User's Manual for the Modular Forms Package of Pari/GP

Henri Cohen

January 12, 2016

1 Introduction

1.1 Trace Formulas

Although the most popular way of doing computations on classical modular forms is through the use of modular symbols or analogous objects, another method is to use the Eichler–Selberg trace formula. To use this formula one needs large tables of class numbers of imaginary quadratic orders, but these are readily computed thanks to recursions due to Hurwitz and Eichler.

We will always work in $M_k(\Gamma_0(N), \chi)$, although other congruence subgroups could be considered. To the author's knowledge, the available general trace formulas are the following:

1. A formula for $\mathrm{Tr}(T(n))$ acting on $S_k(\Gamma_0(N), \chi)$ for any n, coprime to the level N or not. This formula has been known for at least 40 years, and the most useful and ready to implement form of this formula is due to the author, briefly published in XXX (there is an evident misprint to correct), and restated in other papers such as XXX.

2. A formula for $\mathrm{Tr}(T(n)) \circ W_Q$ (W_Q Atkin–Lehner operators) acting on $S_k(\Gamma_0(N))$, due to Skoruppa and Zagier. This formula allows to compute the trace of $T(n)$ on the newspace $S_k^{\mathrm{new}}(\Gamma_0(N))$, but unfortunately has not been worked out for nontrivial χ.

3. Very recently, Booker, Bober, and Lee have found a relation between the trace of $T(n)$ on $S_k(\Gamma_0(N), \chi)$ and the traces on $S_k^{\mathrm{new}}(\Gamma_0(M), \chi)$ for divisors M of N. This formula can be inverted and leads to an explicit formula for $\mathrm{Tr}(T(n))$ acting on $S_k^{\mathrm{new}}(\Gamma_0(N), \chi)$.

4. A very surprising and quite nontrivial consequence of the preceding formula is that the series $\sum_{n\geq 1} \mathrm{Tr}(T(n))q^n$ where $T(n)$ acts on the space $S_k(\Gamma_0(N), \chi)$ (not the newspace) is a modular form, which allows to construct the whole space directly.

Thanks to these formulas, one proceeds as follows:

– As a first step, one constructs a *trace form* $\mathcal{T}(N, \chi)$ (either on S_k, on S_k^{new}, or on the Atkin–Lehner subspaces of S_k^{new}). The coefficient of q is the dimension of the space.

– By applying sufficiently many Hecke operators $T(n)$ with n coprime to N, it is easy to show that we will generate the space, so we can extract a basis.

– If we are in the newspace or in an Atkin–Lehner subspace, using the Hecke algebra we *split* the space into $\mathbb{Q}$-orbits, so that we obtain explicitly the canonical basis of normalized eigenforms.

1.2 Eisenstein Series

In addition to the cusp forms, we also have spaces of Eisenstein series. In the present implementation, these are treated separately, but in the future they will be merged with the cusp forms.

Contrary to most cusp forms, the coefficients of these Eisenstein series are completely explicit, but the exact form of the basis is not readily found in the literature.

Definition 1.1 *1. For any characters χ_1 and χ_2, we set*

$$\sigma_k(\chi_1, \chi_2; n) = \sum_{d\mid n,\ d>0} \chi_1(d)\chi_2(n/d)d^k\ .$$

2. If χ_1 and χ_2 are primitive characters and $k \geq 1$ we set

$$E_k(\chi_1, \chi_2) = \delta_{\chi_2}\frac{L(\chi_1, 1-k)}{2} + \sum_{n\geq 1} \sigma_{k-1}(\chi_1, \chi_2; n)q^n\ ,$$

where $\delta_{\chi_2} = 1$ if χ_2 is the trivial character, and 0 otherwise.

3. If χ is a character we denote by $f(\chi)$ its conductor and by χ_f the primitive character modulo $f(\chi)$ equivalent to χ.

The theorem is as follows:

Theorem 1.2 *If $\chi(-1) = (-1)^k$ (otherwise the space is zero), a basis for the space of Eisenstein series in $M_k(\Gamma_0(N), \chi)$ is given as follows:*

1. *If $k \geq 3$, the forms $E_k(\chi_1, (\chi\chi_1^{-1})_f)(d\tau)$, where χ_1 runs through all primitive characters and d positive integers satisfying the condition $f(\chi_1)f(\chi\chi_1^{-1})d \mid N$.*
2. *If $k = 2$, the same forms as in (1) except if χ is the trivial character, in which case one must first exclude the forms for which χ_1 is trivial, but second add the forms $E_2(d\tau) - pE_2(pd\tau)$ for all primes p and positive integers d such that $pd \mid N$, where as usual $E_2 = 1 - 24\sum_{n\geq 1}\sigma_1(n)q^n$.*
3. *If $k = 1$ (so that χ is odd), the same forms as in (1) except that one restricts to characters χ_1 which are even.*

1.3 Modular Forms of Weight 1

It is well-known that it is difficult to compute spaces of modular forms of weight 1: one cannot use modular symbols nor trace formulas. One can use theta series, but the most common methods rely on dividing two modular forms of higher weights k and $k-1$. The main problem is of course to decide when a modular form is divisible by another.

Several methods have been used to deal with this problem. The present implementation uses what appears to be the most efficient, due to George Schaeffer, called *Hecke stability*. It is based on the following theorem, which is a special case of the theorem proved by Schaeffer.

Theorem 1.3 *Denote by $M_1^*(\Gamma_0(N), \chi)$ the (infinite-dimensional) vector space of meromorphic modular functions of weight 1, level N, and character χ, and let V be a finite-dimensional subspace. If there exists a prime $\ell \nmid N$ such that V is stable under $T(\ell)$, then $V \subset M_1(\Gamma_0(N), \chi)$.*

Corollary 1.4 *Let $\mathcal{E}$ be a nonempty finite subset of $M_1(\Gamma_0(N), \overline{\chi}) \setminus \{0\}$, and for each $E \in \mathcal{E}$, denote by D_E the division-by-E map from $S_2(\Gamma_0(N))$ to $M_1^*(\Gamma_0(N), \chi)$ sending f to f/E. Then if $\ell \nmid N$ is a prime, the maximal $T(\ell)$-stable subset $V_{\mathcal{E},\ell}$ of $V_{\mathcal{E}} = \bigcap_{E\in\mathcal{E}} \operatorname{Im}(D_E)$ satisfies*

$$S_1(\Gamma_0(N), \chi) \subset V_{\mathcal{E},\ell} \subset M_1(\Gamma_0(N), \chi)\ .$$

To transform this into an algorithm, we first need to compute the maximal $T(\ell)$ stable subset, and second we need to compute the specific subspace $S_1(\Gamma_0(N), \chi)$.

To compute the maximal stable subset we proceed as follows. Let r be large, let $g = \dim(S_2(\Gamma_0(N)))$, and for each $E \in \mathcal{E}$ let A_E be the $r \times g$ matrix whose columns are the coefficients of the Fourier expansion of f_i/E for a fixed basis $(f_i)_{1 \le i \le g}$. Note that it is essential to include the constant terms (often 0) since some Eisenstein series have 0 constant term. The image of D_E is equal to the image of A_E, so $V_{\mathcal{E}}$ is the intersection of these images, which is the image of a matrix A easily computed using Algorithm XXX. On the other hand, let M be the matrix of $T(\ell)$ acting on the Fourier expansions at infinity (once again not forgetting the constant terms). The maximal $T(\ell)$-stable subset of $V_{\mathcal{E}}$ expressed on the columns of A is the set of column vectors X such that $MAX = AY$ for some column vector Y. Multiplying on the left by the left inverse of A ($A^{-1}A = I$) gives $A^{-1}MAX = Y$, hence $MAX = AA^{-1}MAX$, i.e., $V_{\mathcal{E}}$ is the kernel of the matrix $(I - AA^{-1})MA$.

2 Internal and External Representations

2.1 Representation of Dirichlet Characters

In the present implementation, we have decided to leave to the user many possible choices for representing Dirichlet characters when working inside the spaces S_k for $k \ge 2$. However, when working with noncusp forms or in M_1, it is preferable to use a slightly sophisticated representation due to B. Conrey. This will certainly change in the future.

Whatever representation is used, it is essential to be able to know: the values of the character (evidently), its *modulus* N, and its *conductor* f. It is also sometimes useful, but not essential, to know its *order* ord. The allowed representations are as follows:

1. A pair $[N, [\chi(1), \dots, \chi(f)]]$.
2. A triple $[N, f, C]$, where C is a closure.
3. A Conrey representation $[N, m, \mathrm{ord}, [\mathrm{tech}]]$, where tech is a vector of technical data.
4. A triple $[N, \mathrm{ord}, [\mathrm{chilog}(1), \dots, \mathrm{chilog}(f)]]$, where the $\mathrm{chilog}(n)$s are integers such that $\chi(n) = \zeta_{\mathrm{ord}}^{\mathrm{chilog}(n)}$ if n is coprime to N, and 0 otherwise.
5. An integer D, representing the Kronecker symbol $\left(\dfrac{D}{n}\right)$.

In the future, we will certainly include or replace by the Conrey representations implemented by P. Molin and/or K. Belabas.

2.2 Representation of Modular Forms

The representation of a classical modular form by its Fourier expansion at infinity has many disadvantages, but because of the use of the trace formula we are almost compelled to use it (this is not the case with modular symbols). Since evidently the number of necessary (or desired) Fourier coefficients is not known in advance, it is reasonable to use *closures*. We have two choices: either a closure $n \mapsto a(n)$ giving the nth Fourier coefficient, or a closure $n \mapsto [a(0), \ldots, a(n-1)]$ giving the first n. The latter would have a number of advantages, but it has a big disadvantage when computing the action of Hecke operators, and all attempts to use it have resulted in slowing down the programs by a factor of at least 5. Thus a modular form will be represented by a closure $n \mapsto a(n)$ for $n \geq 0$.

3 Available Functions on Characters

There are three kinds of functions on characters: general functions available for all types, functions to convert from one type to another, and specific functions on Conrey characters.

- General Functions.

`charmodulus(CHI)`: returns the modulus N.

`charconductor(CHI)`: returns the conductor f. (to test if a character is primitive, test if $f == N$).

`charorder(CHI)`: returns the order of χ.

`chargalois(N,parity=0,flagord=0)`: returns a list of representatives of the Galois orbits of elements coprime to N, to be used in conjunction with `conreycreate(N,.)`. If `parity` is 1 or -1, give only those with that parity, and if `flagord` is set only those whose order is less than or equal to flagord.

`charinv(CHI)`: inverse of the character χ.

`chartoprimitive(CHI)`: primitive character modulo f equivalent to χ.

`charisquadratic(CHI)`: true if character is quadratic (including the trivial) false if not.

`chareval(CHI,n,N=0)`: compute $\chi(n)$, considering χ as a character modulo N. **Warning**: N is not necessarily the modulus of χ: set it to 0 (default) if you want that.

`charparity`(CHI): returns $\chi(-1)$.

`charistrivial`(CHI): returns 1 if χ is the trivial character modulo N.

`chartrivial`(N): creates the trivial character modulo N in some format.

`charvec`(CHI): returns the vector $[\chi(1),\dots,\chi(N)]$, where N is the modulus of χ.

- Character conversions.

`chartype(CHI)`: returns the type (from 1 to 5 as above) of the character.

`charfromconrey(T)`: given a character in Conrey format, convert it to type T (from 1 to 5 as above).

`chartoconrey(CHI)`: convert to Conrey format.

`chartohecke(CHI)`: convert to Hecke (type 4) format.

`charhecketoconrey(CHI)`: convert from Hecke to Conrey format. Used internally, special case of `chartoconrey`.

- Functions specific to Conrey format.

`conreycreate(N,m)`: create the mth Dirichlet character modulo N in the Conrey numbering, together with all necessary technical data. Thus, if for instance you want to range over all characters modulo N, you use this function with all m with $0 \le m < N$ such that $\gcd(m,N)=1$.

`conreyclos(N,m)`, `conreytoclos(CHI)`: create the closure corresponding to the mth Conrey character modulo N, or to the already created Conrey character CHI.

`conreyeval(CHI,n)`: evaluate the Conrey character χ at n. Special case of `chareval(CHI,n)`.

`conreylog(CHI,n)`: Conrey logarithm of χ at n, i.e., a positive integer L such that $\chi(n)=\exp(2\pi iL/\text{ord})$, where ord is the order of the character, -1 if $\gcd(n,N)>1$.

`conreyvec(CHI)`: vector of $\chi(n)$ for $1\le n\le N$, with N modulus of χ.

`conreyveclog(CHI)`: vector of Conrey logarithms of n for $1\le n\le N$, with N modulus of χ.

`conreyinduce(CHI,M)`: Conrey character modulo M *multiple* of the modulus N equivalent to χ.

`conreyinducesimple(CHI,M)`: Conrey number of `conreyinduce(CHI,M)`.

`conreymul(CHI1,CHI2)`: product of the two Conrey characters.

`conreydiv(CHI1,CHI2)`: quotient of the two Conrey characters.

`conreyinv(CHI)`: inverse (conjugate) of the Conrey character χ.

`conreymulprimitive`, `conreydivprimitive`: same as `conreymul` and `conreydiv` except that they return the primitive character equivalent to the product or quotient.

`conreyconductor(CHI)`: special case of `charconductor`.

`conreytoprimitive(CHI)`: special case of `chartoprimitive`.

4 Functions on Modular Forms

4.1 Functions on Eisenstein Series of Weight $k \geq 1$

We have two types of Eisenstein series with character: the type explained above

$$E_k(\chi_1, \chi_2) = \delta_{\chi_2} \frac{L(\chi_1, 1-k)}{2} + \sum_{n \geq 1} \sigma_k(\chi_1, \chi_2; n) q^n ,$$

and the special case $E_k(\chi) := E_k(\chi, 1)$, where 1 is the trivial character modulo 1.

`mfeisen1clos(k,CHI,flinv=0)`: closure giving the coefficients of $E_k(\chi, 1)$, or of $E_k(\overline{\chi}, 1)$ if flinv is set.

`mfeisen1ser(k,CHI,lim=precdl)`: series expansion up to lim.

`mfeisen2clos(k,CHI1,CHI2`: closure giving the coefficients of $E_k(\chi_1, \chi_2)$.

`mfeisen2ser(k,CHI1,CHI2,lim=precdl)`: series expansion up to lim.

`mfeisenbasisclos(N,k,CHI)`: vector of closures of a basis of Eisenstein series for $M_k(\Gamma_0(N), \chi)$.

`mfeisenbasisvec(N,k,CHI,lim=precdl)`: vector expansions up to lim.

`mfeisenbasisser(N,k,CHI,lim=precdl)`: series expansions up to lim.

`mfeisenbasisdim(N,k,CHI)`: dimension of the space of Eisenstein series.

`mfeisencoeffs(mf,F)`: coefficients of closure or series F on the basis formed by the union of, first, the Eisenstein series given by `mfeisenbasisclos`, and second, the basis of cusp forms given in mf as a result of some `mfinit` call.

`mfE4clos()`, `mfE6clos()`, `mfEkclos(k)`: closure corresponding to the classical Eisenstein series E_4, E_6, E_k.

`mfdeltaclos()`: closure corresponding to the classical Ramanujan delta function Δ.

`mfelljclos()`: closure corresponding to the classical elliptic j-invariant (gives $q * j$ instead of j). Warning: this is given for completeness, but if you want the q-expansion of j, use series directly, not closures.

4.2 Utility Functions For Modular Form Computations

• Initializations, conversions, and closure conputations.

`mktab(L)`: constructs a table of class numbers up to L. Calls `mktabh(L)`, so very fast once the latter is constructed.

`mktabh(L)`: constructs a table of Hurwitz class numbers up to L.

`mycl(D)`: only for debugging, class number for $D < -4$.

`myhcl(D)`: only for debugging, 6 times Hurwitz class number $H(-D)$, $D < 0$.

`gcdinfty(a,b)`: $\gcd(a, b^\infty)$ (a and b small).

`primdiv(N)`: list as a vecsmall the primitive divisors Q of N, i.e., such that $\gcd(Q, N/Q) = 1$.

`mfclostovec(F,lim=precdl)`, `mfclostocol(F,lim=precdl)`: F being a closure, vector of $F(n)$ for $0 \leq n <$ lim.

`mfclostoser(F,lim=precdl)`: F being a closure, series of $F(n)$ up to precision lim, in the variable q.

`mfvectoconj(V,ind=1)`: transform a vector of polmods V into a vector of complex by choosing the ind-th root of the common polynomial modulus.

`mfclostoconj(F,lim=precdl,ind=1)`: same as mfvectoconj(mfclostovec(F,lim),ind).

`mfclosembed(F,ind=1)`: embed a closure F into a closure with complex coeffs by choosing the ind-th root of the polynomail defining $\mathbb{Q}(F)$.

`mfclosadd(F, G)`, `mfclossub(F, G)`, `mfclosmul(F, G)`, `mfclosinv(F)`, `mfclosdiv(F, G)`, `mfclosreverse(F)`: corresponding operations on closures. Warning: `mfclosdiv` and `mfclosreverse` are given for completeness, but are very slow. Usually, you will want to do the operations directly on series, not on closures.

`mfclosscalmul(F,la)`: closure $n \mapsto \lambda F(n)$.

`mfcloslinear(vF,vla)`: closure $n \mapsto \sum \lambda_i F_i(n)$.

`mfclosval(F,flag=0)`: valuation of the series corresponding to the closure F. If flag is set, return 100 if F seems to be the zero closure, otherwise error.

• Specific Modular Form Operators.

`mfheckevec(N,k,CHI=[1,[1]],V,n,lim=precdl)`: V being a vector, return the vector $T(n)(V)$ of length lim (not always possible if V too short).

`mfheckeser(N,k,CHI=[1,[1]],S,n)`: S being a series, return $T(n)(S)$ as a shorter series.

`mfheckeclos(N,k,CHI=[1,[1]],F,n)`: F being a closure, return the closure $T(n)(F)$.

`mfbdclos(F,d)`: F being a closure, return the closure $B(d)(F)$, where $B(d)$ is the diamond operator $\tau \mapsto d\tau$.

`mftwist(F,D)`: twist of the closure F by the quadratic character D.

4.3 Main Modular Form Computations

• Auxilliary functions.

`mfcusps(N)`: list of cusps of $\Gamma_0(N)$.

`mfsturm(N,k)`: Sturm bound for $S_k(\Gamma_0(N))$.

`mfdim(N,k,CHI=[1,[1]],flag=0)`: dimension of $S_k(\Gamma_0(N),\chi)$, or of $M_k(\Gamma_0(N),\chi)$ if flag is set.

`mftrace(N,k,CHI=[1,[1]],n,flag=0)`: trace of $T(n)$ on S_k, some verifications done if flag is set.

`mftracevec(N,k,CHI=[1,[1]],lim=precdl)`: vector of traces of $T(n)$ on S_k for $0 \le n \le$ lim.

`mftraceser(N,k,CHI=[1,[1]],lim=precdl)`: trace series on $S_k(\Gamma_0(N),\chi)$.

`mftraceclos(N,k,CHI=[1,[1]])`: trace form on $S_k(\Gamma_0(N),\chi)$ as a closure.

`mfnewdim(N,k,CHI=[1,[1]])`: dimension of $S_k^{\text{new}}(\Gamma_0(N),\chi)$.

`mfnewALdim(N,k,Q,flag=0)`: If $Q > 0$, dimension of Atkin–Lehner Q-space in $S_k^{\text{new}}(\Gamma_0(N),\chi)$. If $Q = 0$, sum of the dimensions (same but slower than `mfnewdim`), if $Q < 0$, all individual dimensions in the order of `primdiv`.

`mfnewtrace(N,k,CHI=[1,[1]],n)`, `mfnewtracevec(N,k,CHI=[1,[1]],Q=0,lim=precdl)`, `mfnewtraceser(N,k,CHI=[1,[1]],Q=0,lim=precdl)`, `mfnewtraceclos(N,k,CHI=[1,[1]],Q=0)`: same as the corresponding ones, but for S_k^{new}, or $S_k^{\text{new},Q}$ if Q is nonzero.

• Main Functions.

`mfinit(N,k,CHI=[1,[1]],code=MFNEW,split=SPRED)`: fundamental initialization function for working in modular form spaces. The meaning of the flags are as follows: if code=MFNEW (0), newspace, code=MFFULL (-1), fullspace computed directly, code=MFOLD (-2), oldspace computed directly, code=MFFULLN (-3) fullspace computed from newspace, code=MFATK (-4), obtain newspace from Atkin-Lehner subspaces, but implemented only for trivial CHI, code=MFOLDN (-5), oldspace computed from newspace. Also, code can be a strictly positive Q, interpreted as a primitive divisor of N (so we must have $Q|N$ and $\gcd(Q,N/Q) = 1$), in which case only the Q-Atkin–Lehner space is given. For the moment the latter is implemented only for trivial CHI.

The flag split is used inside the newspace (code=MFNEW, MFATK, or $Q > 0$ primitive divisor). split=NONSP: do not split the space, split=SPLIT: split but do not try to simplify the defining polynomials (may be expensive), split=SPRED: split and simplify using `polredbest`, split=SPRAT: split but keep only the rational eigenspaces. Warning: in this case, the dimension which is equal to mf[2] is still the full dimension, not the number of rational eigenspaces. This can induce wrong results on other functions.

If fullspace, oldspace, or newspace and nonsplit, the result is a 4-component vector mf, where mf[1]=[N,k,CHI,code], mf[2] is a vector of closures giving a basis for the space (so the dimension of the space is `length(mf[2])`), mf[3] contains the corresponding indices: either j for $T(j)(tf)$ if newspace,

$[M, j, d]$ for $B(d)T(j)(tf_M)$ if fullspace or oldspace, or $[Q, j]$ for Atkin newspace. mf[4] contains the matrix M of first coefficients of basis, of maximal rank and minimum number of rows.

If split, the result has two additional components: mf[5] is the vector of vectors of coefficients on the basis of closures mf[2] of the eigenspaces, as polmods in the variable x, so `length(mf[5])` is the number of eigenspaces, and mf[6] is the vector of the corresponding defining polynomials of the polmods, so `length(mf[6])` is also the number of eigenspaces, and the degrees of the polynomials are the dimensions.

`mfgaldims(mf)`: vector of dimensions of the Galois orbits if split, of the whole space if not.

`mfgalpols(mf)`: vector of polynomials defining the Galois orbits if split, empty vector if not.

`mfbasisvec(mf,lim=precdl,flag=0)`: vector of vectors of length lim giving the eigenform coefficients if split, the basis coefficients if not. If flag is set and not split, give the forms in echelon form.

`mfbasisser(mf,lim=precdl,flag=0)`: vector of series of length lim giving the eigenform coefficients if split, the basis coefficients if not. If flag is set and not split, give the forms in echelon form.

`mfbasisclos(mf)`: if split, vector of closures giving the eigenforms, otherwise vector of closures giving the basis mf[2].

`mftruesturm(mf)`: optimal Sturm bound for space mf, usually quite smaller than Sturm bound.

`mfmkmatcoeffs(mf)` **BAD NAME**: square matrix with complex entries of coefficients of the eigenforms (all Galois embeddings) on the basis mf[2].

`mfcoeffs(mf,F)`: give the coefficients of the closure F on the space mf, error of F does not belong to the space.

`mfnewsplit(mf,flag=1,coderat=0)`: used only for debugging. Split a newspace or an Atkin–Lehner subspace. If flag=1, reduce the polynomials using `polredbest`, and if coderat is set, keep only the rational eigenspaces. The result is a two-component vector [mf5,mf6], where mf5 contains the vector of vectors of coefficients on mf[2] of the eigenspaces, as polmods in the variable x, and mf6 the vector of polynomials defining the Galois orbits.

`mfvectoser(mf,v,lim=precdl)` **BAD NAME**: linear combination with coefficients v of the basis mf[2] as a series to precision lim.

`mfmathecke(mf,n)`: Matrix of $T(n)$ in the space mf.

`mfcloseval0(N,k,CHI=[1,[1]],F,vtau)`: compute the numerical value of the modular form F given as a closure for the complex number vtau, or the vector of complex numbers vtau in H. k must be the weight. N is optional, but if set to 1 assumes that F is of level 1 which considerably improves the algorithm. For now, the algorithm performs very badly if vtau (or some element of vtau if it is a vector) is close to the imaginary axis, except if N=1. Note that the execution time increases only moderately with the length of vtau, so whenever possible, use the vector form if several values must be computed for the same modular form (also exists a bitprec version).

`mfcloseval(mf,F,vtau)`: same as mfcloseval0 but N, k, CHI are given by mf (also exists a bitprec version).

`mfatkin(mf,Q)`: Atkin–Lehner eigenvalues of the eigenforms for the primitive divisor Q. Only for the split newspace, and only for quadratic characters defined modulo N/Q. The computation is done numerically if Q is not a prime, so can be slow. It is advised to use the lowest possible accuracy (or bitprec accuracy) (also exists a bitprec version).

`mfmatatkin(mf,Q)`: Matrix of the Atkin–Lehner operator W_Q on the basis mf[2]. Implemented only for the newspace and real characters defined modulo N/Q. Note that this function is useless when $Q > 0$ since we are already in the Atkin–Lehner space, but it will return a result (also exists a bitprec version).

Warning 1: the result is of the form $[A, M]$, where A is either 1 or Q, and M is a matrix with rational entries. If $A = 1$ (which will always be the case in even weight), the Atkin–Lehner matrix is M, but if $A = Q$, the Atkin–Lehner matrix is $M/\sqrt{(}Q)$.

Warning 2: In low accuracy the result may be false since we do a rational approximation.

`mfatkinclos(mf,Q,F,ATK)`: give as closure $F|W_Q$. `ATK`, if present, is the result of `mfmatatkin(mf,Q)` (also exists a bitprec version).

`mfnewdec(mf,F)`: mf being the FULL (not new) space, returns a vector of 3-component vectors $[M, d, G]$, where $f(\chi) \mid M \mid N$, $d \mid N/M$, and G is a closure in $S_k^{\mathrm{new}}(\Gamma_0(M), \chi)$, such that F is equal to the sum of the $B(d)(G)$ for all these 3-component vectors.

`mfnewlevel(mf,F)`: smallest level dividing N on which the modular form represented by the closure F is defined.

`mfnewdimq1(N,k,CHI=[1,[1]])`: number of rational eigenforms in S_k^{new}.

`mfnewALdimq1(N,k,Q)`: if $Q > 0$, number of rational eigenforms in the Atkin–Lehner newspace for Q, if $Q = 0$ total number (same as `mfnewdimq1`), if $Q < 0$ `vecsmall` of number of rational eigenforms in every Atkin–Lehner newspace, in the same order as that given by `primdiv`.

`mfcuspexpansion(mf,F,cusp)`: cusp being an element of $\mathbb{P}_1(\mathbb{Q})$ in the form ∞ or a/c with $c \mid N$. outputs the closure giving the Fourier expansion at that cusp. For now, only for $\gcd(c, N/c) = 1$, and the variable q is in fact $\exp(2\pi i\tau/c)$ (and not $\exp(2\pi i\tau)$), since the width of the cusp is

$(N/c)/\gcd(c,N/c)$. To see the expansion itself, use `mfclostoser` on the result.

4.4 Applications

`mfsearch(nklim,vmod,flag=0,lim=0,lpr=0)`: search for a modular form given a few initial coefficients, ONLY normalized newforms with integer coefficients and trivial character, although it would be trivial to add quadratic characters.

`vmod` is either a list of the prime index coefficients $[a_2, a_3, a_5, ...a_p]$; one can omit a_p in the list by setting it to a type not t_INT or t_FRAC, the shortest is probably to set it to I. Or `vmod` can be a list of pairs $[..., [p, a_p], ...]$.

`nklim` governs the limits of the search: if it is a t_INT, search for $N \cdot k \leq 2 \cdot \mathrm{nklim}$; if it is a vector of t_INT $[N_2, N_4, N_6, ...]$, search for $N \leq N_2$ for $k = 2$, $N \leq N_4$ for $k = 4$, etc...; if it is a vector of pairs $[[k_1, N_1], [k_2, N_2], ...]$ search with $N \leq N_1$ for $k = k_1$, $N \leq N_2$ for $k = k_2$, etc...

The meaning of `flag` is as follows: if `flag = 0`, search for exact match; if `flag = 1`: search for match with a_p modulo p, if `flag = m` with $m \geq 2$, search for match modulo fixed m for all p.

`lim` is the length of the series given in the result, if `lpr` is set the results are printed as they are found, since the search can be very long.

The result is a vector of 3-component vectors $[N, k, se]$, where N is the level, k the weight, and se the series to `lim` terms.

`mfsearchAL(nklim,vmod,flag=0,lim=0,lpr=0)`: same as `mfsearch`, but loop through the Atkin–Lehner spaces instead of all the newspaces, usually faster. The results are now 4-component vectors $[N, k, Q, se]$, where Q is the Atkin–Lehner divisor.

`mflfuncreate(mf)`, `lfunmodform(mf)`: vector of `lfuncreate` s of all the eigenforms, of length the dimension of the space (all the newspace or Atkin–Lehner space), so each Galois orbit of dimension d has d corresponding `lfuncreate` s, one for each embedding.

`mffromell(E)`: E being an elliptic curve defined over $\mathbb{Q}$ given by an integral model by `ellinit`, returns a 3-component vector `[mf,F,coeffs]`, where F is the closure associated to the newform corresponding to E by modularity, `mf` is the split newspace to which F belongs, and `coeffs` are the coefficients of F on the basis mf[2] of closures. Note that `coeffs` must be one of the coefficients given by mf[5], and this is checked.

4.5 Modular Forms of Weight 1

The functions defining the spaces of Eisenstein series also work in weight 1, so we only need cusp forms.

`mfwt1basis(N,CHI,lim=0)`: Basis of the space $S_1(\Gamma_0(N),\chi)$ as series using George Schaeffer's Hecke stability. `lim` is the order of q-expansions with which the program will compute, but the output is to a much higher series precision.

`mfwt1basisall(N,lim=0)`: Union of bases of spaces $S_1(\Gamma_0(N),\chi)$ for a given N. The result is a vector indexed by the Galois conjugacy classes of χ (see `chargalois`), and each component is [CHI,basis], where CHI is the Dirichlet character in Conrey numbering, and basis is the basis given by `mfwt1basis`. The degree of the polmods defining the basis is equal to the cardinality of the Galois conjugacy class of χ and equal to Euler's ϕ function applied to the order of χ.

`mfwt1dim(N)`: dimension of $S_1(\Gamma_1(N))$.

`mfwt1newdim(N)`: dimension of $S_1^{\mathrm{new}}(\Gamma_1(N))$.