

[Tutorial] Modular Forms

Henri Cohen

January 16, 2018



Theory I: Modularity

A **modular form** is a function F from the upper half-plane $\mathbb{H} = \{\tau \in \mathbb{C}, \Im(\tau) > 0\}$ to \mathbb{C} satisfying:

- A **modularity condition**: for all $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbb{Z})$ (a, b, c, d in \mathbb{Z} and $ad - bc = 1$) such that $N \mid c$, and for all $\tau \in \mathbb{H}$ we have

$$F\left(\frac{a\tau + b}{c\tau + d}\right) = \chi(d)(c\tau + d)^k F(\tau),$$

for some integer k and Dirichlet character χ modulo N .

- A **holomorphy condition**: F is holomorphic on \mathbb{H} , and also at the **cusps** (no need to define this for now).

The set of all matrices as above is denoted $\Gamma_0(N)$, and the vector space of all modular forms of the above type is denoted $M_k(\Gamma_0(N), \chi)$. It is trivially 0 if $\chi(-1) \neq (-1)^k$. Fundamental theorem: it is **finite dimensional**, known dimension.

Theory I: Modularity

A **modular form** is a function F from the upper half-plane $\mathbb{H} = \{\tau \in \mathbb{C}, \Im(\tau) > 0\}$ to \mathbb{C} satisfying:

- A **modularity condition**: for all $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbb{Z})$ (a, b, c, d in \mathbb{Z} and $ad - bc = 1$) such that $N \mid c$, and for all $\tau \in \mathbb{H}$ we have

$$F\left(\frac{a\tau + b}{c\tau + d}\right) = \chi(d)(c\tau + d)^k F(\tau),$$

for some integer k and Dirichlet character χ modulo N .

- A **holomorphy condition**: F is holomorphic on \mathbb{H} , and also at the **cusps** (no need to define this for now).

The set of all matrices as above is denoted $\Gamma_0(N)$, and the vector space of all modular forms of the above type is denoted $M_k(\Gamma_0(N), \chi)$. It is trivially 0 if $\chi(-1) \neq (-1)^k$. Fundamental theorem: it is **finite dimensional**, known dimension.

Theory II: Spaces of Modular Forms

Important **subspaces** of the full space $M_k(\Gamma_0(N), \chi)$ are:

- The space of **cuspidal forms** $S_k(\Gamma_0(N), \chi)$, forms which **vanish** at all cusps.
- The space of **Eisenstein series** $\mathcal{E}_k(\Gamma_0(N), \chi)$: these are easily **explicitly constructed** modular forms, and we have

$$M_k(\Gamma_0(N), \chi) = \mathcal{E}_k(\Gamma_0(N), \chi) \oplus S_k(\Gamma_0(N), \chi).$$

- The space of **oldforms** $S_k^{\text{old}}(\Gamma_0(N), \chi)$, modular forms constructed from forms of a lower **level** M with $f(\chi) \mid M \mid N$, $M < N$ ($f(\chi)$ conductor of χ).
- The space of **newforms** $S_k^{\text{new}}(\Gamma_0(N), \chi)$, defined as a canonical supplement to the space of oldforms.

Theory II: Spaces of Modular Forms

Important **subspaces** of the full space $M_k(\Gamma_0(N), \chi)$ are:

- The space of **cuspidal forms** $S_k(\Gamma_0(N), \chi)$, forms which **vanish** at all cusps.
- The space of **Eisenstein series** $\mathcal{E}_k(\Gamma_0(N), \chi)$: these are easily **explicitly constructed** modular forms, and we have

$$M_k(\Gamma_0(N), \chi) = \mathcal{E}_k(\Gamma_0(N), \chi) \oplus S_k(\Gamma_0(N), \chi).$$

- The space of **oldforms** $S_k^{\text{old}}(\Gamma_0(N), \chi)$, modular forms constructed from forms of a lower **level** M with $f(\chi) \mid M \mid N$, $M < N$ ($f(\chi)$ conductor of χ).
- The space of **newforms** $S_k^{\text{new}}(\Gamma_0(N), \chi)$, defined as a canonical supplement to the space of oldforms.

Theory II: Spaces of Modular Forms

Important **subspaces** of the full space $M_k(\Gamma_0(N), \chi)$ are:

- The space of **cuspidal forms** $S_k(\Gamma_0(N), \chi)$, forms which **vanish** at all cusps.
- The space of **Eisenstein series** $\mathcal{E}_k(\Gamma_0(N), \chi)$: these are easily **explicitly constructed** modular forms, and we have

$$M_k(\Gamma_0(N), \chi) = \mathcal{E}_k(\Gamma_0(N), \chi) \oplus S_k(\Gamma_0(N), \chi).$$

- The space of **oldforms** $S_k^{\text{old}}(\Gamma_0(N), \chi)$, modular forms constructed from forms of a lower **level** M with $f(\chi) \mid M \mid N$, $M < N$ ($f(\chi)$ conductor of χ).
- The space of **newforms** $S_k^{\text{new}}(\Gamma_0(N), \chi)$, defined as a canonical supplement to the space of oldforms.

Theory II: Spaces of Modular Forms

Important **subspaces** of the full space $M_k(\Gamma_0(N), \chi)$ are:

- The space of **cuspidal forms** $S_k(\Gamma_0(N), \chi)$, forms which **vanish** at all cusps.
- The space of **Eisenstein series** $\mathcal{E}_k(\Gamma_0(N), \chi)$: these are easily **explicitly constructed** modular forms, and we have

$$M_k(\Gamma_0(N), \chi) = \mathcal{E}_k(\Gamma_0(N), \chi) \oplus S_k(\Gamma_0(N), \chi).$$

- The space of **oldforms** $S_k^{\text{old}}(\Gamma_0(N), \chi)$, modular forms constructed from forms of a lower **level** M with $f(\chi) \mid M \mid N$, $M < N$ ($f(\chi)$ conductor of χ).
- The space of **newforms** $S_k^{\text{new}}(\Gamma_0(N), \chi)$, defined as a canonical supplement to the space of oldforms.

Theory III: Modular Forms

Because of modularity, a modular form F is periodic of period 1, and because of holomorphy (meromorphy would suffice) it has a **Fourier expansion**

$$F(\tau) = \sum_{n \in \mathbb{Z}} a(n) e^{2\pi i n \tau},$$

the coefficients $a(n)$ being the **Fourier coefficients** of F .

Because of holomorphy at the cusp $i\infty$, we have $a(n) = 0$ for $n < 0$, so if we set $q = e^{2\pi i \tau}$, we have $F(\tau) = \sum_{n \geq 0} a(n) q^n$. This is **not** sufficient in general to ensure holomorphy at all cusps, but can show that this is equivalent to $a(n) = O(n^M)$ for some fixed M .

Theory III: Modular Forms

Because of modularity, a modular form F is periodic of period 1, and because of holomorphy (meromorphy would suffice) it has a **Fourier expansion**

$$F(\tau) = \sum_{n \in \mathbb{Z}} a(n) e^{2\pi i n \tau} ,$$

the coefficients $a(n)$ being the **Fourier coefficients** of F .

Because of holomorphy at the cusp $i\infty$, we have $a(n) = 0$ for $n < 0$, so if we set $q = e^{2\pi i \tau}$, we have $F(\tau) = \sum_{n \geq 0} a(n) q^n$.

This is **not** sufficient in general to ensure holomorphy at all cusps, but can show that this is equivalent to $a(n) = O(n^M)$ for some fixed M .

Theory IV: Hecke Operators

The last important ingredient of the theory are **Hecke operators**. Ad hoc (but correct) definition: if $F(\tau) = \sum_{m \geq 0} a(m)q^m$, the operator $T(n)$ is defined by $T(n)(F)(\tau) = \sum_{m \geq 0} b(m)q^m$ with

$$b(m) = \sum_{d|\gcd(n,m)} \chi(d)d^{k-1} a(mn/d^2).$$

It can be shown that all the above spaces of modular forms are stable under $T(n)$, and that they form a **commuting algebra** of operators, and that if $\gcd(n, N) = 1$, or if we are in the new space $S_k^{\text{new}}(\Gamma_0(N), \chi)$ they are even **Hermitian**.

In particular they are simultaneously diagonalizable, an eigenvector for all $T(n)$ is called an **eigenform**, one proves that $a(1) \neq 0$ so eigenforms are **normalized** by $a(1) = 1$.

Theory IV: Hecke Operators

The last important ingredient of the theory are **Hecke operators**. Ad hoc (but correct) definition: if $F(\tau) = \sum_{m \geq 0} a(m)q^m$, the operator $T(n)$ is defined by $T(n)(F)(\tau) = \sum_{m \geq 0} b(m)q^m$ with

$$b(m) = \sum_{d|\gcd(n,m)} \chi(d)d^{k-1} a(mn/d^2).$$

It can be shown that all the above spaces of modular forms are stable under $T(n)$, and that they form a **commuting algebra** of operators, and that if $\gcd(n, N) = 1$, or if we are in the new space $S_k^{\text{new}}(\Gamma_0(N), \chi)$ they are even **Hermitian**.

In particular they are simultaneously diagonalizable, an eigenvector for all $T(n)$ is called an **eigenform**, one proves that $a(1) \neq 0$ so eigenforms are **normalized** by $a(1) = 1$.

Theory IV: Hecke Operators

The last important ingredient of the theory are **Hecke operators**.

Ad hoc (but correct) definition: if $F(\tau) = \sum_{m \geq 0} a(m)q^m$, the operator $T(n)$ is defined by $T(n)(F)(\tau) = \sum_{m \geq 0} b(m)q^m$ with

$$b(m) = \sum_{d|\gcd(n,m)} \chi(d)d^{k-1} a(mn/d^2).$$

It can be shown that all the above spaces of modular forms are stable under $T(n)$, and that they form a **commuting algebra** of operators, and that if $\gcd(n, N) = 1$, or if we are in the new space $S_k^{\text{new}}(\Gamma_0(N), \chi)$ they are even **Hermitian**.

In particular they are simultaneously diagonalizable, an eigenvector for all $T(n)$ is called an **eigenform**, one proves that $a(1) \neq 0$ so eigenforms are **normalized** by $a(1) = 1$.

Theory V: Construction of Modular Forms

There are not so many ways to construct modular forms explicitly.

- **Eisenstein Series**.
- **Theta functions** associated to a lattice and a spherical polynomial.
- Modular forms attached to **Hecke characters** on imaginary and real quadratic fields.
- Modular forms associated to **elliptic curves** by Wiles's modularity theorem
- **Trace forms** using the Eichler–Selberg trace formula.
- Many types of **operators** enabling to construct new modular forms from existing ones.

Implementation

Three types of objects:

- Modular form **spaces**, initialized by the command `mfinit` with a flag corresponding to one of the five spaces mentioned above.
- Modular forms themselves: if F is such a form, `mfcoefs(F,n)` gives the vector of coefficients $[a(0), a(1), \dots, a(n)]$, and `mfparams(F)` gives $[N, k, \chi, pol]$, level, weight, character, and polynomial in y defining the field $\mathbb{Q}(F)/\mathbb{Q}(\chi)$.
- Dirichlet characters: represented either by a discriminant D for the Kronecker–Legendre symbol (D/n) ($D = 1$ trivial character), by an `intmod Mod(a,N)` with $\gcd(a, N) = 1$ (Conrey numbering), or by a general Pari/GP group $[G, \chi]$.

Three types of objects:

- Modular form **spaces**, initialized by the command `mfinit` with a flag corresponding to one of the five spaces mentioned above.
- Modular forms themselves: if F is such a form, `mfcoefs(F,n)` gives the vector of coefficients $[a(0), a(1), \dots, a(n)]$, and `mfparams(F)` gives $[N, k, \chi, pol]$, level, weight, character, and polynomial in y defining the field $\mathbb{Q}(F)/\mathbb{Q}(\chi)$.
- Dirichlet characters: represented either by a discriminant D for the Kronecker–Legendre symbol (D/n) ($D = 1$ trivial character), by an `intmod Mod(a,N)` with $\gcd(a, N) = 1$ (Conrey numbering), or by a general Pari/GP group $[G, \chi]$.

Three types of objects:

- Modular form **spaces**, initialized by the command `mfinit` with a flag corresponding to one of the five spaces mentioned above.
- Modular forms themselves: if F is such a form, `mfcoefs(F,n)` gives the vector of coefficients $[a(0), a(1), \dots, a(n)]$, and `mfparams(F)` gives $[N, k, \chi, pol]$, level, weight, character, and polynomial in y defining the field $\mathbb{Q}(F)/\mathbb{Q}(\chi)$.
- Dirichlet characters: represented either by a discriminant D for the Kronecker–Legendre symbol (D/n) ($D = 1$ trivial character), by an `intmod Mod(a,N)` with $\gcd(a, N) = 1$ (Conrey numbering), or by a general Pari/GP group $[G, \chi]$.

Practice: Modular Form Leaves I

```
D = mfDelta(); V = mfcoefs(D, 8)
Ser(V,q)
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048, -16744, 84480]
% = 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)
```

Practice: Modular Form Leaves I

```
D = mfDelta(); V = mfcoefs(D, 8)
Ser(V,q)
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048, -16744, 84480]
% = 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)
```

Practice: Modular Form Leaves II

```
E4 = mfEk(4); E6 = mfEk(6);  
apply(x->mfcoefs(x,4), [E4,E6])  
E43 = mfpow(E4, 3); E62 = mfpow(E6, 2);  
DP = mflinear([E43, E62], [1, -1]/1728);  
mfcoefs(DP, 6)  
mfisequal(D, DP)
```

```
% = [[1, 240, 2160, 6720, 17520],  
      [1, -504, -16632, -122976, -532728]]  
% = [0, 1, -24, 252, -1472, 4830, -6048]  
% = 1
```

Practice: Modular Form Leaves II

```
E4 = mfEk(4); E6 = mfEk(6);  
apply(x->mfcoefs(x,4), [E4,E6])  
E43 = mfpow(E4, 3); E62 = mfpow(E6, 2);  
DP = mflinear([E43, E62], [1, -1]/1728);  
mfcoefs(DP, 6)  
mfisequal(D, DP)  
  
% = [[1, 240, 2160, 6720, 17520],  
      [1, -504, -16632, -122976, -532728]]  
% = [0, 1, -24, 252, -1472, 4830, -6048]  
% = 1
```

Practice: Modular Form Leaves III

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

Here `mfetaquo` represents an **eta quotient**, here $\eta(\tau)^2\eta(11\tau)^2$. The corresponding modular form is equal to the modular form associated to the elliptic curve “11a1” of conductor 11.

```
% = [0, 1, -2, -1, 2, 1, 2, -2, 0, -2, -2]
% = 1
```

Practice: Modular Form Leaves III

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

Here `mffrometaquo` represents an **eta quotient**, here $\eta(\tau)^2\eta(11\tau)^2$. The corresponding modular form is equal to the modular form associated to the elliptic curve “11a1” of conductor 11.

```
% = [0, 1, -2, -1, 2, 1, 2, -2, 0, -2, -2]
% = 1
```

Practice: Modular Form Spaces I

```
mf = mfinit([1,12]); L = mfbasis(mf); #L
mfdim(mf)
mfcoefs(L[1],6)
mfcoefs(L[2],6)
```

The default is to ask for the **full** space $M_k(\Gamma_0(N), \chi)$ (flag = 4).

```
% = 2
% = 2
% = [691/65520, 1, 2049, 177148, 4196353, 48828126]
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

Note: for now, the Eisenstein series are given before the cusp forms, and they are normalized with $a(1) = 1$, **not** $a(0) = 1$ (which is impossible in general).

Practice: Modular Form Spaces I

```
mf = mfinit([1,12]); L = mfbasis(mf); #L
mfdim(mf)
mfcoefs(L[1],6)
mfcoefs(L[2],6)
```

The default is to ask for the **full** space $M_k(\Gamma_0(N), \chi)$ (flag = 4).

```
% = 2
% = 2
% = [691/65520, 1, 2049, 177148, 4196353, 48828126]
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

Note: for now, the Eisenstein series are given before the cusp forms, and they are normalized with $a(1) = 1$, **not** $a(0) = 1$ (which is impossible in general).

Practice: Modular Form Spaces II

Note the direct command

```
mfcoefs(mf,6)
```

which outputs

```
% =  
[691/65520      0]  
[          1      1]  
[       2049    -24]  
[   177148    252]  
[  4196353  -1472]  
[  48828126   4830]  
[362976252  -6048]
```

This command is in general **much** faster than asking for each individual expansion in the basis.

Practice: Modular Form Spaces III

The **cuspidal** space is with `flag = 1`:

```
mf = mfinit([1,12], 1); L = mfbasis(mf); #L
mfcoefs(L[1],6)
```

```
% = 1
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

Practice: Modular Form Spaces III

The **cuspidal** space is with `flag = 1`:

```
mf = mfinit([1,12], 1); L = mfbasis(mf); #L
mfcoefs(L[1],6)
```

```
% = 1
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048]
```

Practice: Modular Form Spaces IV

The **newspace** is with `flag = 0`:

```
mf = mfinit([35,2], 0); L = mfbasis(mf); #L
for (i = 1, 3, print(mfcoefs(L[i], 10)))
```

(or more simply `mfcoefs(mf,10)` which gives a matrix)

```
% = 3
[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 modular cusp forms. Usually, one wants **eigenforms**: this is obtained by the command `mfeigenbasis`, which applies only to the newspace, even if the input is larger:

Practice: Modular Form Spaces IV

The **newspace** is with `flag = 0`:

```
mf = mfinit([35,2], 0); L = mfbasis(mf); #L
for (i = 1, 3, print(mfcoefs(L[i], 10)))
```

(or more simply `mfcoefs(mf,10)` which gives a matrix)

```
% = 3
[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 modular cusp forms. Usually, one wants **eigenforms**: this is obtained by the command `mfeigenbasis`, which applies only to the newspace, even if the input is larger:

Practice: Modular Form Spaces V

```
mffields(mf)
L = mfeigenbasis(mf); #L
mfcoefs(L[1],10)
mfcoefs(L[2],3)
lift(mfcoefs(L[2],9))
```

```
% = [y, y^2 - y - 4]
% = 2
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
% = [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)]
% = [0, 1, -y, y - 1, y + 2, 1, -4, -1, -y - 4, -y + 2]
```

Practice: Modular Form Spaces V

```
mffields(mf)
L = mfeigenbasis(mf); #L
mfcoefs(L[1],10)
mfcoefs(L[2],3)
lift(mfcoefs(L[2],9))

% = [y, y^2 - y - 4]
% = 2
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
% = [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)]
% = [0, 1, -y, y - 1, y + 2, 1, -4, -1, -y - 4, -y + 2]
```

Practice: Modular Form Spaces VI

Very often, need **numerical values** of coefficients: need to **embed** in \mathbb{C} , so a given eigenform can give **several** forms. Numerical functions applied to modular forms (for example **mfeval**, which evaluates numerically a form) automatically give a vector of results when there are several embeddings.

To compute the numerical expansion of a form having several embeddings, we use **mfembed** as follows:

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


Practice: Modular Form Spaces VI

Very often, need **numerical values** of coefficients: need to **embed** in \mathbb{C} , so a given eigenform can give **several** forms. Numerical functions applied to modular forms (for example **mfeval**, which evaluates numerically a form) automatically give a vector of results when there are several embeddings.

To compute the numerical expansion of a form having several embeddings, we use **mfembed** as follows:

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

Practice: Modular Form Spaces VII

We apply to our above example:

```
[V1,V2]=mfcoefsemd(L[2],5);
```

```
V1
```

```
V2
```

```
% = [0, 1, 1.5615528128088302749107049279870385126,  
      -2.5615528128088302749107049279870385126,  
      0.43844718719116972508929507201296148743, 1]
```

```
% = [0, 1, -2.5615528128088302749107049279870385126,  
      1.5615528128088302749107049279870385126,  
      4.5615528128088302749107049279870385126, 1]
```

(imaginary parts of $0.E - 38$ omitted).

Practice: Modular Form Spaces VII

We apply to our above example:

```
[V1,V2]=mfcoefsebed(L[2],5);
```

```
V1
```

```
V2
```

```
% = [0, 1, 1.5615528128088302749107049279870385126,  
      -2.5615528128088302749107049279870385126,  
      0.43844718719116972508929507201296148743, 1]
```

```
% = [0, 1, -2.5615528128088302749107049279870385126,  
      1.5615528128088302749107049279870385126,  
      4.5615528128088302749107049279870385126, 1]
```

(imaginary parts of $0.E - 38$ omitted).

Practice: Modular Form Spaces VIII

```
[mf,F,co] = mffromell(ellinit("35a1")); mfcoefs(F, 10)
mfisequal(F, L[1])
```

```
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
% = 1
```

Practice: Modular Form Spaces VIII

```
[mf,F,co] = mffromell(ellinit("35a1")); mfcoefs(F, 10)
mfisequal(F, L[1])
```

```
% = [0, 1, 0, 1, -2, -1, 0, 1, 0, -2, 0]
% = 1
```

Practice: Modular Form Spaces IX

```
apply(x->mfdim([96, 2], x), [0..4])
```

```
% = [2, 9, 7, 15, 24]
```

```
mf = mfinit([96,2]); L = mfbasis(mf);  
for (i = 12, 15, print(mfcoefs(L[i], 14)))
```

```
[23/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[31/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[47/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[95/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

Practice: Modular Form Spaces IX

```
apply(x->mfdim([96, 2], x), [0..4])
```

```
% = [2, 9, 7, 15, 24]
```

```
mf = mfinit([96,2]); L = mfbasis(mf);  
for (i = 12, 15, print(mfcoefs(L[i], 14)))
```

```
[23/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[31/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[47/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[95/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

Practice: Modular Form Spaces IX

```
apply(x->mfdim([96, 2], x), [0..4])
```

```
% = [2, 9, 7, 15, 24]
```

```
mf = mfinit([96,2]); L = mfbasis(mf);  
for (i = 12, 15, print(mfcoefs(L[i], 14)))
```

```
[23/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[31/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[47/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[95/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```


Practice: Modular Form Spaces IX

```
apply(x->mfdim([96, 2], x), [0..4])
```

```
% = [2, 9, 7, 15, 24]
```

```
mf = mfinit([96,2]); L = mfbasis(mf);  
for (i = 12, 15, print(mfcoefs(L[i], 14)))
```

```
[23/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[31/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[47/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

```
[95/24, 1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28]
```

Practice: Modular Form Spaces X

```
F = mflinear([L[14],L[12]], [1,-1]); mfcoefs(F, 50)
G = mfhecke(mf, F, 24); mfcoefs(G, 11)
mftobasis(mf, G)
24*mfcoefs(L[5], 11)
```

```
% = [1, 0, 0, 0, 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, 0, 0, 0, 0, 24, 0, 0]
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288]
% = [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]~
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288]
```

Practice: Modular Form Spaces X

```
F = mflinear([L[14],L[12]], [1,-1]); mfcoefs(F, 50)
G = mfhecke(mf, F, 24); mfcoefs(G, 11)
mftobasis(mf, G)
24*mfcoefs(L[5], 11)
```

```
% = [1, 0, 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]
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288]
% = [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]~
% = [1, 24, 24, 96, 24, 144, 96, 192, 24, 312, 144, 288]
```

Practice: Modular Form Spaces XI

```
mf=mfinit([96,2],0); mffields(mf)
L = mfeigenbasis(mf); for(i=1,2,print(mfcoefs(L[i], 15)))
Fa = mffromell(ellinit("96a1"))[2]; mfcoefs(Fa, 15)
Fb = mffromell(ellinit("96b1"))[2]; mfcoefs(Fb, 15)
mfisequal(mftwist(Fa, -4), Fb)
```

```
% = [y, y]
      [0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2]
      [0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2]
% = [0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2]
% = [0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2]
% = 1
```

Practice: Modular Form Spaces XI

```
mf=mfinit([96,2],0); mffields(mf)
L = mfeigenbasis(mf); for(i=1,2,print(mfcoefs(L[i], 15)))
Fa = mffromell(ellinit("96a1"))[2]; mfcoefs(Fa, 15)
Fb = mffromell(ellinit("96b1"))[2]; mfcoefs(Fb, 15)
mfisequal(mftwist(Fa, -4), Fb)
```

```
% = [y, y]
      [0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2]
      [0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2]
% = [0, 1, 0, 1, 0, 2, 0, -4, 0, 1, 0, 4, 0, -2, 0, 2]
% = [0, 1, 0, -1, 0, 2, 0, 4, 0, 1, 0, -4, 0, -2, 0, -2]
% = 1
```

Practice: Spaces with Characters I

```
mf = mfinit([35,2,5],0); mffields(mf)
F = mfeigenbasis(mf)[1]; lift(mfcoefs(F, 10))
```

```
% = [y^2 + 1]
% = [0, 1, 2*y, -y, -2, -y - 2, 2, -y, 0, 2, -4*y + 2]
```

Because `mffields` gives $y^2 + 1$, in the last output y is equal to one of the two roots of $y^2 + 1 = 0$.

Practice: Spaces with Characters I

```
mf = mfinit([35,2,5],0); mffields(mf)
F = mfeigenbasis(mf)[1]; lift(mfcoefs(F, 10))
```

```
% = [y^2 + 1]
% = [0, 1, 2*y, -y, -2, -y - 2, 2, -y, 0, 2, -4*y + 2]
```

Because `mffields` gives $y^2 + 1$, in the last output y is equal to one of the two roots of $y^2 + 1 = 0$.

Practice: Spaces with Characters II

```
mf = mfinit([23,1,-23], 0); mfdim(mf)
F = mfbasis(mf)[1]; mfcoefs(F, 15)
mfgaloistype(mf,F)
```

`mfgaloistype` asks for the image in $\mathrm{PSL}_2(\mathbb{C})$ of the projective representation associated to the form, here D_3 , the dihedral group with 6 elements (also equal to the symmetric group on 3 letters).

```
% = 1
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, -1, 0, 0]
% = 6
```


Practice: Spaces with Characters II

```
mf = mfinit([23,1,-23], 0); mfdim(mf)
F = mfbasis(mf)[1]; mfcoefs(F, 15)
mfgaloistype(mf,F)
```

`mfgaloistype` asks for the image in $\mathrm{PSL}_2(\mathbb{C})$ of the projective representation associated to the form, here D_3 , the dihedral group with 6 elements (also equal to the symmetric group on 3 letters).

```
% = 1
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, -1, 0, 0]
% = 6
```

Practice: Spaces with Characters III

Since the form is of type D_3 , hence **dihedral**, it can be obtained by theta functions. The command **mffromqf** does what is required:

```
F1 = mffromqf([2,1;1,12])[2]; V1 = mfcoefs(F1, 15)
F2 = mffromqf([4,1;1,6])[2]; V2 = mfcoefs(F2, 15)
(V1 - V2)/2
mfisequal(F, mflinear([F1, F2], [1, -1]/2))
```

```
% = [1, 2, 0, 0, 2, 0, 4, 0, 4, 2, 0, 0, 4, 0, 0, 0]
% = [1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 4, 2, 0, 0]
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, -1, 0]
% = 1
```

Practice: Spaces with Characters III

Since the form is of type D_3 , hence **dihedral**, it can be obtained by theta functions. The command **mffromqf** does what is required:

```
F1 = mffromqf([2,1;1,12])[2]; V1 = mfcoefs(F1, 15)
F2 = mffromqf([4,1;1,6])[2]; V2 = mfcoefs(F2, 15)
(V1 - V2)/2
mfisequal(F, mflinear([F1, F2], [1, -1]/2))
```

```
% = [1, 2, 0, 0, 2, 0, 4, 0, 4, 2, 0, 0, 4, 0, 0, 0]
% = [1, 0, 2, 2, 2, 0, 2, 0, 2, 2, 0, 0, 4, 2, 0, 0]
% = [0, 1, -1, -1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, -1, 0]
% = 1
```

Practice: Modular Forms of Weight One I

Note that when we wrote `mf=mfinit([23,1,-23], 0)`, we **knew** that the character to choose is $(-23/n)$. What if we don't? This is especially important for **weight 1** modular forms.

```
G = znstar(23, 1);  
L = [[G,chi]|chi<-chargalois(G),zncharisodd(G,chi)]; #L  
apply(x->mfdim([23,1,x], 1), L)  
apply(x->charorder(x[1],x[2]), L)
```

The above shows the most general way to define a Dirichlet character: first define the **group G** using `znstar(N,1)` (flag 1 necessary), then specify `chi` on generators, e.g., using `chargalois` or otherwise.

```
% = 2  
% = [0, 1]  
% = [22, 2]
```

Practice: Modular Forms of Weight One I

Note that when we wrote `mf=mfinit([23,1,-23], 0)`, we **knew** that the character to choose is $(-23/n)$. What if we don't? This is especially important for **weight 1** modular forms.

```
G = znstar(23, 1);  
L = [[G,chi]|chi<-chargalois(G),zncharisodd(G,chi)]; #L  
apply(x->mfdim([23,1,x], 1), L)  
apply(x->charorder(x[1],x[2]), L)
```

The above shows the most general way to define a Dirichlet character: first define the **group G** using `znstar(N,1)` (flag 1 necessary), then specify `chi` on generators, e.g., using `chargalois` or otherwise.

```
% = 2  
% = [0, 1]  
% = [22, 2]
```

Practice: Modular Forms of Weight One II

```
mfa = mfinit([23,1,0], 1); #mfa
mf = mfa[1]; mfdim(mf)
mfparams(mf)
```

This illustrates **wildcards**: the **0** (which is of course not limited to weight **1**) means that the result is a vector of `mf` of all spaces with given level and weight, but varying character (here, `mfparams` says that the only one is $(-23/n)$).

```
% = 1
% = 1
% = [23, 1, -23, 1]
```

Practice: Modular Forms of Weight One II

```
mfa = mfinit([23,1,0], 1); #mfa
mf = mfa[1]; mfdim(mf)
mfparams(mf)
```

This illustrates **wildcards**: the **0** (which is of course not limited to weight **1**) means that the result is a vector of `mf` of all spaces with given level and weight, but varying character (here, `mfparams` says that the only one is $(-23/n)$).

```
% = 1
% = 1
% = [23, 1, -23, 1]
```

Practice: Modular Forms of Weight One III

```
wt1exp(lim1,lim2)=
{
  my(mfall,mf,chi);
  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 ]
    )
  );
}
```


Practice: Modular Forms of Weight One IV

Copy the preceding program from the GP file available with the tutorial on the website: it explores “exotic” weight 1 forms between given levels, i.e., those whose projective image is not dihedral, so cannot easily be constructed explicitly (image A_4 code -12 , S_4 code -24 , A_5 code -60 , opposite of their cardinality).

For instance, try `w1exp(1,230)`, or `w1exp(633,633)`. The latter outputs

```
[633, Mod(71, 633), 2, 10, 60]
```

Practice: Modular Forms of Weight One IV

Copy the preceding program from the GP file available with the tutorial on the website: it explores “exotic” weight 1 forms between given levels, i.e., those whose projective image is not dihedral, so cannot easily be constructed explicitly (image A_4 code -12 , S_4 code -24 , A_5 code -60 , opposite of their cardinality).

For instance, try `w1exp(1,230)`, or `w1exp(633,633)`. The latter outputs

```
[633, Mod(71, 633), 2, 10, 60]
```

Practice: Miscellaneous Commands I

```
mf=mfinit([96,6],0); mffields(mf)
mfatkineigenvalues(mf,3)
mf=mfinit([96,3,-3],0); mffields(mf)
mfatkineigenvalues(mf,32)
mfatkineigenvalues(mf,3)
```

```
% = [y, y, y, y, y, y, y^2 - 31, y^2 - 31]
% = [[-1], [-1], [-1], [1], [1], [1], [-1, -1], [1, 1]]
% = [y^4 + 8*y^2 + 9, y^4 + 4*y^2 + 1]
% = [[I, -I, -I, I], [-I, I, I, -I]]
% = [[0.47.... ]] /* complicated complex numbers */
```

The reason we obtain complicated complex numbers in the last command is that the character $(-3/.)$ is not defined modulo $N/Q = 96/3 = 32$. These numbers, called *pseudo-eigenvalues*, are algebraic and of modulus 1.

Practice: Miscellaneous Commands I

```
mf=mfinit([96,6],0); mffields(mf)
mfatkineigenvalues(mf,3)
mf=mfinit([96,3,-3],0); mffields(mf)
mfatkineigenvalues(mf,32)
mfatkineigenvalues(mf,3)
```

```
% = [y, y, y, y, y, y, y^2 - 31, y^2 - 31]
% = [[-1], [-1], [-1], [1], [1], [1], [-1, -1], [1, 1]]
% = [y^4 + 8*y^2 + 9, y^4 + 4*y^2 + 1]
% = [[I, -I, -I, I], [-I, I, I, -I]]
% = [[0.47.... ]] /* complicated complex numbers */
```

The reason we obtain complicated complex numbers in the last command is that the character $(-3/.)$ is not defined modulo $N/Q = 96/3 = 32$. These numbers, called *pseudo-eigenvalues*, are algebraic and of modulus 1.

Practice: Miscellaneous Commands I

```
mf=mfinit([96,6],0); mffields(mf)
mfatkineigenvalues(mf,3)
mf=mfinit([96,3,-3],0); mffields(mf)
mfatkineigenvalues(mf,32)
mfatkineigenvalues(mf,3)
```

```
% = [y, y, y, y, y, y, y^2 - 31, y^2 - 31]
% = [[-1], [-1], [-1], [1], [1], [1], [-1, -1], [1, 1]]
% = [y^4 + 8*y^2 + 9, y^4 + 4*y^2 + 1]
% = [[I, -I, -I, I], [-I, I, I, -I]]
% = [[0.47.... ]] /* complicated complex numbers */
```

The reason we obtain complicated complex numbers in the last command is that the character $(-3/.)$ is not defined modulo $N/Q = 96/3 = 32$. These numbers, called *pseudo-eigenvalues*, are algebraic and of modulus 1.

Practice: Miscellaneous Commands II

```
mf = mfinit([96,2]); L = mfbasis(mf);  
mfdim([96,2],3)  
apply(x->mfconductor(mf,x), L)
```

```
% = 15
```

```
% = [16, 32, 48, 96, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96,  
     24, 48, 96, 32, 96, 48, 96, 96, 96]
```

Since the dimension of the Eisenstein space (code 3) is 15, this gives the conductors (lowest possible level) of the 15 Eisenstein series, then those of the 9 cusp forms in the given basis of `mf`.

Practice: Miscellaneous Commands II

```
mf = mfinit([96,2]); L = mfbasis(mf);  
mfdim([96,2],3)  
apply(x->mfconductor(mf,x), L)
```

```
% = 15
```

```
% = [16, 32, 48, 96, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96,  
     24, 48, 96, 32, 96, 48, 96, 96, 96]
```

Since the dimension of the Eisenstein space (code 3) is 15, this gives the conductors (lowest possible level) of the 15 Eisenstein series, then those of the 9 cusp forms in the given basis of `mf`.

Practice: Miscellaneous Commands III

```
C = mfcusps(108)
apply(x->mfcuspwidth(108,x), C)
NK = [108,3,-4];
apply(x->mfcuspisregular(NK,x), C)
[c | c<-C, !mfcuspisregular(NK,c)]

% = [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]
% = [108, 27, 12, 12, 27, 3, 3, 4, 4, 3,
      3, 1, 1, 4, 1, 1, 1, 1]
% = [1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1]
% = [1/2, 1/6, 5/6, 1/18, 5/18, 1/54]
```


Practice: Miscellaneous Commands III

```
C = mfcusps(108)
apply(x->mfcuspwidth(108,x), C)
NK = [108,3,-4];
apply(x->mfcuspisregular(NK,x), C)
[c | c<-C, !mfcuspisregular(NK,c)]

% = [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]
% = [108, 27, 12, 12, 27, 3, 3, 4, 4, 3,
      3, 1, 1, 4, 1, 1, 1, 1]
% = [1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1]
% = [1/2, 1/6, 5/6, 1/18, 5/18, 1/54]
```

Practice: Miscellaneous Commands IV

```
E4 = mfEk(4); G = mfderivE2(E4); mfcoefs(G, 6)
mfcoefs(mfEk(6), 6)/(-3)
F = mfderivE2(E4, 3); (-9)*mfcoefs(F, 5)
mfisequal(mfEk(10), mflinear([F],[-9]))
```

```
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [1, -264, -135432, -5196576, -69341448, -515625264]
% = 1
```

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

```
% = 1.4557628922687093224624220035988692874
% = 1.4557628922687093224624220035988692874
```

Practice: Miscellaneous Commands IV

```
E4 = mfEk(4); G = mfderivE2(E4); mfcoefs(G, 6)
mfcoefs(mfEk(6), 6)/(-3)
F = mfderivE2(E4, 3); (-9)*mfcoefs(F, 5)
mfisequal(mfEk(10), mflinear([F],[-9]))
```

```
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [1, -264, -135432, -5196576, -69341448, -515625264]
% = 1
```

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

```
% = 1.4557628922687093224624220035988692874
% = 1.4557628922687093224624220035988692874
```

Practice: Miscellaneous Commands IV

```
E4 = mfEk(4); G = mfderivE2(E4); mfcoefs(G, 6)
mfcoefs(mfEk(6), 6)/(-3)
F = mfderivE2(E4, 3); (-9)*mfcoefs(F, 5)
mfisequal(mfEk(10), mflinear([F],[-9]))
```

```
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [1, -264, -135432, -5196576, -69341448, -515625264]
% = 1
```

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

```
% = 1.4557628922687093224624220035988692874
% = 1.4557628922687093224624220035988692874
```

Practice: Miscellaneous Commands IV

```
E4 = mfEk(4); G = mfderivE2(E4); mfcoefs(G, 6)
mfcoefs(mfEk(6), 6)/(-3)
F = mfderivE2(E4, 3); (-9)*mfcoefs(F, 5)
mfisequal(mfEk(10), mflinear([F],[-9]))
```

```
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [-1/3, 168, 5544, 40992, 177576, 525168, 1352736]
% = [1, -264, -135432, -5196576, -69341448, -515625264]
% = 1
```

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

```
% = 1.4557628922687093224624220035988692874
% = 1.4557628922687093224624220035988692874
```

Practice: Miscellaneous Commands V

```
mf = mfinit([96,4], 0); 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]
```

Practice: Miscellaneous Commands V

```
mf = mfinit([96,4], 0); 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]
```

Practice: Miscellaneous Commands VI

```
P = charpoly(M)
print(factor(P))
```

```
% = x^6 - 1456*x^4 + 209664*x^2 - 2985984
[x - 36, 1; x - 12, 1; x - 4, 1; x + 4, 1;
      x + 12, 1; x + 36, 1]
```

Note that this shows that all the eigenvalues of $T(7)$ are integral, so the splitting will be entirely rational and the eigenforms with integral coefficients. Let's check:

Practice: Miscellaneous Commands VI

```
P = charpoly(M)
print(factor(P))
```

```
% = x^6 - 1456*x^4 + 209664*x^2 - 2985984
[x - 36, 1; x - 12, 1; x - 4, 1; x + 4, 1;
      x + 12, 1; x + 36, 1]
```

Note that this shows that all the eigenvalues of $T(7)$ are integral, so the splitting will be entirely rational and the eigenforms with integral coefficients. Let's check:

Practice: Miscellaneous Commands VII

```
mffields(mf)
L = mfeigenbasis(mf); for(i=1,6,print(mfcoefs(L[i],15)))

% = [y, y, y, y, y, y]
[0, 1, 0, 3, 0, 10, 0, 4, 0, 9, 0, -20, 0, 70, 0, 30]
[0, 1, 0, 3, 0, 2, 0, 12, 0, 9, 0, 60, 0, -42, 0, 6]
[0, 1, 0, 3, 0, -14, 0, -36, 0, 9, 0, -36, 0, 54, 0, -42]
[0, 1, 0, -3, 0, 10, 0, -4, 0, 9, 0, 20, 0, 70, 0, -30]
[0, 1, 0, -3, 0, 2, 0, -12, 0, 9, 0, -60, 0, -42, 0, -6]
[0, 1, 0, -3, 0, -14, 0, 36, 0, 9, 0, 36, 0, 54, 0, 42]
```

Note again the **twisting** phenomenon: there are three eigenforms, and three twists by the character $(-4/n)$.

Practice: Miscellaneous Commands VII

```
mffields(mf)
L = mfeigenbasis(mf); for(i=1,6,print(mfcoefs(L[i],15)))

% = [y, y, y, y, y, y]
[0, 1, 0, 3, 0, 10, 0, 4, 0, 9, 0, -20, 0, 70, 0, 30]
[0, 1, 0, 3, 0, 2, 0, 12, 0, 9, 0, 60, 0, -42, 0, 6]
[0, 1, 0, 3, 0, -14, 0, -36, 0, 9, 0, -36, 0, 54, 0, -42]
[0, 1, 0, -3, 0, 10, 0, -4, 0, 9, 0, 20, 0, 70, 0, -30]
[0, 1, 0, -3, 0, 2, 0, -12, 0, 9, 0, -60, 0, -42, 0, -6]
[0, 1, 0, -3, 0, -14, 0, 36, 0, 9, 0, 36, 0, 54, 0, 42]
```

Note again the **twisting** phenomenon: there are three eigenforms, and three twists by the character $(-4/n)$.

Practice: Miscellaneous Commands VIII

```
[mfB,M,C]=mfatkininit(mf,3); M
```

```
% =
```

```
[ 0 -3 0 0 -24 0]
```

```
[-1/3 0 -4/3 0 0 -12]
```

```
[ 0 0 0 -9/5 -6/5 0]
```

```
[ 0 0 -2/3 0 0 -1]
```

```
[ 0 0 1/6 0 0 3/2]
```

```
[ 0 0 0 1/5 4/5 0]
```

Practice: Miscellaneous Commands VIII

```
[mfB,M,C]=mfatkininit(mf,3); M
```

```
% =
```

```
[ 0 -3 0 0 -24 0]
```

```
[-1/3 0 -4/3 0 0 -12]
```

```
[ 0 0 0 -9/5 -6/5 0]
```

```
[ 0 0 -2/3 0 0 -1]
```

```
[ 0 0 1/6 0 0 3/2]
```

```
[ 0 0 0 1/5 4/5 0]
```

Practice: Miscellaneous Commands IX

The matrix of the Atkin–Lehner involution W_Q is the above matrix divided by C , but here $C = 1$: `[C,matdet(M/C)]` outputs `[1, -1]`. Since the eigenvalues are real in even weight and no character, this means that there is an **odd** number of -1 , hence an odd number of $+1$:

```
[C,matdet(M/C)]  
mfatkineigenvalues(mf,3)
```

```
% = [1, -1]  
% = [[-1], [-1], [-1], [1], [1], [1]]
```

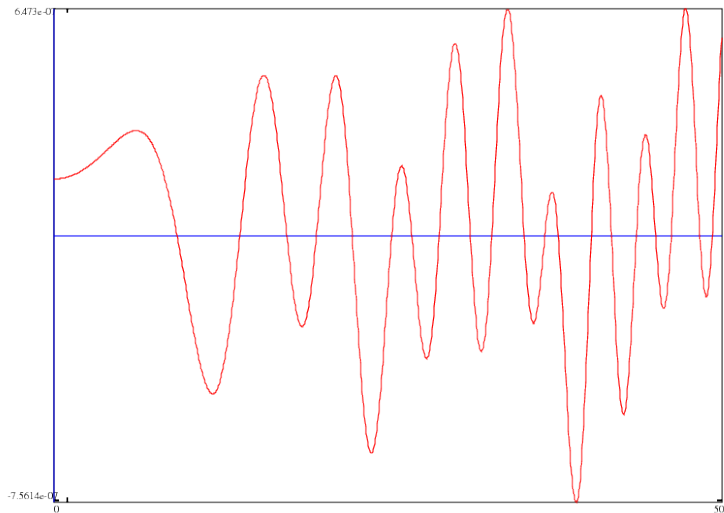
Practice: Miscellaneous Commands IX

The matrix of the Atkin–Lehner involution W_Q is the above matrix divided by C , but here $C = 1$: `[C,matdet(M/C)]` outputs `[1, -1]`. Since the eigenvalues are real in even weight and no character, this means that there is an **odd** number of -1 , hence an odd number of $+1$:

```
[C,matdet(M/C)]  
mfatkineigenvalues(mf,3)
```

```
% = [1, -1]  
% = [[-1], [-1], [-1], [1], [1], [1]]
```


Practice: Combination with L-Functions II



Practice: Combination with L-Functions III

```
PP = mfperiodpol(mf,D,-1); PP /= polcoeff(PP,1);  
bestappr(PP)  
PM = mfperiodpol(mf,D,1); PM /= polcoeff(PM,0);  
bestappr(PM)  
mfperiodpolbasis(12)
```

```
% = x^9 - 25/4*x^7 + 21/2*x^5 - 25/4*x^3 + x  
% = -x^10 + 691/36*x^8 - 691/12*x^6  
      + 691/12*x^4 - 691/36*x^2 + 1  
% = [x^8 - 3*x^6 + 3*x^4 - x^2,  
      4*x^9 - 25*x^7 + 42*x^5 - 25*x^3 + 4*x, x^10 - 1]
```

Practice: Combination with L-Functions III

```
PP = mfperiodpol(mf,D,-1); PP /= polcoeff(PP,1);  
bestappr(PP)  
PM = mfperiodpol(mf,D,1); PM /= polcoeff(PM,0);  
bestappr(PM)  
mfperiodpolbasis(12)
```

```
% = x^9 - 25/4*x^7 + 21/2*x^5 - 25/4*x^3 + x
```

```
% = -x^10 + 691/36*x^8 - 691/12*x^6
```

```
      + 691/12*x^4 - 691/36*x^2 + 1
```

```
% = [x^8 - 3*x^6 + 3*x^4 - x^2,
```

```
      4*x^9 - 25*x^7 + 42*x^5 - 25*x^3 + 4*x, x^10 - 1]
```

Practice: Miscellaneous Commands IX

```
E4 = mfEk(4); F = mfbracket(E4, E4, 2); mfcoefs(F, 6)/4800
D = mfDelta(); mftaylor(D, 9)*1728
D3 = mftwist(D, -3); mfcoefs(D3, 9)
P = mfparams(D3)
mf = mfinit(D3, 1); mftobasis(mf, D3)
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048]
% = [1, 0, -1/12, 0, 1/96, 0, 1/288, 0, -11/2304, 0]
% = [0, 1, 24, 0, -1472, -4830, 0, -16744, -84480, 0]
% = [9, 12, 1, y]
% = [0, 0, 0, 0, 0,
      5546/4131, -1232/12393, -47/16524, 11/24786]~
```

Practice: Miscellaneous Commands IX

```
E4 = mfEk(4); F = mfbracket(E4, E4, 2); mfcoefs(F, 6)/4800
D = mfDelta(); mftaylor(D, 9)*1728
D3 = mftwist(D, -3); mfcoefs(D3, 9)
P = mfparams(D3)
mf = mfinit(D3, 1); mftobasis(mf, D3)
```

```
% = [0, 1, -24, 252, -1472, 4830, -6048]
% = [1, 0, -1/12, 0, 1/96, 0, 1/288, 0, -11/2304, 0]
% = [0, 1, 24, 0, -1472, -4830, 0, -16744, -84480, 0]
% = [9, 12, 1, y]
% = [0, 0, 0, 0, 0,
      5546/4131, -1232/12393, -47/16524, 11/24786]~
```

Practice: Miscellaneous Commands X

```
F = mffromell(ellinit("49a1"))[2]; mfisCM(F)
mfisequal(F, mftwist(F, -7))
mf = mfinit([23,1,-23],1); F = mfeigenbasis(mf)[1];
mfisCM(F)
mfisequal(F, mftwist(F, -23))
```

```
% = -7
% = 1
% = -23
% = 0
```


Practice: Miscellaneous Commands X

```
F = mffromell(ellinit("49a1"))[2]; mfisCM(F)
mfisequal(F, mftwist(F, -7))
mf = mfinit([23,1,-23],1); F = mfeigenbasis(mf)[1];
mfisCM(F)
mfisequal(F, mftwist(F, -23))
```

```
% = -7
% = 1
% = -23
% = 0
```

Practice: Searching: mfeigensearch I

We want to **search** for normalized **eigenforms** with **integral** (equivalently, rational) Fourier coefficients, given a few $a(p)$ for p prime, possibly modulo something.

```
L = mfeigensearch([[1..30],4], [[2,2],[3,-1]]); #L
F = L[1]; mfparams(F)
mfcoefs(F, 10)
```

```
% = 1
% = [26, 4, 1, y]
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
```

Practice: Searching: mfeigensearch I

We want to **search** for normalized **eigenforms** with **integral** (equivalently, rational) Fourier coefficients, given a few $a(p)$ for p prime, possibly modulo something.

```
L = mfeigensearch([[1..30],4], [[2,2],[3,-1]]); #L
F = L[1]; mfparams(F)
mfcoefs(F, 10)
```

```
% = 1
```

```
% = [26, 4, 1, y]
```

```
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
```

Practice: Searching: mfeigensearch II

```
L=mfeigensearch([[1..30],4], [[2,Mod(2,5)], [3,Mod(-1,5)]])  
[ mfparams(F)[1] | F <- L ]  
F1 = L[1]; mfcoefs(F1, 10)  
F2 = L[2]; mfcoefs(F2, 10)  
F = mflinear([F1, F2], [-1, 1]); mfcoefs(F, 14)/5  
mfsturm([26,4])
```

```
% = [26, 26]
```

```
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
```

```
% = [0, 1, 2, 4, 4, -18, 8, 20, 8, -11, -36]
```

```
% = [0, 0, 0, 1, 0, -7, 2, 11, 0, 3, -14, -10, 4, 0, 22]
```

```
% = 15
```

Practice: Searching: mfeigensearch II

```
L=mfeigensearch([[1..30],4], [[2,Mod(2,5)], [3,Mod(-1,5)]])  
[ mfparams(F)[1] | F <- L ]  
F1 = L[1]; mfcoefs(F1, 10)  
F2 = L[2]; mfcoefs(F2, 10)  
F = mflinear([F1, F2], [-1, 1]); mfcoefs(F, 14)/5  
mfsturm([26,4])
```

```
% = [26, 26]
```

```
% = [0, 1, 2, -1, 4, 17, -2, -35, 8, -26, 34]
```

```
% = [0, 1, 2, 4, 4, -18, 8, 20, 8, -11, -36]
```

```
% = [0, 0, 0, 1, 0, -7, 2, 11, 0, 3, -14, -10, 4, 0, 22]
```

```
% = 15
```

Practice: Searching: mfsearch

A more primitive searching is the **mfsearch** command:

```
W = mfsearch([[1..35],3],[0,1,2,3,4,5,6,7,8],1);  
[mfparams(F) | F <- W]  
mfcoefs(W[1],10)  
mfcoefs(W[2],10)
```

```
% = [[30, 3, -3, y], [30, 3, -15, y]]  
% = [0, 1, 2, 3, 4, 5, 6, 7, 8, -14, -30]  
% = [0, 1, 2, 3, 4, 5, 6, 7, 8, -21, -50]
```

We are searching for modular forms with rational coefficients, of weight **3** and level less than or equal to **35**, in the cuspidal space (code **1**) whose Fourier expansion begins with $q + 2q^2 + 3q^3 + 4q^4 + 5q^5 + 6q^6 + 7q^7 + 8q^8 + \dots$. We find that there are two, both of level **30**, one with character $(-3/.)$, the second $(-15/.)$, and we give **11** coefficients.



Practice: Searching: mfsearch

A more primitive searching is the **mfsearch** command:

```
W = mfsearch([[1..35],3],[0,1,2,3,4,5,6,7,8],1);  
[ mfparams(F) | F <- W ]  
mfcoefs(W[1],10)  
mfcoefs(W[2],10)
```

```
% = [[30, 3, -3, y], [30, 3, -15, y]]  
% = [0, 1, 2, 3, 4, 5, 6, 7, 8, -14, -30]  
% = [0, 1, 2, 3, 4, 5, 6, 7, 8, -21, -50]
```

We are searching for modular forms with rational coefficients, of weight **3** and level less than or equal to **35**, in the cuspidal space (code **1**) whose Fourier expansion begins with $q + 2q^2 + 3q^3 + 4q^4 + 5q^5 + 6q^6 + 7q^7 + 8q^8 + \dots$. We find that there are two, both of level **30**, one with character $(-3/.)$, the second $(-15/.)$, and we give **11** coefficients.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to 500 are reachable in reasonable weight.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to 500 are reachable in reasonable weight.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to 500 are reachable in reasonable weight.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to 500 are reachable in reasonable weight.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to 500 are reachable in reasonable weight.

Advanced Commands

The Pari/GP modular form package is unique in that it implements a number of advanced functions on modular forms not available in other packages:

- 1 Fourier expansion of $F|_k\gamma$, and in particular expansion at any cusp.
- 2 Numerical computation of Atkin–Lehner pseudo-eigenvalues.
- 3 Numerical evaluation of a form near the real axis.
- 4 Numerical computation of **symbols**, i.e., integrals over any path.
- 5 Numerical computation of general **Petersson products**.

This is based on the computation of bases of modular form spaces made of products of Eisenstein series, and of general expansions of these series. Although more expensive than previous computations, once the precomputations are done the rest is essentially immediate. In practice levels up to **500** are reachable in reasonable weight.

Fourier expansion of $F|_{k\gamma}$

```
mf = mfininit([32,4],0); F = mfbasis(mf)[1]; mfcoefs(F,10)
mfslashexpansion(mf,F,[0,-1;32,0],10,1,&A);
A
```

Here we ask for the action of the **Fricke involution**
 $\tau \mapsto -1/(32\tau)$ on F ; the parameter **1** asks the program to
“rationalize” the result, and **A** will be explained below.

```
% = [0, 3, 0, 0, 0, 2, 0, 0, 0, 47, 0]
% = [0, 1, 0, 16, 0, 22, 0, 32, 0, -27, 0]
% = [0, 1]
```

$A = [0, 1]$ means that the expansion will be of the form
 $q^0 \sum_{n \geq 0} a(n)q^{n/1}$, here simply $\sum_{n \geq 0} a(n)q^n$. Thus

$$F|_4 W_{32} = q + 16q^3 + 22q^5 + 32q^7 - 27q^9 + O(q^{11}).$$

Fourier expansion of $F|_{k\gamma}$

```
mf = mfininit([32,4],0); F = mfbasis(mf)[1]; mfcoefs(F,10)
mfslashexpansion(mf,F,[0,-1;32,0],10,1,&A);
A
```

Here we ask for the action of the **Fricke involution**
 $\tau \mapsto -1/(32\tau)$ on F ; the parameter **1** asks the program to
“rationalize” the result, and **A** will be explained below.

```
% = [0, 3, 0, 0, 0, 2, 0, 0, 0, 47, 0]
% = [0, 1, 0, 16, 0, 22, 0, 32, 0, -27, 0]
% = [0, 1]
```

$A = [0, 1]$ means that the expansion will be of the form
 $q^0 \sum_{n \geq 0} a(n)q^{n/1}$, here simply $\sum_{n \geq 0} a(n)q^n$. Thus

$$F|_4 W_{32} = q + 16q^3 + 22q^5 + 32q^7 - 27q^9 + O(q^{11}).$$

Fourier expansion of $F|_{k\gamma}$ II

```
mf = mfinit([12,8],0); F = mfbasis(mf)[1];
mfslashexpansion(mf,F,[1,0;2,1],7,0,&A)
A
mfslashexpansion(mf,F,[1,0;2,1],7,1,&A)

% = [0, 0, 0, 0.6666666... + 0.E-38*I, 0,
      -3.99999999... + 6.9282032302...*I, 0,
      -11.99999999... - 20.7846096908...*I]

% = [0, 3]
% = [0, 0, 0, 2/3, 0, Mod(8*t, t^2 + t + 1),
      0, Mod(-24*t - 24, t^2 + t + 1)]
```

Here $A = [0, 3]$ so the expansion is in powers of $q^{1/3}$ (still with q^0 in front); the first command (parameter 0) gives the coefficients as complex numbers (whose real part is easy to recognize), and the last (parameter 1) “rationalizes” the result, showing that these coefficients seem to be (are in fact) in $\mathcal{O}(\exp(2\pi i/3))$.

Fourier expansion of $F|_{k\gamma}$ II

```
mf = mfinit([12,8],0); F = mfbasis(mf)[1];
mfslashexpansion(mf,F,[1,0;2,1],7,0,&A)
A
mfslashexpansion(mf,F,[1,0;2,1],7,1,&A)
% = [0, 0, 0, 0.6666666... + 0.E-38*I, 0,
      -3.99999999... + 6.9282032302...*I, 0,
      -11.99999999... - 20.7846096908...*I]
% = [0, 3]
% = [0, 0, 0, 2/3, 0, Mod(8*t, t^2 + t + 1),
      0, Mod(-24*t - 24, t^2 + t + 1)]
```

Here $A = [0, 3]$ so the expansion is in powers of $q^{1/3}$ (still with q^0 in front); the first command (parameter 0) gives the coefficients as complex numbers (whose real part is easy to recognize), and the last (parameter 1) “rationalizes” the result, showing that these coefficients seem to be (are in fact) in $\mathbb{Q}(\exp(2\pi i/3))$.

Fourier expansion of $F|_{k\gamma}$ III

```
mf = mfinit([12,7,-4],0); F = mfbasis(mf)[1];  
mfslashexpansion(mf,F,[1,0;6,1],5,1,&A)  
A
```

```
% = [-5/32, 81/32, 21/16, -597/8, 1215/32, 1689/8]  
% = [1/2, 1]
```

Here we have an example with $A[1] = 1/2 \neq 0$: we have

$$F|_7 \begin{pmatrix} 1 & 0 \\ 6 & 1 \end{pmatrix} = q^{1/2}(-5/32 + (81/32)q + (21/16)q^2 - (597/8)q^3 + \dots).$$

Fourier expansion of $F|_{k\gamma}$ III

```
mf = mfinit([12,7,-4],0); F = mfbasis(mf)[1];  
mfslashexpansion(mf,F,[1,0;6,1],5,1,&A)  
A
```

```
% = [-5/32, 81/32, 21/16, -597/8, 1215/32, 1689/8]  
% = [1/2, 1]
```

Here we have an example with $A[1] = 1/2 \neq 0$: we have

$$F|_7 \begin{pmatrix} 1 & 0 \\ 6 & 1 \end{pmatrix} = q^{1/2}(-5/32 + (81/32)q + (21/16)q^2 - (597/8)q^3 + \dots).$$

Evaluation of a Form I

`mfeval` can easily evaluate a form near the real axis:

```
mf = mfinit([12,4],1); F = mfbasis(mf)[1];
mfeval(mf,F,1/Pi+10^(-6)*I)
mfeval(mf,F,1/Pi+10^(-7)*I)
mfeval(mf,F,1/Pi+10^(-8)*I)

% = -89811.049350396250531782882568405506024
    - 58409.940965200894541585402642924371696*I
% = 4.8212468504661113183253396691813292261 E-52
    + 6.7885262281520647908871247541561415340 E-52*I
% = 0
```

These results are immediate and **correct**: at height 10^{-6} the value is large, at height 10^{-7} very small (and really of the order of 10^{-52} with 30 correct decimals). Of course the value is not exactly 0 at height 10^{-8} but cannot be computed with 38 decimals default accuracy (simply increase the accuracy to **57D**, the value is of the order of 10^{-69}).



Evaluation of a Form I

`mfeval` can easily evaluate a form near the real axis:

```
mf = mfinit([12,4],1); F = mfbasis(mf)[1];
mfeval(mf,F,1/Pi+10^(-6)*I)
mfeval(mf,F,1/Pi+10^(-7)*I)
mfeval(mf,F,1/Pi+10^(-8)*I)

% = -89811.049350396250531782882568405506024
    - 58409.940965200894541585402642924371696*I
% = 4.8212468504661113183253396691813292261 E-52
    + 6.7885262281520647908871247541561415340 E-52*I
% = 0
```

These results are immediate and **correct**: at height 10^{-6} the value is large, at height 10^{-7} very small (and really of the order of 10^{-52} with 30 correct decimals). Of course the value is not exactly 0 at height 10^{-8} but cannot be computed with 38 decimals default accuracy (simply increase the accuracy to **57D**, the value is of the order of 10^{-69}).

Evaluation of a Form II

Second, it can also evaluate forms at **cusps**:

```
T = mfTheta(); mf = mfinit(T); mfeval(mf,T,[0,1/2,1,oo])
```

```
% = [1/2 - 1/2*I, 0, 1/2 - 1/2*I, 1]
```

Warning: the value at a cusp is **not** the limit as τ tends to the cusp because of the automorphy factor $(c\tau + d)^{-k}$:

```
mfeval(mf,T,10^(-8)*I)
```

```
% = -7071.0678118654752440084436210484903928  
      + 2.407412430484044816 E-35*I
```

This number is equal to $-10^4\sqrt{2}/2$.

Evaluation of a Form II

Second, it can also evaluate forms at **cusps**:

```
T = mfTheta(); mf = mfinit(T); mfeval(mf,T,[0,1/2,1,oo])
```

```
% = [1/2 - 1/2*I, 0, 1/2 - 1/2*I, 1]
```

Warning: the value at a cusp is **not** the limit as τ tends to the cusp because of the automorphy factor $(c\tau + d)^{-k}$:

```
mfeval(mf,T,10^(-8)*I)
```

```
% = -7071.0678118654752440084436210484903928  
      + 2.407412430484044816 E-35*I
```

This number is equal to $-10^4\sqrt{2}/2$.

Evaluation of a Form II

Second, it can also evaluate forms at **cusps**:

```
T = mfTheta(); mf = mfinit(T); mfeval(mf,T,[0,1/2,1,oo])
```

```
% = [1/2 - 1/2*I, 0, 1/2 - 1/2*I, 1]
```

Warning: the value at a cusp is **not** the limit as τ tends to the cusp because of the automorphy factor $(c\tau + d)^{-k}$:

```
mfeval(mf,T,10^(-8)*I)
```

```
% = -7071.0678118654752440084436210484903928  
      + 2.407412430484044816 E-35*I
```

This number is equal to $-10^4\sqrt{2}/2$.

Periods and Symbols I

If F has weight $k \geq 2$ integral, a generalized **period** is the polynomial given by the integral

$$J(F; s_1, s_2) = \int_{s_1}^{s_2} (X - \tau)^{k-2} F(\tau) d\tau,$$

where s_j are points in the completed upper-half plane. In particular the coefficients give the integrals of $\tau^j F(\tau)$ for $0 \leq j \leq k - 2$.

Most important when s_j are cusps. Necessary precomputation of **symbols** (no need for the definition), then other computations immediate. Also necessary for **Petersson products**.

Periods and Symbols II

```
mf = mfinit([35,2],1); F = mfbasis(mf)[1];  
FS = mfsymbol(mf,F);  
mfsymboleval(FS,[0,oo])  
mfsymboleval(FS,[1/2,3/5])  
mfsymboleval(FS,[I,2*I])  
mfsymboleval(FS,[1/2,I])
```

```
% = 0.31404011074188471664161704390256378537*I  
% = -0.14296962919184795604253140534195291798  
    - 0.26199756419561033271653744806924309759*I  
% = 0.00088969563028739893631700037491116258378*I  
% = -0.61518300331940868645187865843466669894*I
```

Periods and Symbols II

```
mf = mfinit([35,2],1); F = mfbasis(mf)[1];  
FS = mfsymbol(mf,F);  
mfsymboleval(FS,[0,oo])  
mfsymboleval(FS,[1/2,3/5])  
mfsymboleval(FS,[I,2*I])  
mfsymboleval(FS,[1/2,I])
```

```
% = 0.31404011074188471664161704390256378537*I  
% = -0.14296962919184795604253140534195291798  
    - 0.26199756419561033271653744806924309759*I  
% = 0.00088969563028739893631700037491116258378*I  
% = -0.61518300331940868645187865843466669894*I
```

Periods and Symbols III

```
mf = mfinit([5,4],1); F = mfbasis(mf)[1];  
FS = mfsymbol(mf,F);  
mfsymboleval(FS,[0,oo])
```

```
% = 0.025682886503399670885091327035730701191*I*x^2  
+ 0.020865138644297634350206531603632923359*x  
- 0.0051365773006799341770182654071461402382*I
```

Note that `mfsymboleval` can also be applied to noncuspidal forms: in case of divergent integrals the result is a rational function or a polynomial of degree $k - 1$, which can easily be interpreted.

Periods and Symbols III

```
mf = mfinit([5,4],1); F = mfbasis(mf)[1];  
FS = mfsymbol(mf,F);  
mfsymboleval(FS,[0,oo])
```

```
% = 0.025682886503399670885091327035730701191*I*x^2  
+ 0.020865138644297634350206531603632923359*x  
- 0.0051365773006799341770182654071461402382*I
```

Note that `mfsymboleval` can also be applied to noncuspidal forms: in case of divergent integrals the result is a rational function or a polynomial of degree $k - 1$, which can easily be interpreted.

Periods and Symbols V

There also exist simpler functions `mfperiodpol` (integral from 0 to ∞) and `mfperiodpolbasis` (only in level 1):

```
# /* timer on */  
mf = mfinit([96,6],0); F = mfbasis(mf)[1];  
FS = mfsymbol(mf,F);  
mfsymboleval(FS,[0,oo]);  
mfperiodpol(mf,F);  
  
time = 24 ms.  
time = 9,477 ms.  
time = 0 ms.  
time = 76 ms.
```

(results on next page).

The `mfsymbol` computation requires 9.477 seconds, but the evaluation is instantaneous. If you only need the integral from 0 to ∞ , as here, no need for symbols, the computation requires only 0.076 seconds.

Periods and Symbols V

There also exist simpler functions `mfperiodpol` (integral from 0 to ∞) and `mfperiodpolbasis` (only in level 1):

```
# /* timer on */
mf = mfinit([96,6],0); F = mfbasis(mf)[1];
FS = mfsymbol(mf,F);
mfsymboleval(FS,[0,oo]);
mfperiodpol(mf,F);

time = 24 ms.
time = 9,477 ms.
time = 0 ms.
time = 76 ms.
```

(results on next page).

The `mfsymbol` computation requires 9.477 seconds, but the evaluation is instantaneous. If you only need the integral from 0 to ∞ , as here, no need for symbols, the computation requires only 0.076 seconds.

Periods and Symbols VI

```
% = 46.366702389191867463049266055452963967*I*x^4  
+ 3.8953700388682004473225316269956194525*x^3  
- 0.56826542231980277465186820072941104401*I*x^2  
- 0.15489398386891152199982272551206710377*x  
+ 0.024487897732315785610377476118978713061*I  
% = /* same result */
```

Recall the Petersson product in level N and weight k :

$$\langle F, G \rangle = \frac{1}{[\Gamma : \Gamma_0(N)]} \int_{\Gamma_0(N) \backslash \mathbb{H}} y^k F(\tau) \overline{G(\tau)} \frac{dx dy}{y^2} .$$

This is available for any two forms, even for non eigenforms or noncuspidal, as long as the integral converges; it **needs** the precomputation of symbols using **mfsymbol**. As usual, this precomputation may take some time, but the subsequent ones are essentially instantaneous.

Petersson Products II

```
mf = mfinit([96,4],0); [F1,F2] = mfbasis(mf);  
FS1 = mfsymbol(mf,F1); FS2 = mfsymbol(mf,F2);  
mfpetersson(FS1)  
mfpetersson(FS2)  
mfpetersson(FS1,FS2)
```

```
% = 0.00061471684149817788924091516302517391826
```

```
% = 0.0055324515734836010031682364672265652647
```

```
% = 1.6262535777977610381 E-40 + 1.2754930021943223828 E-41
```

The `mfsymbol` computations take each 2.5 seconds, but after everything is instantaneous. Note that `mfpetersson(FS,FS)` can be abbreviated to `mfpetersson(FS)`. Also, even though $F1$ and $F2$ are not eigenforms, the last result seem to show that they are orthogonal: this is true, prove it!

Petersson Products II

```
mf = mfinit([96,4],0); [F1,F2] = mfbasis(mf);  
FS1 = mfsymbol(mf,F1); FS2 = mfsymbol(mf,F2);  
mfpetersson(FS1)  
mfpetersson(FS2)  
mfpetersson(FS1,FS2)
```

```
% = 0.00061471684149817788924091516302517391826
```

```
% = 0.0055324515734836010031682364672265652647
```

```
% = 1.6262535777977610381 E-40 + 1.2754930021943223828 E-41
```

The `mfsymbol` computations take each 2.5 seconds, but after everything is instantaneous. Note that `mfpetersson(FS,FS)` can be abbreviated to `mfpetersson(FS)`. Also, even though $F1$ and $F2$ are not eigenforms, the last result seem to show that they are orthogonal: this is true, prove it!

Petersson Products III

Example of noncuspidal Petersson products:

```
mf12 = mfinit([12,5,-3]);  
E1 = mfeisenstein(5,1,-3);  
E2 = mfeisenstein(5,-3,1);  
cusps = mfcusps(12)  
[mfcuspval(mf12,E1,c) | c<-cusps]  
[mfcuspval(mf12,E2,c) | c<-cusps]
```

```
% = [ 0, 1/2, 1/3, 1/4, 1/6, 1/12]  
% = [ 0, 0, 1, 0, 1, 1]  
% = [1/3, 1/3, 0, 1/3, 0, 0]
```

`mfcuspval` computes the valuation of a form at a cusp. The above results show that at the six cusps of $\Gamma_0(12)$, one of the two Eisenstein series vanishes, so their Petersson product will converge.

Petersson Products III

Example of noncuspidal Petersson products:

```
mf12 = mfinit([12,5,-3]);  
E1 = mfeisenstein(5,1,-3);  
E2 = mfeisenstein(5,-3,1);  
cusps = mfcusps(12)  
[mfcuspval(mf12,E1,c) | c<-cusps]  
[mfcuspval(mf12,E2,c) | c<-cusps]
```

```
% = [ 0, 1/2, 1/3, 1/4, 1/6, 1/12]  
% = [ 0, 0, 1, 0, 1, 1]  
% = [1/3, 1/3, 0, 1/3, 0, 0]
```

`mfcuspval` computes the valuation of a form at a cusp. The above results show that at the six cusps of $\Gamma_0(12)$, one of the two Eisenstein series vanishes, so their Petersson product will converge.

Petersson Products III

```
P(mf) = mfpetersson(mfsymbol(mf,E1),mfsymbol(mf,E2));  
mf3 = mfinit([3,5,-3]); mf96 = mfinit([96,5,-3]);  
P(mf12)  
P(mf3);  
P(mf96);
```

```
time = 149 ms.  
% = -1.8848216716468969562647734582232071466 E-5  
      -1.9057659114817512165 E-43*I  
time = 16 ms.  
time = 4,412 ms.
```

Of course, because of the normalizing factor $1/[\Gamma : \Gamma_0(N)]$ all results are the same, but the required time increases very fast with the level (at least like its square).

Petersson Products III

```
P(mf) = mfpetersson(mfsymbol(mf,E1),mfsymbol(mf,E2));  
mf3 = mfinit([3,5,-3]); mf96 = mfinit([96,5,-3]);  
P(mf12)  
P(mf3);  
P(mf96);
```

```
time = 149 ms.  
% = -1.8848216716468969562647734582232071466 E-5  
      -1.9057659114817512165 E-43*I  
time = 16 ms.  
time = 4,412 ms.
```

Of course, because of the normalizing factor $1/[\Gamma : \Gamma_0(N)]$ all results are the same, but the required time increases very fast with the level (at least like its square).

Thank you for your attention !