USING `ETAEXPRESSIONS`

LEWIS COMBES

1. Introduction

Let η be the Dedekind eta function. The Magma code in the `EtaExpressions` repo allows expressions for classical modular forms to be computed in terms of η. This code was used to compute all the examples given in the paper [1]. This repo also contains data files with eta quotients for some small levels and weight $k = 2$. More such examples can be computed using the code of Rouse and Webb, available at

`https://users.wfu.edu/rouseja/eta/`

The repo consists of the following files:

- `is_in_span_of_quots.m` — Tells you whether a given form is in the span of the list of eta quotients provided.
- `minimal_expression_below.m` — Finds (at least in principle!) the minimal eta expression for your given form. In practice, it runs very slowly because it performs exhaustive checks, which take a very long time. Not useful for larger computation.
- `minimal_expression_above.m` — Finds progressively smaller eta expressions for your chosen form using a random algorithm. Not guaranteed to find anything close to minimal, but often does a decent job.
- `check_expression.m` — If an expression is stored in Magma as a list `quots` of eta quotients and a list `coeffs` of coefficients, this code can be used to check that the given form really does equal the expression given. This is especially useful for all the expressions stored in the `big_expressions` folder.

1.1. **Acknowledgements.** This code was written in preparation for the University of Sheffield Undergraduate Research Internship project *"Eta expressions associated to elliptic curves"*. The project was funded by the Heilbronn Institute for Mathematical

research. The computations were all performed on the University of Sheffield Magma server.

2. Getting started

The most important thing to note: none of this code computes eta quotients themselves. These are a necessary ingredient, and one you will need to provide for yourself. The code of Rouse and Webb noted above is the best place to go for this. From here, we will assume a list of quotients has already been computed (even if this is a tall order for even modest levels and weights!)

The quotients should be fed into the code as a list called `quots`.

3. Examples

Example 3.1. We begin with a simple example: the computation of a minimal eta expression for the form f with LMFDB label `30.2.a.a`. To do this, we load the file `30.txt`, containing 272 eta quotients. We would first like to know if f has an eta quotient at level 30. To do this, we use `is_in_span_of_quots.m` with the parameters

- `N:=30;`
- `d:=1;`
- `k:=2;`
- `HeckeKernels:=[ ];`

Note, since f is the only new cusp form at level 30 and weight 2, we don't need to include any information in `HeckeKernels` in order to pick it out. Running the file, the important point is given by the final line

```
v in sub<RR|vecs>;
```

which returns `true`, so we can find an eta expression at this level.

At this point, if we were not interested in minimality, we could simply write

```
Solution(Matrix(vecs),v);
```

in the terminal and be done with it. In this instance, we get the vector

```
(0 20/9 -31/18 5/9 -10/3 -1/6 0 -7/9 29/9 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 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 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 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 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 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 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -45)
```

which is encouraging, since it is mostly zeroes. It has 8 non-zero entries, which is quite good, but we want to know if we can do better. In this case, we can use `minimal_expression_below.m` to exhaustively check all combinations of 2 eta quotients, then 3, and so on, each time checking if f is in their span. This method spirals out of control very quickly, and so we limit the combinations we check with a big number limit. Since we have 272 eta quotients, and our big number limit is 10^8, we can check all combinations of up to 3 eta quotients.

Running `minimal_expression_below.m`, we find 14 pairs of eta quotients with f in their span, which are stored in the variable `expressions`. These are sorted with respect to their height, and we can view them all with the code commented out at the bottom of the file. There are 7 expressions with height 0, at which point it is a matter of personal taste to select.

The vector returned from the simple check with `is_in_span_of_quots.m` is therefore improved to the tune of 6 eta quotients.

Example 3.2. We will now perform a more complicated example. We want to express the form g with LMFDB label `43.2.a.a` in terms of eta quotients. Looking at our data files, we note that there are no eta quotients in the space $M_2(\Gamma_0(43))$, meaning we will have to go further afield to find an expression. Indeed, we check the levels $43d$, with $d = 1, 2, \ldots$ using `is_in_span_of_quots.m`, until we find a space that returns `true`.

We find that the first such space is when $d = 8$, so that $43d = 344$. In this space, there are 77 eta quotients. So our big number limit means we can check combinations of up to 5 eta quotients to try and locate g. This is done using `minimal_expression_below.m` as before.

Letting this crunch the numbers for a while, we see that there is no combination of 5 or fewer eta quotients containing g in their span. In theory we could let the code check larger numbers of combinations, but in its current iteration this cannot be done without lots of memory, so we now use the complementary method of `minimal_expression_above.m` to try and bound this number from above.

Again, this is a simple matter of setting the code to run and finding something else to do while it chugs along. The first expression it returns (when we wrote this document, anyway) is of length 34. However, this is quickly replaced with further expressions of smaller height[1] and length. This seems to be a common phenomenon when using this code: the first expression is usually much longer than it needs to be, and smaller expressions are found quickly; as the length and height of the record-holder decrease, smaller expressions are rarer, and so new records are found less frequently.

Exercise 3.3. Develop a reasonable statistical model for the distribution of lengths of eta expressions of a given form f, and use the incidence of new record-holders to approximate the probability that the minimal expression has been found. And if you do that, please let me know how, because I would be very interested to hear about it!

Eventually the code settles down into the expression of length 20 and height about 34.4074 that can be found in the `big_expressions` folder in this repo. Of course, it is impossible to know if this expression is minimal. Even if the code sits on this expression for a very long time, there is no guarantee a smaller one is simply evading capture without checking all possible combinations of a smaller number of quotients.

Example 3.4. We can also use the code to check the big expressions in the `data` folder. For example, the expression for the form h with LMFDB label `61.2.a.a` we list has length 101, with large coefficients. There is precious little hope of checking this by hand, so we use `check_expression.m`. It uses the same parameters to specify the form and level, and loading in `61.2.a.a.txt`, it is a simple matter of checking whether the sum of the quotients (considered as abstract vectors) with the coefficients is equal to (the abstract vector representing) h. This is done in the final line

[1]This is the total height of the coefficients, see the paper [1]

```
&+[vecs[i]*coeffs[i] :  i in [1..#vecs]] eq v;
```

which, thankfully, returns `true`.

References

[1] E. Chan and L. Combes. *Expressions for weight 2 cusp forms in holomorphic eta quotients.*