# Basic Computation with Modular Forms

Ricky Magner

This tutorial will assume you either have Sage installed or have an account to use it online.

Let's start with the basic objects: spaces of modular forms.

```
sage:  M = ModularForms(1,12)
sage:  M
Modular Forms space of dimension 2 for Modular
Group SL(2,Z) of weight 12 over Rational Field
```

The first line establishes $M$ as the space of modular forms of level 1 and weight 12. In Sage, once an object is defined, you can type it and then hit Enter to see a short description of it. Doing this for $M$ gives the information on the third line, where we learn the dimension of $M$ (as a $\mathbb{C}$-vector space) and also the weight. But what else can we get from $M$? As usual in programming, when you have an object you can access certain variables describing it using a period, then some function. A function will end with parentheses, which is where you would put an argument into the function if desired/supported.

```
sage:  M.dimension()
2
sage:  M.basis()
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 +
O(q^6),
1 + 65520/691*q + 134250480/691*q^2 +
11606736960/691*q^3 + 274945048560/691*q^4 +
3199218815520/691*q^5 + O(q^6)
]
```

The first command is straightforward and should work for any kind of vector space object. The second is when things begin to get interesting. Sage can compute a basis of modular forms for our given level and weight instantly. So how do we start using them?

You can tell by the brackets that M.basis() returns a list of elements separated by a comma. We can access these elements by going back to $M$ and using the command .gen($n$) for the $n$th element, or just $M.n$. (Remember to start counting at 0 always!!) Let's try getting that cusp form out of there, while also showing off some more functions.

```
sage:  S = CuspForms(1,12); S
Cuspidal subspace of dimension 1 of Modular Forms
space of dimension 2 for Modular Group SL(2,Z) of
weight 12 over Rational Field
sage:  S.basis()
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 +
O(q^6)
]
sage:  Delta = S.0
sage:  Delta
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 +
O(q^6)
```

Note on the first line we performed two commands at once (defining $S$ and checking out its flavor text) by inserting a semicolon. You might have been tempted to use a command like "Delta = S.basis()" but that would have defined Delta as a *list* containing the function we want - which is totally different (obvious to anyone who knows the difference between a number and the singleton set containing that number).

Now that we've got the Delta function, what can we do with it? There are plenty of functions we can apply which can be found using a neat Sage feature. If we know we want to do something to Delta specifically, we can write "Delta." which looks like the start of a command. Maybe we're curious to see which begin with the letter $c$. Type "Delta.c" and hit the Tab key to see a full list of commands that begin with this string.

```
sage:  Delta.c
Delta.cartesian_product Delta.character
Delta.cuspform_lseries
Delta.category Delta.coefficients
```

Delta.coefficients looks interesting, but what does it do? It's a function based on the period placement syntax, so we can get a small description by typing it *without* parentheses (which is what you'd type to run the function).

```
sage:  Delta.coefficients
<bound method CuspidalSubmodule_level1_Q_with
_category.element_class.coefficients of q -
24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)>
```

Still seems mysterious, so you could try reading the documentation online about this function to try to see some examples. Or try it out yourself.

Speaking of coefficients, let's see more of the Delta function. We can do this in multiple ways. The first is to change the precision of all elements of our cuspform space $S$ (which only has one element for us) to see as many coefficients as we'd like. This will then automatically set the precision of all elements. Alternatively, we can do this manually using a few commands. Here we used the qexp command which will print the $q$-expansion of our form, but takes precision as an argument. Note this is temporary and reverts to the precision defined in $S$ after.

```
sage:  S.set_precision(3)
sage:  Delta
q - 24*q^2 + O(q^3)
sage:  Delta.qexp(8)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 -
6048*q^6 - 16744*q^7 + O(q^8)
sage:  Delta
q - 24*q^2 + O(q^3)
```

If you want to find specific coefficients without looking through a $q$-expension, just use brackets with the coefficient number to retrieve the value. Here we compute $\tau(17)$.

```
sage:  Delta[17]
-6905934
```

We have enough information about Sage to start verifying some claims made in class regarding relationships between modular forms. Let's get our tools prepared.

```
sage:  M.set_precision(5)
sage:  G12 = M.1; G12
1 + 65520/691*q + 134250480/691*q^2 +
11606736960/691*q^3 + 274945048560/691*q^4 +
O(q^5)
sage:  N = CuspForms(11,2)
sage:  N.dimension()
1
sage:  f11 = N.0; f11
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
```

The first thing we'll do is renormalize our Eisenstein series of weight 12 so that the coefficient of $q$ is 1. The syntax below might look strange if you don't have experience programming, but it means "assign the right value to the left variable."

```
sage:  G12 = 691*G12/65520
sage:  G12
691/65520 + q + 2049*q^2 + 177148*q^3 +
4196353*q^4 + O(q^5)
```

Now we can start comparing coefficients and see what relations hold. Letting $a(n)$ be the coefficients of $f_{11}$, we said that $a(n) \equiv \tau(n) \mod 11$, or "$f_{11} \equiv \Delta \mod 11$." Unfortunately it's not as easy in Sage to type something like "Delta mod 11" to get something we could compare to "f11 mod 11." However, here's some code we can use to verify the statement for the first twenty coefficients. (When writing code in Sage, end each statement with a colon, then hit Enter for the next line. Also, we use % for mod.)

```
sage:  for i in range(1,20):
....:  (f11[i]-Delta[i])%11
```

After running you should see a stream of 0's fly up the screen, which means $a(n) \equiv \tau(n) \mod 11$ for all $1 \leq n \leq 20$. Of course, if you want to run this for say $n$ up to 100, you'd probably prefer to not have to see 100 zeros on the screen. If you have experience programming you can easily modify the code in whatever way will convince you that the statement is true without having to deal with junk on screen (i.e. something like an if statement notifying you if there was a counterexample). Sage's native language for coding is Python so if you have experience with that it won't be hard to come up with something yourself.

Another claim was that $\tau(n) \equiv \sigma_{11}(n) \mod 691$, i.e. "$\Delta \equiv G_{12} \mod 691$." In this case, $G_{12}$ and $\Delta$ belong to the same space, so we can take their difference and just look at the coefficient of that form mod 691. But wait, why didn't we do that before? We know we can subtract Fourier series just fine. But $f_{11}$ came from CuspForms(11,2) and $\Delta$ came from CuspForms(1,12) which is a subspace of ModularForms(1,12), so from Sage's perspective they live in different worlds - you cannot subtract them. We would have gotten an error if we tried.

```
sage:  h = Delta-G12
sage:  h
-691/65520 - 2073*q^2 - 176896*q^3 - 4197825*q^4
+ O(q^5)
sage:  for i in range(1,20):
....:  h[i]%691
```

Zeros all the way through!

Lastly, there was a mysterious connection between $f_{11}$ and a certain elliptic curve. Let's try to verify the claims made.

```
sage:  E11 = EllipticCurve([0,-1,1,0,0]); E11
Elliptic Curve defined by y^2 + y = x^3 - x^2
over Rational Field
```

We use the function Np($p$) to count the number of points on $E_{11}$ mod $p$.

```
sage:  E11.Np(3)
5
sage:  E11.Np(11)
11
```

Recall the trace of Frobenius is defined by $a_\ell = \ell + 1 - \#E(\mathbb{F}_\ell)$. We can make a table (you might make it look fancier with better code) comparing this value to $a(\ell)$, the $\ell$th coefficient of $f_{11}$, for the first 10 primes.

```
sage:  for i in primes_first_n(10):
....:  print(i, f11[i], i+1-E11.Np(i))
....:
(2, -2, -2)
(3, -1, -1)
(5, 1, 1)
(7, -2, -2)
(11, 1, 1)
(13, 4, 4)
(17, -2, -2)
(19, 0, 0)
(23, -1, -1)
(29, 0, 0)
```

It seems like $a_\ell = a(\ell)$ ! If they are equal for all $\ell$, then their $L$-functions must be equal. Let's use Sage to compare their $L$-function values at $s = 1$. Unfortunately, trying to run the command "f11.lseries()" causes an error, but the message is helpful. It suggests "using a newform constructor" instead, so let's try grabbing $f_{11}$ from a newform vector space.

```
sage:  A = Newforms(11,2); A
[q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)]
sage:  f11 = A[0]; f11
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
```

Here our space $A$ didn't have a *.gen* function associated to it. In fact, using Tab to look at the functions, it seems like $A$ was just a plain old list! So

to access its elements, we used brackets as shown above. Now let's compute the $L$-series and compare.

```
sage:  Lf = f11.lseries(); LE = E11.lseries()
sage:  Lf(1)
0.253841860855911
sage:  LE(1)
0.253841860855911
```

The two functions seem to agree at 1. Checking other random values seems to suggest they are the same! Also, because $L(E, 1) \neq 0$, shouldn't $E(\mathbb{Q})$ be finite?...

```
sage:  E11.rank()
0
```

Everything seems to work! But then looking at the documentation for how Sage computes the rank of elliptic curves, we see "IMPLEMENTATION: Uses L-functions, mwrank, and databases," so we can't be sure this confirms our $L$-function value was reasonable if that's used to prove the rank of $E_{11}$ is 0.