5)  \\ two conjugate forms
%4 = [[0, 1, 1.5615..., -2.5615..., 0.43844..., 1],
      [0, 1, -2.561...,, 1.5615...,, 4.5615..., 1]]
```

The first eigenform found above is *rational*, hence by the modularity theorem there exists up to isogeny a unique elliptic curve to which it corresponds. We check this by writing

```
? [mf,F] = mffromell(ellinit("35a1")); mfcoefs(F, 10)
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
? mfisequal(F, L[1])
% = 1
```

For a more typical example (still with no character):

```
? [ mfdim([96,2], flag) | flag <- [0..4] ]
% = [2, 9, 7, 15, 24]
```

This gives us the dimensions of the new space, the cuspidal space, the old space, the space of Eisenstein series, and the whole space of modular forms.

Just for fun, we write (recall that the default is the full space):

```
? mf = mfinit([96,2]); L = mfbasis(mf);
? for (i = 12, 15, print(mfcoefs(L[i], 15)))
[23/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24]
[31/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24]
[47/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24]
[95/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24]
```

Apparently, these four Eisenstein series differ only by their constant term, which is of course not possible. Indeed:

```
? F = mflinear([L[14],L[12]],[1,-1]); mfcoefs(F, 50)
% = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\
     0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\
     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0]
? G = mfhecke(mf, F, 24); mfcoefs(G, 12)
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288, 96]
? mftobasis(mf, G)
% = [0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\
     0, 0, 0, 0, 0, 0]~
? 24*mfcoefs(L[5], 12)
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288, 96]
```

The first command shows that the Eisenstein series differ on their n-th Fourier coefficient for $n = 0$, 24, and 48, and the second command applies the Hecke operator T_{24} (sometimes denoted U_{24}) to the difference, whose effect is to replace $a(n)$ by $a(24n)$, giving the much more compact output of G. The last commands show that G is equal to 24 times the fifth Eisenstein series `L[5]`.

```
? mf=mfinit([96,2],0); mffields(mf)
% = [y, y]
? L = mfeigenbasis(mf); for(i=1, 2, print(mfcoefs(L[i], 16)))
[0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2, 0]
[0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2, 0]
? Fa = mffromell(ellinit("96a1"))[2]; mfcoefs(Fa, 16)
% = [0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2, 0]
? Fb = mffromell(ellinit("96b1"))[2]; mfcoefs(Fb, 16)
% = [0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2, 0]
```

The `mffromell` function returns a triple `[mf,F,C]`, where `mf` is the modular form cuspidal space to which `F` belongs, `F` is the rational eigenform

corresponding to the elliptic curve by modularity, and `C` is the vector of coefficients of `F` on the basis in `mf`, which we recall is usually not a basis of eigenforms (otherwise `F` would belong to this basis).

Note also that `Fa` and `Fb` are twists of one another:

```
? mfisequal(mftwist(Fa, -4), Fb)
% = 1
```

5 Interlude: Dirichlet characters

There are many ways to represent multiplicative characters on $(\mathbb{Z}/N\mathbb{Z})^*$ in `Pari/Gp`, we will list them by increasing order of sophistication, restricting to characters with complex values:

- A quadratic character $(D/.)$ (Kronecker symbol) is described by the integer D. For instance 1 is the trivial character.

- There is a (noncanonical but fixed) bijection between $(\mathbb{Z}/N\mathbb{Z})^\times$ and its character group, via *Conrey labels*. So `Mod`(a, N) represents a character whenever a is coprime to N. This makes it easy to loop on all characters without worrying too much about which is which. In this labeling, `Mod(1,N)` is the trivial character, and characters are multiplied/divided by performing the corresponding operation on their Conrey labels.

- The finite abelian group $G = (\mathbb{Z}/N\mathbb{Z})^*$ is written

$$G = \bigoplus_{i \le n} (\mathbb{Z}/d_i\mathbb{Z}) \cdot g_i,$$

  with $d_n \mid \cdots \mid d_2 \mid d_1$ (SNF condition), all $d_i > 0$, and $\prod_i d_i = \phi(N)$. The SNF condition makes the d_i unique, but the generators g_i, of respective order d_i, are definitely not unique. The $\oplus$ notation means that all elements of G can be written uniquely as $\prod_i g_i^{n_i}$ where $n_i \in \mathbb{Z}/d_i\mathbb{Z}$. The g_i are the so-called *SNF generators* of G. The command `znstar`(N) outputs the SNF structure (group order, d_i and g_i), but $G =$ `znstar`$(N, 1)$ is needed to initialize a group we can work with: most importantly we can now solve discrete logarithm problems and decompose elements on the g_i.

  A character on the abelian group $\oplus(\mathbb{Z}/d_j\mathbb{Z})g_j$ is given by a row vector $\chi = [a_1, \ldots, a_n]$ of integers $0 \le a_i < d_i$ such that $\chi(g_j) = e(a_j/d_j)$ for

all j, with the standard notation $e(x) := \exp(2i\pi x)$. In other words, $\chi(\prod g_j^{n_j}) = e(\sum a_j n_j/d_j)$. In this encoding $[0,\dots,0]$ is the trivial character. Of course a character χ must always be given as a *pair* $[G,\chi]$, since χ is meaningless without knowledge of the (g_i) or the (d_i).

The command `znchar`(S) converts a datum describing a character to the third form $[G,\chi]$. The command `znchartokronecker` converts a character of order ≤ 2 to the first form $(D/.)$, and functions such as `zncharconductor`, `znchartoprimitive`, and `zncharinduce` allow to restrict or extend characters between different $(\mathbb{Z}/M\mathbb{Z})^*$.

Note the important fact that it is necessary to give the two arguments G and χ separately to these functions, for instance `zncharconductor(G,chi)` (and not `zncharconductor([G,chi])`).

Functions such as `charmul`, `chardiv`, `charpow`, `charorder` or `chareval` apply to more general abelian characters than characters on $(\mathbb{Z}/N\mathbb{Z})^\times$, whence the prefix `char` instead of `znchar`.

6 A Third Session: Nontrivial Characters

Recall that a nontrivial character can be represented either by a discriminant D (not necessarily fundamental), the character being the Legendre–Kronecker symbol (D/n), or by its *Conrey label* in $(\mathbb{Z}/N\mathbb{Z})^\times$, for instance `Mod(161,633)` (which has order 42, as `znorder` tells us).

Defining modular form spaces with character is as simple as without: we replace the parameters $[N,k]$ by $[N,k,\chi]$. Instead of `mf=mfinit([35,2])`, one can write `mf=mfinit([35,2,5], 0)`, where 5 is the quadratic character $(5/.)$. Thus:

```
? mf = mfinit([35,2,5],0); mffields(mf)
% = [y^2 + 1]
? F = mfeigenbasis(mf)[1]; lift(mfcoefs(F, 10))
% = [0, 1, 2*y, -y, -2, -y - 2, 2, -y, 0, 2, -4*y + 2]
```

where in the last output y is equal to one of the two roots of $y^2+1=0$, i.e., $\pm i$.

Working with nontrivial characters allows us in particular to work with odd weights, and in particular in weight 1:

```
? mf = mfinit([23,1,-23], 0); mfdim(mf)
```

```
% = 1
? F = mfbasis(mf)[1]; mfcoefs(F, 16)
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, -1, 0, 0, -1]
? mfgaloistype(mf,F)
% = 6
```

The last output means that the image in $\mathrm{PSL}_2(\mathbb{C})$ of the projective representation associated to F is of type D_3. Note that an "exotic" representation is given by a negative number, opposite of the cardinality of the projective image.

Since this form is of dihedral type, it can be obtained via theta functions. Indeed:

```
? F1 = mffromqf([2,1; 1,12])[2]; V1 = mfcoefs(F1, 16)
% = [1, 2, 0, 0, 2, 0, 4, 0, 4, 2, 0, 0, 4, 0, 0, 0, 2]
? F2 = mffromqf([4,1; 1,6])[2]; V2 = mfcoefs(F2, 16)
% = [1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 4, 2, 0, 0, 4]
? (V1 - V2)/2
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, -1, 0, 0, -1]
? mfisequal(F, mflinear([F1, F2], [1, -1]/2))
% = 1
```

Here we were lucky in that we "knew" that the correct character was $(-23/n)$. But what if we did not know this ? The first observation is that modular form spaces corresponding to Galois conjugate characters are isomorphic (χ is Galois conjugate to χ' if $\chi' = \chi^m$ for some m coprime to the order of χ). Thus, it is sufficient to find a representative of each equivalence class, and this is given by the `GP` commands `G=znstar(N,1); chargalois(G)`, where N is the level of the desired character (note that N will not necessarily be the conductor of the characters). This exactly outputs a list of representative of each equivalence class (do not for now try to understand the details of this command, nor the fact that `chargalois` and `znstar` have optional parameters). However, this is not quite yet what we want. Although only for efficiency, we want characters with the same parity as the weight, otherwise the corresponding modular form spaces will be 0. This is achieved by the `GP` command `zncharisodd(G,chi)` which does what you think it does. Let us first do this for $N = 23$: we write

```
? G = znstar(23, 1);
? L = [chi | chi<-chargalois(G), zncharisodd(G,chi)]; #L
% = 2
```

```
? [mfdim([23,1,[G,chi]], 0) | chi <- L ]
% = [0, 1]
? [charorder(G,chi) | chi <- L]
% = [22, 2]
```

This tells us that (up to Galois conjugation) there are two possible odd characters, one, of order 22, giving a 0-dimensional space, the other being the quadratic character given above. Note that `chargalois` returns (orbits of) characters attached to an arbitrary abelian finite group G while `mfinit` expects a *pair* `[G,chi]` for some `znstar` G, as written above.

When doing long explorations with all characters of a certain level, it is preferable to use *wildcards*. For instance, instead of the above one can write:

```
? mfall = mfinit([23,1,0], 0); #mfall
% = 1
? mf = mfall[1]; mfdim(mf)
% = 1
? mfparams(mf)
% = [23, 1, -23, 0]
```

This does not exactly give us the same information: the third parameter 0 in the first command asks for *all* nonempty spaces of level 23 and weight 1, and the program tells us that there is only one, of dimension 1. The last command `mfparams` outputs `[N,k,CHI,space]`, so here tells us that the corresponding character is the Kronecker–Legendre symbol $(-23/n)$.

Using wildcards, let us explore levels in certain ranges: we write

```
wt1exp(lim1,lim2)=
{ my(mfall,mf,chi,v);
  for (N = lim1, lim2,
    mfall = mfinit([N,1,0], 0); /* use wildcard */
    for (i=1, #mfall,
      mf = mfall[i];
      chi = mfparams(mf)[3]; /* nice format: D or Mod(a,N) */
      [ print([N,chi,-t]) | t<-mfgaloistype(mf), t < 0 ]
    )
  );
}
```

For instance, `wt1exp(1,230)` outputs in 4 seconds

```
[124, Mod(87, 124), 12]
[133, Mod(83, 133), 12]
[148, Mod(105, 148), 24]
[171, Mod(94, 171), 12]
[201, Mod(104, 201), 12]
[209, Mod(197, 209), 12]
[219, Mod(8, 219), 12]
[224, Mod(95, 224), 12]
[229, Mod(122, 229), 24]
[229, Mod(122, 229), 24]
```

Thus, the smallest exotic A_4 form is in level 124 and the smallest S_4 form is in level 148. Note that in level 229, we have two (non Galois conjugate) eigenforms of type S_4.

If we type `wt1exp(633,633)`, in 6 seconds we obtain `[633, Mod(107, 633), 60]`, and this level is indeed the lowest level for which there exists a type A_5 form. The character orders are obtained either as `znorder(chi)` (since all the `chi` are `intmods`), or using the general construction

```
[G,v] = znstar(chi);
ord = charorder(G,v)
```

where we first convert `chi` to a general abelian character in $[G, \chi]$ format.

7 Leaf Functions

Although we have already seen most of these functions in the first session, we repeat some of examples here.

7.1 Functions Created from Scratch

We now start a slightly more systematic exploration of the available functions. We begin by *leaf functions*, i.e., functions created from scratch or from a given mathematical object.

```
? D = mfDelta(); mfcoefs(D, 5)
% = [0, 1, -24, 252, -1472, 4830]
? E4 = mfEk(4); mfcoefs(E4, 5)
% = [1, 240, 2160, 6720, 17520, 30240]
? E6 = mfEk(6);
? D2 = mflinear([mfpow(E4, 3), mfpow(E6, 2)], [1, -1]/1728);
```

```
? mfisequal(D, D2)
% = 1
```

Self-explanatory. More complicated Eisenstein series:

```
? E3 = mfeisenstein(1, 1, -3); mfcoefs(E3, 10)
% = [1/6, 1, 0, 1, 1, 0, 0, 2, 0, 1, 0]
? E4 = mfeisenstein(5, -4, 1); mfcoefs(E4, 10)
% = [5/4, 1, 1, -80, 1, 626, -80, -2400, 1, 6481, 626]
? H2 = mfEH(5/2); mfcoefs(H2,10)
% = [1/120, -1/12, 0, 0, -7/12, -2/5, 0, 0, -1, -25/12, 0]
```

The `mfeisenstein(k,c1,c2)` command generates the Eisenstein series of weight k and characters `c1` and `c2`. The `mfEH(k)` command is specific to half-integral weight k and generates the Cohen–Eisenstein series of weight k.

```
? T = mfTheta(); mfcoefs(T,16)
% = [1, 2, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 2]
? mf = mfinit([4, 5, -4]); mftobasis(mf, mfpow(T, 10))
% = [64/5, 4/5, 32/5]
? B = mfbasis(mf); apply(mfdescribe, B)
% = ["F_5(1, -4)", "F_5(-4, 1)", "TR^new([4, 5, -4, y])"]
? mfisCM(B[3])
% = -4
```

Here, we compute the coefficients of θ^{10} on the basis of `mf` (we know of course the level, weight, and character). We then apply the `mfdescribe` function, which tells us that the first two forms in the basis are Eisenstein series, and the third one is some trace form on the cuspidal new space. However, the last command says that this third basis element is a *CM form*, so that its coefficients can be computed just as fast as those of Eisenstein series, so that there does exist an explicit formula for the number of representations as a sum of ten squares.

Keeping the above sessions, we can also write:

```
? mftobasis(mf, mfpow(H2, 2))
% = [1/18000, 1/18000, -3/2000]~
```

7.2 Functions Created from Mathematical Objects

```
? [mf,F,co] = mffromell(ellinit("26b1")); co
% = [1/2, 1/2]~
? mfcoefs(F,10)
% = [0, 1, 1, -3, 1, -1, -3, 1, 1, 6, -1]
```

This creates the modular form attached by modularity to the second isogeny class of elliptic curves over $\mathbb{Q}$ for conductor 26. The result is a 3-component vector: `mf` is the modular form space, F the modular form, and `co` are the coefficients of F on the basis of `mf`.

Similarly, there are functions `mffromlfun` (from L-functions attached to eigenforms), `mffromqf` (from quadratic forms) and `mffrometaquo`:

```
? F = mffrometaquo([1, 2; 11, 2]); mfcoefs(F, 10)
% = [0, 1, -2, -1, 2, 1, 2, -2, 0, -2, -2]
? F = mffrometaquo([1, 2; 2, -1]); mfparams(F)
% = [16, 1/2, 1, y]
? mfcoefs(F, 10)
% = [1, -2, 0, 0, 2, 0, 0, 0, 0, -2, 0]
```

The `mfparams` command tells us that $F \in M_{1/2}(\Gamma_0(16))$.

8 Atkin, Hecke and Expanding Operators

```
? mf = mfinit([96,4], 0); mfdim(mf)
% = 6
? M = mfheckemat(mf, 7)
% =
[0    0   0    372    696   0]

[0    0  36      0      0 -96]

[0 27/5   0 -276/5 -276/5   0]

[1    0 -12      0      0  62]

[0    0   1      0      0 -16]

[0 -3/5   0   14/5  -16/5   0]
```

```
? P = charpoly(M)
% = x^6 - 1456*x^4 + 209664*x^2 - 2985984
? factor(P)
% =
[x - 36 1]

[x - 12 1]

[ x - 4 1]

[ x + 4 1]

[x + 12 1]

[x + 36 1]
```

Note a few things: first, the matrix of the Hecke operator $T(7)$ does not have integral coefficients. Indeed, recall that the basis of modular forms in `mf` is mostly random, so there is no reason for the matrix to be integral. On the other hand, since the eigenvalues of Hecke operators are algebraic integers, the characteristic polynomial of $T(7)$ must be monic with integer coefficients. As it happens, it factors completely into linear factors to the power 1, so all the eigenvalues of $T(7)$ are in fact in $\mathbb{Z}$: this immediately shows that the splitting will be entirely rational and the eigenforms with integer coefficients. Let's check:

```
? mffields(mf)
% = [y, y, y, y, y, y]
? L = mfeigenbasis(mf); for(i=1,6,print(mfcoefs(L[i],16)))
[0, 1, 0, 3, 0, 10, 0, 4, 0, 9, 0, -20, 0, 70, 0, 30, 0]
[0, 1, 0, 3, 0, 2, 0, 12, 0, 9, 0, 60, 0, -42, 0, 6, 0]
[0, 1, 0, 3, 0, -14, 0, -36, 0, 9, 0, -36, 0, 54, 0, -42, 0]
[0, 1, 0, -3, 0, 10, 0, -4, 0, 9, 0, 20, 0, 70, 0, -30, 0]
[0, 1, 0, -3, 0, 2, 0, -12, 0, 9, 0, -60, 0, -42, 0, -6, 0]
[0, 1, 0, -3, 0, -14, 0, 36, 0, 9, 0, 36, 0, 54, 0, 42, 0]
```

We see that of the six eigenforms, the last three are twists of the first three.

There also exists the command `G=mfhecke(mf,F,n)`, which given a modular form F in `mf`, outputs the modular form $T(n)F$.

```
? mf=mfinit([96,6],0); mffields(mf)
% = [y, y, y, y, y, y, y^2 - 31, y^2 - 31]
? mfatk = mfatkininit(mf,3);
% factor(charpoly(mfatk[2]/mfatk[3]))
% =
[x - 1 5]

[x + 1 5]
```

This requires a little explanation: the command `mfatkininit(mf,3)` computes a number of quantities necessary to work with the Atkin–Lehner operator W_3 in the space `mf`. The main part of the result is the second component, which is essentially the matrix of W_3 on the basis of `mf`, and which is guaranteed to have exact coefficients (here rational). However in the general case, the matrix of W_3 is equal to `mfatk[2]/mfatk[3]`, where `mfatk[3]` may be an inexact complex number. For now you need not worry about the first component.

Thus, the eigenvalues (or possibly the pseudo-eigenvalues) must be of modulus 1, and in the case of a quadratic character defined modulo N/Q, they are equal to ± 1 in even weight, to $\pm i$ in odd weight. Here, 1 and -1 both occur 5 times. However, this does not tell us which eigenvalues correspond to each eigenspace. For this, we do the following:

```
? mfatkineigenvalues(mf,3)
% = [[-1], [-1], [-1], [1], [1], [1], [-1, -1], [1, 1]]
? mf=minit([96,3,-3],0); mffields(mf)
% = [y^4 + 8*y^2 + 9, y^4 + 4*y^2 + 1]
? mfatkineigenvalues(mf,32)
% = [[I, -I, -I, I], [-I, I, I, -I]]
? mfatkineigenvalues(mf,3)
% = [[a, -conj(a), -a, conj(a)], [b, -conj(b), conj(b), -b]]
```

The first command tells us that in the six rational eigenspaces, the first three have eigenvalue -1, the other three $+1$, and in the eigenspaces of dimension 2, the first eigenspace has both eigenvalues -1, the second both $+1$. As is seen from the next lines, it is of course not necessary for the eigenvalues of W_Q in the same eigenspace to be equal.

In the next two commands, we are now in a case where the character is non trivial and the weight odd. The eigenvalues are now $\pm i$, and not equal in the same eigenspace.

Finally, the last command is a case where the character is not defined modulo $N/Q = 96/3 = 32$, so we only have pseudoeigenvalues, which are simply of absolute value 1 by Atkin–Lehner theory. Here, a and b are complicated complex numbers and `conj` denotes the complex conjugate (using the `algdep` command, one can check that a is a root of $9x^4 + 10x^2 + 9 = 0$ and b is a root of $3x^4 - 2x^2 + 3 = 0$.

Note that when the character is (trivial or) quadratic and defined modulo N/Q the output is always rounded, but otherwise, the eigenvalues are given as approximate complex numbers.

As for the Hecke operators, there exists an `mfatkin` command, whose syntax is `mfatkin(mfatk, F)`, where `mfatk` is the output of an `mfatkininit` command and F is in the space `mfatk`, and which outputs the modular form $F|_k W_Q$, where Q is implicit in `mfatk`.

Finally note the `mfbd` expanding command which computes $B(d)F$:

```
? E4 = mfEk(4); mfcoefs(E4,6)
% = [1, 240, 2160, 6720, 17520, 30240, 60480]
? F = mfbd(E4,2); mfcoefs(F,6)
% = [1, 0, 240, 0, 2160, 0, 6720]
```

9 Algebraic Functions on Modular Forms

Here we give examples of functions on modular forms which do not involve any approximate numerical computation. We have already mentioned the most important ones: `mfhecke`, `mfatkin`, and `mfbd`.

```
? E4 = mfEk(4); F = mfderivE2(E4); mfcoefs(F,5)
% = [-1/3, 168, 5544, 40992, 177576, 525168]
? E6 = mfEk(6); mfisequal(F, mflinear([E6], [-1/3]))
% = 1
? G = mfbracket(E4, E6, 1); mfcoefs(G,5)
% = [0, -3456, 82944, -870912, 5087232, -16692480]
? mfisequal(G, mflinear([mfDelta()], [-3456]))
% = 1
```

In the first commands, we compute the Serre derivative of E_4, and check that it is equal to $-E_6/3$. The name `mfderivE2` of course comes from the fact that the Serre derivative involves the quasi-modular Eisenstein series E_2. Note that there exists the function `mfderiv` (including to negative order, corresponding to integration), which is provided for the user's convenience

for certain computations, but whose output is outside the range of modular forms.

The second computation checks that the first Rankin–Cohen bracket of E_4 and E_6 is a multiple of Δ.

You may complain that it is heavy to write an `mflinear` command as above simply to compute a scalar multiple of a form. But nothing prevents you from defining in a script that you read at the beginning of your session:

```
mfscalmul(F,s)=mflinear([F],[s]);
mfadd(F,G)=mflinear([F,G],[1,1]);
mfsub(F,G)=mflinear([F,G],[1,-1]);
```

There also exist the natural operations on modular forms `mfmul`, `mfdiv` (which may result in modular functions, i.e., with poles), and `mfpow`. There is also a function `mfshift` (multiply or divide by a power of q), but which again takes us outside the range of modular forms.

```
? E4 = mfEk(4); F = mftwist(E4, -3); mfcoefs(F, 7)
% = [0, 240, -2160, 0, 17520, -30240, 0, 82560]
? mfparams(F)
% = [9, 4, 1, y]
? mf = mfinit([4,5,-4], 1); F = mfbasis(mf)[1]; mfcoefs(F, 10)
% = [0, 1, -4, 0, 16, -14, 0, 0, -64, 81, 56]
? mfisCM(F)
% = -4
? G = mftwist(F, -4); mfcoefs(G, 10)
% = [0, 1, 0, 0, 0, -14, 0, 0, 0, 81, 0]
? mfparams(G)
% = [16, 5, -4, y]
? mfconductor(mfinit(G, 1), G)
% = 8
```

This session illustrates a number of important issues concerning *twisting*. In the first commands, we twist E_4 by the quadratic character -3 (in the present implementation, only twisting by quadratic characters is allowed), and we see that the resulting form has level $9 = (-3)^2$. Fine. In the next command, we compute the unique form in $S_4(\Gamma_0(5), \chi_{-4})$, and see that it has CM by $\mathbb{Q}(\sqrt{-4})$.

However, note that the form is not equal to the form twisted by the character χ_{-4} (only the coefficients of q^n with n prime to 4 are equal, the

others vanish). The `mfparams` command tells us that the twisted form has level $16 = (-4)^2$. However, the final command tells us that in fact it has level 8: `mfconductor` gives the smallest level on which the form is defined.

```
? mf = mfinit([96,2], 1); L = mfbasis(mf);
? apply(x->mfconductor(mf,x), L)
% = [24, 48, 96, 32, 96, 48, 96, 96, 96]
? apply(x->mftonew(mf,x)[1][1..2], L)
% = [[24, 1], [24, 2], [24, 4], [32, 1], [32, 3],\
 [48, 1], [48, 2], [96, 1], [96, 1]]
```

Here we compute the full cuspidal space $S_2(\Gamma_0(96))$, of dimension 9, and we ask which is the lowest level on which each form in the basis is defined. This list shows that there is one form F_1 in level 24 which, by applying $B(d)$ with $d = 2$ and $d = 4$ gives a form of level 48 and one of level 96. Then a form F_2 in level 32, by applying $B(3)$ gives a form of level 96, a form F_3 in level 48, by applying $B(2)$ gives a form of level 96, and finally two genuine forms of level 96 (so that the dimension of the newspace is equal to 2, which we can check by typing `mfdim([96,2],0)`).

The last command `mftonew` checks all this; look at the precise description of the command.

10 Cusps and Cosets

Recall that in the present version of the package, the only congruence subgroup that is considered is $\Gamma_0(N)$, so when we consider cusps in the geometrical sense, they are cusps of $\Gamma_0(N)$, and cosets are right cosets of $\Gamma_0(N)$ in Γ, so that $\Gamma = \bigsqcup_j \Gamma_0(N)\gamma_j$.

The function `mfcusps(N)` gives the list of all (equivalence classes of) cusps of $\Gamma_0(N)$, `mfcuspwidth(N,cusp)` gives the width of the cusp; these are linked to the *geometry*. On the other hand, the notion of *regularity* of a cusp is linked to the specific modular form space, and the function `mfcuspisregular([N,k,CHI],cusp)` determines if the cusp is regular or not:

```
? C = mfcusps(108)
% = [0, 1/2, 1/3, 2/3, 1/4, 1/6, 5/6, 1/9, 2/9, 1/12,\
       5/12, 1/18, 5/18, 1/27, 1/36, 5/36, 1/54, 1/108]
? [mfcuspwidth(108,c) | c<-C]
% = [108, 27, 12, 12, 27, 3, 3, 4, 4, 3, 3, 1, 1, 4,\
```

```
       1, 1, 1, 1]
? NK = [108,3,-4];
? [mfcuspisregular(NK,c) | c<-C]
% = [1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1]
? [c | c<-C, !mfcuspisregular(NK,c)]
% = [1/2, 1/6, 5/6, 1/18, 5/18, 1/54]
```

The first command list the 18 cusps of $\Gamma_0(108)$ (`mfnumcusps(108)` gives this directly, useful if there are thousands of cusps and you do not want them explicitly), the second command prints their widths, and the last commands show that the cusps 1/2, 1/6, 5/6, 1/18, 5/18, and 1/54 are irregular in the space $M_3(\Gamma_0(108), \chi_{-4})$, and the others are regular.

There is another command `mfcuspval` having to do with cusps, but this will be mentioned later.

```
? C = mfcosets(4)
% = [[0, -1; 1, 0], [1, 0; 1, 1], [0, -1; 1, 2],\
     [0, -1; 1, 3], [1, 0; 2, 1], [1, 0; 4, 1]]
? mftocoset(4, [1, 1; 2, 3], C)
% = [[-1, 1; -4, 3], 5]
```

The `mfcosets(N)` command lists all right cosets of $\Gamma_0(N)$ in Γ. Note that in the present implementation the trivial coset is always the last one, and is represented by the matrix $[1, 0; N, 1]$, but since this may change one must be careful.

The `mftocoset(N,M,C)` command gives a two-component vector $[\gamma, i]$, where $\gamma \in \Gamma_0(N)$ is such that $M = \gamma \cdot C[i]$.

11 The mfslashexpansion command

We now give examples of the use of advanced features of the package, which use inexact complex arithmetic. However in many cases the results are known algebraic numbers and, if asked to do so, the function gives them exactly.

This command returns the Fourier expansion at infinity of $f|_k\gamma$, for $\gamma \in \mathrm{GL}_2(\mathbb{Q})^+$. It returns a vector v of coefficients, which can only be interpreted together with three extra parameters $\alpha \in \mathbb{Q}_{\geq 0}$, $w \in \mathbb{Z}_{\geq 1}$ and a 2×2 upper triangular matrix $A = [a, b; 0, d]$ (equal to the identity if

$\gamma \in \mathrm{PSL}_2(\mathbb{Z})$). We have $f|_k\gamma = F|_k A$, with

$$F(\tau) = q^\alpha \sum_{n \geq 0} v[n] q^{n/w}$$

and $q = e(\tau)$. Of course, $F|_k A = (a/d)^{k/2} F(\tau + b/d)$ so the exact expansion is easily inferred from the returned one, whereas the chosen encoding allows to compute the coefficients $v[n]$ in a smaller number field than if we had included all the constants into v. It is important to note that the three parameters α, w, A only depend on the modular form space and γ, but not on the form f.

```
? mf = mfinit([4,6]); B = mfbasis(mf);
? for (i=1, #B, \
    print( mfslashexpansion(mf,B[i],[1,0;2,1],5,1,&P) ))
\\ we don't print P which is [0, 1, [1,0;0,1]] in all cases
[-1/504, 1, 33, 244, 1057, 3126]
[-1/504, 0, 1, 0, 33, 0]
[-1/32256, -1/64, 33/64, -61/16, 1057/64, -1563/32]
[0, -1, 0, 12, 0, -54]
? R = mfslashexpansion(mf,B[1],[0,-1;4,0],5,1,&P); [P,R]
% = [[0, 1, [1,0;0,1]], [-8/63, 0, 0, 0, 64, 0]]
? R = mfslashexpansion(mf,B[1],[0,-1;1,0],5,1,&P); [P,R]
% = [[0, 4, [1,0;0,1]], [-1/504, 0, 0, 0, 1, 0]]

? mf=mfinit([4,7,-4]); B=mfbasis(mf);
? for (i=1, #B, \
    print( mfslashexpansion(mf,B[i],[1,0;2,1],5,1,&P) ))
\\ we don't print P which is [1/2, 1, [1,0;0,1]] in all cases
[1/64, 91/8, 7813/32, 7353/4, 530713/64, 221445/8]
[1, -728, 15626, -117648, 530713, -1771560]
[1/16, -15/2, 5/8, 75, -231/16, -465/2]
[2, 0, 20, 0, -462, 0]
? mfslashexpansion(mf,B[1],[0,-1;4,0],5,1,&P)
% = [Mod(61/256*t, t^2 + 1), Mod(-1/64*t, t^2 + 1),
     Mod(-1/64*t, t^2 + 1), Mod(91/8*t, t^2 + 1),
     Mod(-1/64*t, t^2 + 1), Mod(-7813/32*t, t^2 + 1)]
? P
% = [0, 1, [1, 0; 0, 1]]
? R=mfslashexpansion(mf,B[1],[0,-1;4,0],5,0)
% = [0.23828125000000000000000000000000000000*I,\
```

```
    -0.015625000000000000000000000000000000000*I,\
    -0.015625000000000000000000000000000000000*I,\
    11.375000000000000000000000000000000000000*I,\
    -0.015625000000000000000000000000000000000*I,\
  -244.156250000000000000000000000000000000*I]
? bestappr(R)
% = [61/256*I, -1/64*I, -1/64*I, 91/8*I, -1/64*I, -7813/32*I]
```

Here are some detailed explanations. The first space is $M_6(\Gamma_0(4))$, of dimension 4. We ask for $1+5$ terms of the Fourier expansion of $F|_6\gamma$ for all F in the given basis, and $\gamma = [1,0;2,1]$, which is one of the possible Fourier expansions at the cusp $1/2$. The last parameter P contains $[\alpha, w, A]$ after the call and the 1 means we want an exact algebraic expression (a 0 as used in the last example means we expect floating point complex numbers).

We obtain the 4 desired expansions. In the next commands, we do the same for the first basis element and $\gamma = [0,-1;4,0]$, which is the *Fricke involution*, and corresponds to the cusp 0. The next command, which does essentially the same computation, uses $\gamma = [0,-1;1,0]$, and now $P = [0,4,[1,0;0,1]]$ which tells us that the expansion is in powers of $q^{1/4}$.

The next example is the space $M_7(\Gamma_0(4),\chi_{-4})$, also of dimension 4, and we ask the same thing. Now P tells us that $\alpha = 1/2$ and $w = 1$ so we now have for instance

$$B[1]|_7\gamma = (1/64)q^{1/2} + (91/8)q^{3/2} + \cdots$$

In the next command, we expand $B[1]|_7\gamma$ with γ equal to the Fricke involution; and finally we check numerically by setting the one-to-last parameter to 0 and obtain the expansion as raw complex numbers. We recognize them immediately using the `bestappr` command.

Note that in the special case (like here) where γ is a Fricke (or more generally an Atkin–Lehner) involution, we can proceed otherwise to obtain the expansion:

```
? mfatk = mfatkininit(mf,4); C = mfatk[3]
% = -1.0000000000000000000000000000*I
? F = mfatkin(mfatk, B[1]); mfcoefs(F, 6)
% = [61/256, -1/64, -1/64, 91/8, -1/64, -7813/32, 91/8]
```

This tells us that the true expansion of $F|_7W_4$ is the expansion that is output divided by the constant C, so we recover the previous expansion.

It is important to see what affects the timing and correctness of the `mfslashexpansion` command. The following session gives typical examples:

```
? mf = mfinit([496,4],0); F = mfbasis(mf)[1]; mfdim(mf)
time = 329 ms.
% = 45
? mfslashexpansion(mf,F,[1,0;3,1],5,0,&P);
time = 1,316 ms.
? mfslashexpansion(mf,F,[1,0;3,1],5,1,&P);
time = 51,136 ms.

? mf = mfinit([503,4],0); F = mfbasis(mf)[1]; mfdim(mf)
time = 1,505 ms.
% = 125
? sizebyte(mf)
% = 5123352
? mfslashexpansion(mf,F,[1,0;3,1],5,0,&P);
time = 32,640 ms.
? sizebyte(mf)
% = 18216400
? mfslashexpansion(mf,F,[1,0;3,1],5,0,&P);
time = 6,504 ms.
```

We omit the numerical outputs since they have no significance for the present discussion. We notice several things:

- First, the time for rationalization (flag 1) in the first example is extremely large: 51 seconds instead of 1.3. The reason for this is that the width of the corresponding cusp (here 1/3) is equal to $P[2] = 496$, and the program must recognize algebraic numbers in the large cyclotomic field $\mathbb{Q}(\zeta_{496})$ which takes a huge amount of time. In fact, at the default accuracy of $38D$, the result is certainly wrong.

- Second, the time depends enormously on the dimension: the expansion for dimension 125 is 25 times slower than for dimension 45, of course not surprising, but it must be taken into account.

- Third, and most importantly, the last command shows the cache effect: exactly the same instruction now requires only 6.5 seconds instead of 32.6. This is because, behind the scenes, the first `mfslashexpansion` precomputed a number of quantities which it stored in your variable

`mf`: in fact, the `sizebyte` commands show that, after the first expansion, the size of `mf` has more than tripled.

12 Analytic Commands

The existence of the `mfslashexpansion` command allows us to do many useful things. In fact, already the `mfatkininit` and `mfatkin` commands would not be possible without it. Immediate applications are the `mfcuspval` command which computes the valuation at cusps, and the `mfeval` command, which in addition to computing values in the upper-half plane (see below), also computes values at the cusps:

```
? T = mfTheta(); mf=mfinit(T);C=mfcusps(mf)
% = [0, 1/2, 1/4]
? [ mfcuspval(mf,T,c) | c<-C ]
% = [0, 1/4, 0]
? mfeval(mf, T, C) \\ or [mfeval(mf,T,c) | c<-C]
% = [1/2 - 1/2*I, 0, 1]
```

More sophisticated is the computation of numerical periods, and more generally of *symbols*

$$\int_{s_1}^{s_2} (X-\tau)^{k-2} F|_k\gamma(\tau)\, d\tau \ ,$$

where s_1 and s_2 are two cusps (e.g., $s_1 = 0$, $s_2 = \infty$):

```
? mf = mfinit([96,4],0); [F1] = mfbasis(mf);
? FS1 = mfsymbol(mf,F1);
time = 2,272 ms
? mfsymboleval(FS1,[0,oo])
% = 2.0968669678226579060336519703627002478*I*x^2\
  + 0.3636858065631763556844427744284294007 3*x\
  - 0.0493157368347131091382972119865106437 80*I
? mfsymboleval(FS1,[1,5/2])
% =  4.1937339356453158120673039407254004956*I*x^2\
  + (0.7273716131263527113688855488568588014 7\
  - 14.678068774758605342235563792538901735*I)*x\
 + (-1.2729003229711172448955497104995029026\
  + 15.103654043044843600467382361156555509*I)
? mfsymboleval(FS1,[1,2],[0,-1;1,0])
```

```
% = (0.54552870984476453352666416164264410111\
   + 2.52245223610889616426547053898035402 22*I)*x^2\
 + (-0.72737161312635271136888554885685880148\
   - 6.29060090346797371810095591108810074 34*I)*x\
   + 4.19373393564531581206730394072540049 56*I
```

The general strategy for computing these quantities is first to do a precomputation which only involves `mf` and the form F using `mfsymbol`, which can take a few seconds, but afterwards all the computations are instantaneous.

Note that if you only want the period polynomial from 0 to ∞ use `mfperiodpol(mf,F1)` which gives the same answer as before but in only 20 ms.

You may also use `mfsymboleval` in two other ways, but note that in this case the precomputation is not used so the computation may be slow:

```
? mf=mfinit([96,6],0);F=mfbasis(mf)[1];
? FS=mfsymbol(mf,F);
time = 9,761 ms.
? mfsymboleval(FS,[I,oo])
% = 0.0029721...*I*x^4 + 0.0137806...*x^3 + ... + 0.0061009...
? mfsymboleval(FS,[I,2*I])
% = 0.0029665...*I*x^4 + 0.0137326...*x^3 + ... + 0.0059760...
? mfsymboleval(FS,[I/10000,I])
% = 46.363730...*I*x^4 + 3.8815894...*x^3 + ... + 0.0183869...
? -x^4*subst(mfsymboleval(FS,[I,10000*I],[0,-1;1,0]),x,-1/x)
% = 46.363730...*I*x^4 + 3.8815894...*x^3 + ... + 0.0183869...
? mfsymboleval([mf,F],[I,oo])
% = 0.0029721...*I*x^4 + 0.0137806...*x^3 + ... + 0.0061009...
? mfsymboleval([mf,F],[I,2*I])
% = 0.0029665...*I*x^4 + 0.0137326...*x^3 + ... + 0.0059760...
```

These examples illustrate four points:

1. Computing an `mfsymbol` may be rather long (9.8 seconds in this example), although as already mentioned, subsequent computations of symbols *between cusps* will then be instantaneous.

2. As the next three commands show, `mfsymboleval` also accepts paths with endpoints in the upper half-plane. Although we have tried to optimize the computation, in certain cases (but not in the above example)

when one of the endpoints is close to the real line the computation may be slow.

3. The next command shows the use of the extra parameter γ which asks to integrate $F|_k\gamma$ instead of F, here with `ga=[0,-1;1,0]`. This allows to perform the same computation with endpoints which are away from the real line. This is essentially what is done *automatically* by `mfsymboleval`.

4. The last two commands show a special format which avoids doing the longish `mfsymbol` computation: the results are obtained almost instantaneously *without* using symbols. The price to pay in using this "cheaper" format is that the endpoints of the path cannot be cusps other than `oo`.

```
? mf = mfinit([96,4],0); [F1,F2] = mfbasis(mf);
? FS1 = mfsymbol(mf,F1); FS2 = mfsymbol(mf,F2);
? mfpetersson(FS1)
% = 0.000614716841498177889240915163025173918 26
? mfpetersson(FS2)
% = 0.005532451573483601003168236467226565264 7
? mfpetersson(FS1, FS2)
% = 1.5879887877319313665 E-40 + 7.652958013165934297 E-42*I
```

Same remark: once the `mfsymbols` `FS1` and `FS2` initialized, all the Petersson product computations (as well as others) are essentially immediate. Note that since neither `F1` nor `F2` are eigenforms, there is no reason for their Petersson product to vanish. To prove it does we do as follows:

```
? BE = mfeigenbasis(mf);
? M = Mat([mftobasis(mf,f) | f<-BE]); M^(-1)
% =
[1  3  10   4 -20  70]

[1  3   2  12  60 -42]

[1  3 -14 -36 -36  54]

[1 -3  10  -4  20  70]

[1 -3   2 -12 -60 -42]
```

```
[1 -3 -14  36  36  54]
```

On the other hand, it is immediate to see that `BE[i+3]` is a twist of `BE[i]` and that as a consequence their Petersson square are equal. It follows from the shape of the above matrix that the Petersson scalar product of $B[i]$ with $B[j]$ will vanish when the corresponding scalar product of the corresponding columns vanish, hence for $(i,j) = (1,2)$, $(1,4)$, $(1,5)$, $(2,3)$, $(2,6)$, $(3,4)$, $(3,5)$, $(4,6)$, and $(5,6)$.

Note that `mfpetersson` can also be used for two noncuspidal forms, as long as the Petersson product converges. Consider the following example:

```
? mf = mfinit([12,5,-3]); cusps = mfcusps(mf);
? E1 = mfeisenstein(5,1,-3); [mfcuspval(mf,E1,c) | c<-cusps]
% = [0, 0, 1, 0, 1, 1]
? E2 = mfeisenstein(5,-3,1); [mfcuspval(mf,E2,c) | c<-cusps]
% = [1/3, 1/3, 0, 1/3, 0, 0]
? P(mf) =
  { my(E1S = mfsymbol(mf,E1));
    my(E2S = mfsymbol(mf,E2));
    mfpetersson(E1S,E2S); }
? P(mf)
% = -1.8848216716468969562647734582232071466 E-5\
     - 1.9057659114817512165 E-43*I
? mf3 = mfinit([3,5,-3]); P(mf3)
time = 16 ms.
? mf96 = mfinit([96,5,-3]); P(mf96)
time = 3,521 ms.
```

The first commands create two Eisenstein series of weight 5, $E_5(1,\chi_{-3})$ and $E_5(\chi_{-3},1)$, which belong to $M_5(\Gamma_0(3),\chi_{-3})$. In the next commands, we look at the larger space of level 12 and compute the valuations of E_1 and E_2 at the six cusps of $\Gamma_0(12)$. We see that at these six cusps one of the two Eisenstein series vanishes, so the Petersson product will converge, and is computed in the next command. In the last commands we compute the same product but in level 3 and level 96; because of the normalization, we obtain essentially the same result (not given), but of course the times are very different: 0.016 seconds in level 3 and 3.5 seconds in level 96.

There are two more important numerical functions: evaluating a modular form at a point in the upper half plane, and evaluating the corresponding L-function. We begin by a trivial example:

```
? E4 = mfEk(4); mf = mfinit(E4); mfeval(mf,E4,I)
% = 1.4557628922687093224624220035988692874
? 3*gamma(1/4)^8/(2*Pi)^6
% = 1.4557628922687093224624220035988692874
```

This is of course a trivial computation, simply sum the q-expansion. The fact that the value of a modular form with rational coefficients such as E_4 at a *CM point* such as i has an explicit expression is a consequence of complex multiplication.

```
? mf = mfinit([12,4],1); F = mfbasis(mf)[1];
? mfeval(mf, F, 1/Pi + 10^(-6)*I)
% = -89811.049350396250531782882568405506024\
   - 58409.940965200894541585402642924371696*I
? mfeval(mf, F, 1/Pi + 10^(-7)*I)
% = 4.821246850466111318325339669181329226 1 E-52\
  + 6.788526228152064790887124754156141534 0 E-52*I
```

Several remarks are in order.

1. We are evaluating a modular form very near the real axis. If the form was in level 1 such as E_4 above, we could use a modular transformation to reduce to the evaluation in the fundamental domain of Γ, which would be very fast. Here we do something similar but more sophisticated.

2. Contrary to most examples, the result at height 10^{-7} is not a numerical approximation of 0, the exact value is indeed as printed to the given accuracy.

3. It is amusing to see the large oscillations of the value: at height 10^{-6} the value is still in the 10^5 range, and at 10^{-7} it is in the 10^{-52} range. Of course it must eventually tend to 0 since F is a cusp form (for E_4 it would tend to infinity).

4. When applying `mfeval` at a *cusp* (as above for `mfTheta()`), the result is the value at the cusp, but is in general *not* equal to the limit of the value of the modular form when the argument tends to the cusp, since this limit is often infinite for a noncuspidal form.

Note that when dealing with *eigenforms*, which may have several embeddings into $\mathbb{C}$, the result will have several components, one for each embedding:

```
? mf = mfinit([23,2],0); F=mfeigenbasis(mf)[1];
? mfeval(mf,F,I)
% = [0.0018695834459685012330841605500720163964,\
     0.0018618146628840767703527958851699552194]
```

More generally, this embedding problem affects all numerical functions. Continuing the above example:

```
? mfparams(F)
% = [23, 2, 1, y^2 - y - 1]
? mfslashexpansion(mf,F,[0,-1;1,0],5,1)
% = [0, -1/23, 1/23*y, -2/23*y + 1/23, -1/23*y + 1/23, 2/23*y]
? FS = mfsymbol(mf,F); mfpetersson(FS,FS)
% =
[0.00394889657400250316885... -1.0827196147167250830 E-40]

[-1.2120247024777595243 E-40 0.00564425429876478351015...]
```

The y in the second result is thus understood to be *one* of the roots of the polynomial $y^2 - y - 1$, and the result of `mfpetersson` is a 2×2 *diagonal* matrix because of the two embeddings of F.

The other important evaluation function is that of the L-function attached to a modular form. In fact, the modular form package only creates (in a clever way) the L-function, all the rest of the work is done by the L-function package. Note the important fact that the modular form need not be an eigenform or even stable under the Fricke involution.

```
? E4 = mfEk(4); mf=mfinit(E4); LE = lfunmf(mf,E4);
? lfun(LE, 2) / Pi^2
% = -3.3333333333333333333333333333333333333
? lfun(LE, 0)
% -1
? D = mfDelta(); mf=mfinit(D); LD = lfunmf(mf,D);
? lfunlambda(LD, 3)/lfunlambda(LD, 5)
% = 1.5555555555555555555555555555555555556
? lfunlambda(LD, 1)/lfunlambda(LD, 3)
% = 2.3444283646888567293777134587554269175
? bestappr(%)
% = 1620/691
? mf = mfinit([23,2],0); F = mfbasis(mf)[1]; L = lfunmf(mf,F);
```

```
? lfun(L, 2)
% = 1.5959983753450272580976413437480171832
? G = mfeigenbasis(mf)[1]; M = lfunmf(mf,G);
? apply(x->lfun(x,I),M)
% = [-0.158560333732547406573278445796721556 64\
    + 0.796713699225048183776026803446863119 69*I,\
     -0.102302788165090239089937756630307120 37\
    + 0.659542239830925832877845222682952995 13*I]
```

Note that the constant term $a(0)$ is ignored by the L-function, but can be recovered thanks to the formula $a(0) = -L(F, 0)$.

The last commands illustrate first the fact that the L-functions can be computed for non-eigenforms (F is not an eigenform), and second that if there are several embeddings, the `lfunmf` function returns a vector of `lfunmf`, one for each embedding.

Another illustration of the L-function package:

```
? LIN = lfuninit(LD, [6, 6, 50]);
? ploth(t = 0, 50, lfunhardy(LIN, t))
```

13 The mfeigensearch and mfsearch commands

The last commands that we want to illustrate are *searching* commands, The idea is simple: you believe that you have a modular form, but you do not know its level, weight, character, or field of definition of its coefficients, but only a number of its Fourier coefficients, perhaps only modulo p, and you would like to find forms which "match" your given form.

In this degree of generality, the search space is too wide. We have therefore decided to reduce the generality, so as to make the search more reasonable. Note that this will probably vary with the different versions of the program, so what is described here may be more restrictive than future versions. In the present implementation, we assume that the form we are looking for has rational coefficients, so that its character is (trivial or) quadratic.

The `mfsearch` command does this naively but is likely to be more efficient than taylor-made scripts:

```
? V = mfsearch([60,2],[0,1,2,3,4,5,6], 1); #V
time = 5 ms.
```

```
% = 3
? V = mfsearch([[1..60],2],[0,1,2,3,4,5,6], 1); #V
time = 40 ms.
% = 5
? [ mfparams(f) | f<-V ]
% = [[56, 2, 8, y], [58, 2, 1, y],
     [60, 2, 1, y], [60, 2, 12, y], [60, 2, 60, y]]
? [ print(mfcoefs(f,10)) | f<-V ]
[0, 1, 2, 3, 4, 5, 6, -6, -4, -7, -20]
[0, 1, 2, 3, 4, 5, 6, -34, 37, 22, 7]
[0, 1, 2, 3, 4, 5, 6, 20, 0, -27, -6]
[0, 1, 2, 3, 4, 5, 6, -170/9, -272/9, -11/3, -134/9]
[0, 1, 2, 3, 4, 5, 6, 200/13, -304/13, -435/13, -278/13]
```

This command looks for all forms, first in level 60 then in level 1 to 60 and weight 2 whose first coefficients are $[0, 1, 2, 3, 4, 5, 6]$, the final 1 is optional and specifies the *space* (in `mfinit` sense) where the search is performed, here the cuspidal space (by default the full space). It returns a list of 3 forms in level 60 and 5 in total.

The `mfeigensearch` command is more interesting. We look for is a cuspidal *eigenform* whose field of definition is $\mathbb{Q}$, so that its Fourier coefficients are integers, and its character is (trivial or) quadratic. An example is as follows:

```
? AP = [[2,2], [3,-1]]  \\ a(2) = 2 and a(3) = -1
? L = mfeigensearch([[1..120],4], AP); #L
% = 2
? [f,g] = L; [mfparams(f), mfparams(g)]
% = [[26, 4, 1, y], [118, 4, 1, y]]
? mfcoefs(f, 10)
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
? mfcoefs(g, 10)
% = [0, 1, 2, -1, 4, -13, -2, -27, 8, -26, -26]
```

The first command asks for all forms as above in weight 4 and level from 1 up to 120, such that $a(2) = 2$ and $a(3) = -1$. The answer is that there are two forms, which we call f and g. We compute their levels (26 and 118 respectively), notice they have trivial character, and list their Fourier coefficients up to 10 and we see that indeed $a(2) = 2$ and $a(3) = -1$ in both cases.

To specify the coefficients that we want there are a number of ways. The simplest, as above, is to give the list of pairs of integers $[p, a(p)]$. For instance:

```
? L = mfeigensearch([[1..80],2], [[2,2], [7,-3]]); #L
% = 1
? F = L[1]; mfparams(F)
% = [75, 2, 1, y]
? mfcoefs(F, 12)
% = [0, 1, 2, -1, 2, 0, -2, -3, 0, 1, 0, 2, -2]
```

The coefficient $a(p)$ may also be given as an `intmod` `Mod`(a, m) then one looks for a match for $a(p)$ modulo m. For instance, we come back to our first example:

```
? AP5 = [[2,Mod(2,5)], [3,Mod(-1,5)]]; \\ now modulo 5
? L=mfeigensearch([[1..120], 4], AP); #L
% = 3
? [ mfparams(f)[1] | f <- L ]
% = [26, 26, 118]
? [F1,F2] = L;  \\ let's consider the first two
? mfcoefs(F1, 10)
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
? mfcoefs(F2, 10)
% = [0, 1, 2, 4, 4, -18, 8, 20, 8, -11, -36]
? F = mflinear([F1, F2], [-1, 1]);
? content(mfcoefs(F, mfsturm([26,4])+1))
% = 5
```

Working modulo 5, we now find that there is an extra eigenform satisfying our criteria, and perhaps surprisingly, again in level 26. The first, `F1`, is the one found above, with $a(2) = 2$ and $a(3) = -1$. The second, `F2`, has $a(2) = 2$ but $a(3) = 4 \equiv -1 \pmod 5$.

But we can go further and see that this is not a simple coincidence: the next command shows that both eigenforms seem to be congruent modulo 5, at least up to $a(10)$. In fact they are indeed congruent modulo 5: to prove this, we use the fact that the basic Sturm bound (the one obtained using `mfsturm([N,k])`, not `mfsturm(mf)`) is also valid modulo p. Since all coefficients are congruent up to the Sturm bound, they are congruent for all n.

14 Half-Integral Weight Functions

14.1 General Functions

Many of the commands that we have seen, and most importantly the `mfinit` and `mfdim` command, can be used verbatim in the case of modular forms of half-integral weight, sometimes with small differences.

- Two functions created from mathematical objects can give forms of half-integral weight, `mfetaquo` and `mffromqf`.
- Leaf functions created from scratch are `mfTheta`, which gives the standard Jacobi theta function of weight 1/2, and `mfEH`, which gives the Cohen–Hurwitz Eisenstein series of half-integral weight.

```
? F = mffrometaquo([2,5;1,-2;4,-2]); Ser(mfcoefs(F,10),q)
% = 1 + 2*q + 2*q^4 + 2*q^9 + O(q^11)
? T = mfTheta(); mfisequal(F,T)
% = 1
? F = mffromqf(2*matid(3))[2]; Ser(mfcoefs(F,5),q)
% = 1 + 6*q + 12*q^2 + 8*q^3 + 6*q^4 + 24*q^5 + O(q^6)
? mfisequal(F, mfpow(T,3))
% = 1
```

- The only spaces which are *directly* available by `mfinit` and `mfdim` are the full cuspidal space and the full modular form space. The new space can be defined in some cases but indirectly, using Kohnen's theory, see below.
- The only Hecke operators $T(n)$ which are nonzero are those where n is a square (we have not programmed the $T(p)$ with p dividing the level).

14.2 Specific Functions

The most important specific function in half-integral weight is `mfshimura`, which computes the Shimura lift corresponding to a discriminant D (1 by default) and also returns an `mf` space containing the lift:

```
? mf=mfinit([60,5/2],1); F=mfbasis(mf)[1];
? D = [1,5,8,12,13,17,21];
? for (i=1, #D, \
```

```
    [mf2,G] = mfshimura(mf,F,D[i]); print(mfcoefs(G,10)))
[0, 1, 2, 0, 2, -1, -2, 6, 6, -3, -10]
[0, 0, 0, -1, 0, 0, 20, 0, 0, -2, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 24, 0, 0, 0, 0, -48]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 3, 52, -5, -120, 14, -156, 3, 0]
? mfdescribe(mf)
% = "S_5/2(G_0(60, 1))"
? mfdescribe(mf2)
% = "S_4(G_0(30, 1))"
```

Two things to notice: first the image can be identically 0. Second, the program takes some time (20 seconds for the above), because computing a Shimura image takes time proportional to D^4.

The other specific functions are related to the Kohnen +-space. Continuing the above example:

```
? K=mfkohnenbasis(mf); matsize(K)
% = [14, 4]
? K[,1]
% = [-1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]~
? F=mflinear(mf,K[,1]);
? Ser(mfcoefs(F,35),q)
% = -q + 2*q^4 - 4*q^16 + 3*q^21 - 6*q^24 + 5*q^25 + O(q^36)
```

The first command shows that although the dimension of the cuspidal space is 14, that of the Kohnen +-space is 4; if desired, the corresponding modular forms can be obtained by `mflinear(mf,K[,j])` for each j as done in the next command. The Fourier expansion of F given on the last line shows that the only nonzero coefficients of q^n occur when $n \equiv 0, 1 \pmod 4$. Continuing:

```
? [mf2,FS]=mfshimura(mf,F); mfparams(FS)
% = [15, 4, 1, y]
? [mf2,FS]=mfshimura(mf,mfbasis(mf)[1]); mfparams(FS)
% = [30, 4, 1, y]
```

These commmands show that the image of an element of the Kohnen +-space has level $15 = 60/4$, while the second shows that the image of a random form has level $30 = 60/2$.

A final command related to the Kohnen +-space is `mfkohnenbijection`. This allows, in half-integral weight, to compute the new space, its splitting, and the eigenforms:

```
? [mf3,M,K,shi] = mfkohnenbijection(mf);
? M * mfheckemat(mf3,11) * M^(-1)
% =
[ 48  24  24  24]

[  0  32   0 -20]

[-48 -72 -40 -72]

[  0   0   0  52]
? mf30 = mfinit(mf3,0); B0 = mfbasis(mf30); #B0
% = 2
? BNEW = [mflinear(mf, K * M * mftobasis(mf3,f)) | f<-B0];
? BE = mfeigenbasis(mf30);
? BEIGEN = [mflinear(mf, K * M * mftobasis(mf3,f)) | f<-BE ];
? Ser(mfcoefs(BEIGEN[1],24),q)
% = q + q^4 - 3*q^9 - 5*q^16 + 6*q^21 + 3*q^24 + O(q^25)
? Ser(mfcoefs(BEIGEN[2],24),q)
% = q^5 + q^8 - 3*q^12 - 4*q^17 + 3*q^20 + O(q^25)
? mfcoefs(BEIGEN[1],10^4);
time = 7,532 ms.
```

The `mfkohnenbijection` command computes a square matrix M giving a Hecke-module isomorphism from the space $S_{2k-1}(\Gamma_0(N),\chi^2)$ to the Kohnen +-space $S_k^+(\Gamma_0(4N,\chi))$. Note that this makes sense only when N is squarefree.

Thus, M allows to transport all problems from the "difficult" space $S = S_k^+(\Gamma_0(4N),\chi)$ to the "easy" space $S_{2k-1}(\Gamma_0(N),\chi^2)$. For instance, the next command (essentially instantaneous) gives the matrix of the Hecke operator $T(121)$ on S; a direct implementation using the action of $T(121)$ would take 10.6 seconds.

The vector `BNEW` computed afterwards gives a basis of the Kohnen new space $S_k^{+,\mathrm{new}}(\Gamma_0(4N),\chi)$, here of dimension 2.

The vector `BEIGEN` computed in a similar way contains the eigenfunctions of this new space. The example of `BEIGEN[2]` shows that, contrary to the integral weight case, these eigenfunctions can have vanishing coefficient of

q^1. Note that we *know* by construction that the image of `BEIGEN[j]` by any Shimura lift is a multiple of `BE[j]` (with the same index j).

The above construction of the new space and the eigenforms being so useful, a specific function exists for this purpose: instead of all the above, simply write `[mf30,BNEW,BEIGEN]=mfkohneneigenbasis(mf,bij)`. Here `BNEW` and `BEIGEN` will be matrices whose columns are the coefficients of a basis of the Kohnen new space and of the eigenforms respectively, and `mf30` the corresponding new space of integral weight.

15 Reference Manual for the Package

We give a brief description in alphabetical order of all the functions specific to the package. To use the package, it is sometimes necessary to use functions on characters or functions of the `lfun` package, but those will not be described here.

Note that when a modular form F can be embedded in $\mathbb{C}$ in several ways (typically for eigenforms), some functions give a vector (or even a matrix for bilinear operations) of results, one for each embedding: this occurs specifically for `lfunmf`, `mfeval`, `mfmanin`, `mfpetersson`, `mfsymboleval`. This will not always be specified.

`getcache()`: returns technical information about auto-growing caches.

`lfunmf(mf,{`F`})`: creates the L-function associated to F, for use in the `lfun` package, where F need not be an eigenform. If F is omitted, output all L-functions associated to the eigenforms. If F (or the eigenforms) have several embeddings in $\mathbb{C}$, output the vector of the corresponding `lfunmf`.

`mfatkin(mfatk, F)`: computes $F|_k W_Q$, where $Q \| N$, where `mfatk` must have been initialized by `mfatk=mfatkininit(mf,Q)`.

`mfatkineigenvalues(mf, Q)`: `mf` being a cuspidal or new space and Q a primitive divisor of N, output the vector of Atkin–Lehner eigenvalues or pseudo-eigenvalues for each Galois eigenspace.

`mfatkininit(mf,Q)`: initialization function for the `mfatkin` function. The output is `[mfb,M,C,mf]`, where C is a complex constant, M/C is the matrix of the Atkin–Lehner operator W_Q from the space `mf` to the space `mfb` (set equal to 0 if equal to `mf`). The matrix M is guaranteed to be with exact coefficients (rational or `polmods`).

`mfbasis(mf,{space` $= 4$`})`: gives a basis of the space of modular forms `mf`, either output by an `mfinit` command, in which case `space` is ignored, or `mf=[N,k,CHI]` (use `mfeigenbasis` for the eigenforms).

`mfbd(F,d)`: gives $B(d)(F)$, $B(d)$ expanding operator.

`mfbracket(F,G,{`$m=0$`})`: mth Rankin–Cohen bracket of F and G.

`mfcoef(F,n)`: nth Fourier coefficient $a(n)$ of F.

`mfcoefs(F,n)`: vector $[a(0), a(1), ..., a(n)]$ of the Fourier coefficients of F up to n. If F is a modular form *space*, give the matrix whose columns are the vectors of the Fourier coefficients of the basis.

`mfconductor(mf,F)`: smallest M such that F belongs to $M_k(\Gamma_0(M), \chi)$.

`mfcosets(N)`: list of right cosets of Γ modulo $\Gamma_0(N)$. In the present implementation, the trivial coset is the last and represented by the matrix $[1, 0; N, 1]$. N can also be an `mf`.

`mfcuspisregular(NK,cusp)`: `NK` being $[N, k, \chi]$ or an `mf`, determine if the cusp is regular or not.

`mfcusps(N)`: list of cusps of $\Gamma_0(N)$. N can also be an `mf`.

`mfcuspval(mf,F,cusp)`: valuation of modular form F at `cusp`, which can be a rational number or `oo`.

`mfcuspwidth(N,cusp)`: width of `cusp` in $\Gamma_0(N)$. N can also be an `mf`.

`mfDelta()`: Ramanujan's Delta function of weight 12.

`mfderiv(F,{`$m=1$`})`: mth derivative $q.d/dq$ of F, where m can be negative, corresponding to integration (the constant term is then set to 0 by convention). The result is only quasi-modular.

`mfderivE2(F,{`$m=1$`})`: mth Serre derivative $q.d/dqF - kE_2F/12$.

`mfdescribe(F,{&`G`})`: F being a modular form or a modular form space, gives a human-readable description of F. If the address of G is given, put in it the vector of parameters of the outmost operator defining F (empty vector if F is a leaf or a modular form space).

`mfdim(mf,{space =` 4`})`: dimension of the space `mf`, where `mf` can also be of the form $[N, k, \chi]$ in which case `space` is taken into account. `mf` can also be of the form $[N, k, 0]$, where 0 is a wildcard, in which case it gives detailed information for each character χ for which the corresponding space of level N, weight k and given character is nonzero: each result is of the form `[order,Conrey,dim,dimdih]`, where `Conrey` is the Conrey label for the character, `order` is its order, `dim` is the dimension of the corresponding space, and `dimdih`, which is computed only in weight 1, is the dimension of the subspace of dihedral forms.

`mfdiv(F,G)`: division of `F` by `G`.

`mfEH(k)`: k being half-integral, gives the Cohen–Eisenstein series of weight k on $\Gamma_0(4)$.

`mfeigenbasis(mf)`: `mf` containing the new space, gives (in some order) the basis of normalized eigenforms.

`mfeigensearch(NK,AP)`: search for normalized eigenforms with integer coefficients in spaces specified by `NK`, satisfying conditions satisfied by `AP`. `NK` is a pair $[N,k]$, the search being in level N and weight k with trivial or quadratic character; the parameter N may be replaced by a vector of allowed levels. `AP` is a list of pairs $[[p_1,a(p_1)],...,[p_n,a(p_n)]]$, where $a(p)$ is either an integer or an `intmod` (match modulo $a(p)$.`mod`).

`mfeisenstein(k,{`χ_1`},{`χ_2`})`: Eisenstein series $E_k(\chi_1)$ or $E_k(\chi_1,\chi_2)$, normalized so that $a(1)=1$ (so `mfeisenstein(k)` without any character argument is equal to `mfEk(k)` multiplied by $-B_k/(2k)$).

`mfEk(k)`: Eisenstein series E_k for the full modular group normalized so that $a(0)=1$, including for $k=2$.

`mfeval(mf,F,vtau)`: evaluation of F at the point `vtau` (or a vector of points) in the completed upper half-plane. If F is an eigenform with several embeddings in $\mathbb{C}$, evaluate at each embedding.

`mffields(mf)`: `mf` containing the new space, gives the list of relative polynomials defining the number field extensions for all the Galois orbits of the eigenforms. `mf` can also be a modular form, in which case the result is the number field extension of $\mathbb{Q}(\chi)$ in which the Fourier coefficients of `mf` lie.

`mffromell(e)`: `e` being an elliptic curve defined over $\mathbb{Q}$ in `ellinit` format, gives `[mf,F,coe]`, where `F` is the eigenform corresponding to `e` by modularity, `mf` the corresponding new space, and `coe` the coefficients of `F` on the basis of `mf`.

`mffrometaquo(eta,{flag = 0})`: `eta` being a matrix representing an eta quotient, gives the corresponding modular form or function. If the result is not a modular form or function, return an error if `flag=0`, or 0 otherwise. If the result has negative valuation, normalize to valuation 0.

`mffromlfun(L)`: `L` being the L-function of a self-dual modular form with rational coefficients, for instance a rational eigenform, retun `[NK,space,v]`, where `mf = mfinit(NK,space)` is a modular form space containing the form and `mftobasis(mf,v)` yields the coefficients of `F` on the basis of `mf`.

`mffromqf(Q,{`P`})`: `Q` being an even integral quadratic form of even dimension and `P` an optional homogeneous spherical polynomial with respect to `Q`, gives `[mf,F,coe]`, where `F` is the theta function associated to `Q` and `P`, `mf` the corresponding space, and `coe` the coefficients of `F` on the basis of `mf`.

`mfgaloistype(mf,{`F`})`: `mf` being either $[N, 1, \chi]$ or a new or cuspidal space of weight 1 forms, outputs the type of the projective representations attached to all the eigenforms in `mf`, or only that of `F` if it is given. The output is $2n$ for D_n, or -12, -24, -60 for A_4, S_4, A_5.

`mfhecke(mf,F,n)`: Computes $T(n)(f)$, where $T(n)$ is the nth Hecke operator. Note that the level which is used is that of the modular form space `mf`, not that of F if it is different.

`mfheckemat(mf,n)`: matrix of $T(n)$ on the space `mf`.

`mfinit(NK,{space` $= 4$`})`: create the space of modular forms associated to $NK = [N, k, \chi]$ or $NK = [N, k]$. Codes for `space` is 0, new space, 1 cuspidal space, 2 old space, 3 space of Eisenstein series, 4 full space M_k (default). NK can also be of the form $NK = [N, k, 0]$, where 0 is a wildcard, in which case it gives the vector of all nonzero `mfinit` for each Galois orbit of characters χ.

`mfisCM(F)`: returns 0 if F does not have complex multiplication, and the CM discriminant(s) if it does. Note that in weight 1 F may have two CM discriminants, which occurs iff its galoistype is D_2.

`mfisequal(F,G,{lim` $= 0$`})`: Are F and G equal, or at least are their first `lim+1` Fourier coefficients equal ?

`mfkohnenbasis`(mf): `mf` being a cuspidal space of half-integral weight and level $4N$ with N squarefree, computes a basis B of the Kohnen +-space as a matrix whose columns are the coefficients of B on the basis of `mf`.

`mfkohnenbijection`(mf): `mf` being a cuspidal space of half-integral weight, computes `[mf2,M,K,shi]`, where `M` is a matrix giving a Hecke-module isomorphism from the cuspidal space `mf2` of weight $2k - 1$ and level N to the Kohnen +-space of weight k and level $4N$, the columns of the matrix `K` are the coefficients of the Kohnen +-space on the basis of `mf`, and `shi` gives technical information about which linear combination of Shimura lifts has been chosen.

`mfkohneneigenbasis`(mf,bij): `mf` being a cuspidal space of half-integral weight and `bij` the output of `mfkohnenbijection(mf)`, computes a triple `[mf0,Bnew,Beigen]`, where `Bnew` and `Beigen` are matrices whose columns

are the coefficients of a basis of the Kohnen new space and of the eigenforms on the basis of `mf` respectively, and `mf0` is the corresponding new space of integral weight $2k-1$.

`mflinear(vecF,vecL)`: linear combination of the forms in `vecF` with coefficients in `vecL`. Forms must have the same weight and character, but not necessarily the same level. This function is used for simpler operations such as

```
mflinear([F],[s])        \\ scalar multiplication
mflinear([F,G],[1,1])    \\ addition
mflinear([F,G],[1,-1])   \\ subtraction
```

If `vecF=mfbasis(mf)`, it is better to write `mflinear(mf,vecL)` instead, since coefficient computations will be faster.

`mfmanin(FS)`: FS being a modular symbol associated to an eigenform, returns $[[P^+,P^-],[\omega^+,\omega^-,r]]$ where the $P^\pm$ are the even/odd polynomials of special values, the $\omega^\pm$ the corresponding periods, and $r = \Im(\omega^+\overline{\omega^-})/ < F,F >$.

`mfmul(F,G)`: product of the modular forms F and G.

`mfnumcusps(N)`: number of cusps of $\Gamma_0(N)$.

`mfparams(F)`: returns parameters $[N,k,\chi,P]$ of the modular form F, where K is the polynomial defining the number field containing the coefficients of F (e.g., y if F is rational), or $[-1,-1,-1,0]$ if it is not defined. If F is a modular form space, returns $[N,k,\chi,\mathit{space}]$.

`mfperiodpol(mf,F,{parity = 0})`: period polynomial of the form F; if the `parity` argument is 1 or -1, return the even/odd period polynomial.

`mfperiodpolbasis(k,{parity = 0})`: basis of period polynomials of weight k for the full modular group, even/odd ones if `parity` is 1 or -1.

`mfpetersson(FS,{GS = FS})`: FS and GS being the modular symbols associated to F and G with `mfsymbol`, computes the Petersson product of F and G with the usual normalization $1/[\Gamma:\Gamma_0(N)]$. (Petersson square if GS is omitted.)

`mfpow(F, n)`: Modular form F to the power n.

`mfsearch([N,k], V, {space = 4})`: search for *rational* modular forms of weight k and level N in the specified modular form spaces whose Fourier expansion up to the length of V exactly matches V. The output is a list

of forms. The parameter N may be replaced by a list of allowed levels, e.g. `[`N_1`..`N_2`]` for all levels between N_1 and N_2.

`mfshift(F,m)`: `F` divided by q^m, omitting the remainder if there is one, where m can be positive or negative. The result is usually not a modular form.

`mfshimura(mf,F,{`$D = 1$`})`: F being a modular form of half-integral weight $k \geq 3/2$ and D a discriminant, return `[mf2,FS,v]`, where `FS` is the corresponding Shimura lift of integral weight $2k-1$, `mf2` the corresponding modular form space and `v` the coefficients of `FS` on the basis of `mf2`. By extension, D can also be a positive squarefree integer.

`mfslashexpansion(mf,f,g,n,flrat,{&`P`})`: compute the Fourier expansion of $f|_k g$ to order n, where f is a form in `mf` and $g \in M_2^+(\mathbb{Q})$. If `flrat` is set, try to "rationalize" (error if unsuccessful). If the output is $[a(0), ..., a(n)]$ and the optional P contains parameters $[\alpha, w, A]$, then $f|_k g = F|_k A$ where $F(\tau) = q^\alpha \sum_{0 \leq j \leq n} a(j) q^{j/w}$, with $q = \exp(2\pi i \tau)$. A is always upper triangular and usually the identity, so that $F|_k A$ is immediate to compute.

`mfspace(mf,{`F`})`: type of modular space `mf` if F is omitted, or of a modular form F in `mf`: result is 0 for new, 1 for cuspidal, 2 for old, 3 for full, 4 for Eisenstein, and -1 if form is not in the space.

`mfsplit(mf,{dimlim = 0},{flag = 0})`: compute the eigenforms in `mf`, and limit the dimension of each Galois orbit to `dimlim` if set. `flag` is used to avoid some long computations (see doc). The space `mf` *must* contain the new space. Note that the result is only a two-component vector `vF,vK`, where `vF` is a vector of eigenforms and `vK` the corresponding number fields, but is *not* similar to the output of an `mfinit` command.

`mfsturm(mf)`: If `mf` is a space, true Sturm bound of `mf`, i.e., largest valuation at infinity of a nonzero form. If `mf` is $[N, k, \chi]$, only an upper bound.

`mfsymbol(mf,F)`: initialize data for working with integrals related to F such as `mfsymboleval`, `mfpetersson`, and `mfmanin`.

`mfsymboleval(FS,path,{`γ`})`: FS being the modular symbol assocated to some form F and `path` being $[s_1, s_2]$ where s_1 and s_2 are cusps or points in the upper half-plane, evaluate the symbol on the path, i.e., compute the polynomial $\int_{s_1}^{s_2} (X - \tau)^{k-2} F(\tau)\, d\tau$. If $\gamma \in GL_2^+(\mathbb{Q})$ is given, replace F by $F|_k \gamma$. If the integral diverges, the result will be either a rational function or a polynomial of degree $d > k - 2$.

`mftaylor(F,n,{fl = 0})`: for now, only for $F \in M_k(\Gamma)$ and at the point i. Compute the first n Taylor coefficients of F around i; if `fl` is set compute in fact p_n such that

$$f(\tau) = (2i/(\tau+i))^k \sum_{n \geq 0} p_n((\tau-i)/(\tau+i))^n \ .$$

`mfTheta({χ})`: unary theta series corresponding to the primitive Dirichlet character χ, thus in weight $1/2$ (resp., $3/2$) if χ is even (resp., odd).

`mftobasis(mf,F,{flag = 0})`: coefficients of form F on the basis in `mf`. If `flag` is set, do not return an error if F does not belong to `mf` or not enough coefficients.

`mftocoset(N,M,L)`: L being the list of cosets output by `L=mfcosets(N)` and M being in $\mathrm{SL}_2(\mathbb{Z})$, output a pair $[\gamma, i]$ such that $M = \gamma L[i]$, where $\gamma \in \Gamma_0(N)$.

`mftonew(mf,F)`: Decompose F is the cuspidal space `mf` as a sum of $B(d)G_M$ where $G_M \in S_k^{\mathrm{new}}(\Gamma_0(M), \chi)$ and $dM \mid N$, return the vector of $[M, d, G]$.

`mftraceform(NK,{space = 0})`: gives the trace form corresponding to $NK = [N, k, \chi]$ and `space` (only the new space and the cuspidal space).

`mftwist(F,D)`: twist of the form F by the quadratic character (D/n).