2015-05-29-105121-billeray

William Stein

5/29/2015

Contents

0.1 Author of this worksheet: William Stein 1
0.1.1 Question in email on May 28, 2015 from Nicolas Billerey 1

0.1 Author of this worksheet: William Stein

0.1.1 Question in email on May 28, 2015 from Nicolas Billerey

Hi William, I suspect that there exists a CM form of weight 12, level 23^2 and trivial Nebentypus character which is congruent to Δ mod 23. Similarly, I suspect that there also exists a CM form of weight 16, level 31^2 and trivial Nebentypus congruent mod 31 to the unique newform of wt 16, level 1 and rational integer coefficients.

First want to check there is nothing at level 23, so what we find below is at level 23^2

```
%time M = ModularSymbols(23, base_ring=GF(23), weight=12, sign=1)
```
```
CPU time: 0.14 s, Wall time: 0.15 s
```

```
d = delta_qexp(20); show(d)
```
$q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 - 6048q^6 - 16744q^7 + 84480q^8 - 113643q^9 - 115920q^{10} + 534612q^{11} - 370944q^{12} - 577738q^{13} + 401856q^{14} + 1217160q^{15} + 987136q^{16} - 6905934q^{17} + 2727432q^{18} + 10661420q^{19} + O(q^{20})$

```
%time T2 = M.hecke_operator(2)
```
```
CPU time: 0.01 s, Wall time: 0.01 s
```

```
%time V2 = (T2 - d[2]).kernel()
```
```
CPU time: 0.01 s, Wall time: 0.01 s
```

The following dimension shows that we have only the two images of delta under degen map, namely delta(q) and delta($q^2$3).*Nothingelseispossible*.

```
V2.dimension()
```
```
2
```

OK, time to look into level 23^2. We know a priori that we will have $\Delta(q)$, $\Delta(q^{23})$, and $\Delta(q^{23^2})$, the images under the degeneracy maps corresponding to the divisors of 23^2. So the question is whether or not there is a fourth form.

In the calculation below we will work modulo 23 at level 23^2 and deduce that there is such a fourth form (and no others). However, the mod 23 calculation doesnt tell us anything else about that form just that it exists!

Incidentally, working mod a prime like below is fine as long as the prime isnt 2 or 3 if it is, then there are problems.

```
import sage_server; sage_server.MAX_OUTPUT_MESSAGES = 10000  # below may \
   generate a lot of output
```

```
# this is a nontrivial sparse linear algebra computation over a small \
   finite field, which should take 10s.
%time M = ModularSymbols(23^2, base_ring=GF(23), weight=12, sign=1)
M
```

```
CPU time: 10.20 s, Wall time: 10.66 s

Modular Symbols space of dimension 507 for Gamma_0(529) of weight 12 with sign 1 over
Finite Field of size 23
```

```
d = delta_qexp(600); show(d[:15])
```

$q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 - 6048q^6 - 16744q^7 + 84480q^8 - 113643q^9 - 115920q^{10} + 534612q^{11} - 370944q^{12} - 577738q^{13} + 401856q^{14} + O(q^{600})$

```
%time T2 = M.hecke_operator(2)
```

```
CPU time: 0.02 s, Wall time: 0.02 s
```

```
%time V2 = (T2 - d[2]).kernel()
```

```
CPU time: 0.39 s, Wall time: 0.49 s
```

```
# positive evidence
V2.dimension()
```

```
4
```

```
%time V3 = (V2.hecke_operator(3) - d[3]).kernel()
```

```
CPU time: 0.33 s, Wall time: 0.33 s
```

```
# more positive evidence
V3.dimension()
```

```
4
```

```
%time V5 = (V3.hecke_operator(5) - d[5]).kernel()
```

```
CPU time: 0.37 s, Wall time: 0.37 s
```

```
# even more evidence
V5.dimension()
```

```
4
```

To prove we have a congruence with a newform we have to check up to the Sturm bound:

```
M.sturm_bound()
```

552

Thats big and could be done in the following straightforward way, which would take a day. However, Im too impatient and there is a trick that is massively faster (reducing an O(N) computation to O(1)).

```
%time  # so get overall time
d = delta_qexp(M.sturm_bound()+1)
V = M
p = 2
while p <= M.sturm_bound() and V.dimension()>1:
    print "working on p=%s"%p; sys.stdout.flush()
    if p != 23:
        %time V = (V.hecke_operator(p) - d[p]).kernel()
        print p, V.dimension()
    else:
        print "ignoring p=23 (for now)"
    p = next_prime(p)
    if p > 30:
        # THIS IS TOO SLOW FOR ME... but should work -- don't run.
        print "you should be morally convinced by now"
        break
```

```
working on p=2
CPU time: 0.10 s, Wall time: 0.10 s
2 4
working on p=3
CPU time: 0.00 s, Wall time: 0.01 s
3 4
working on p=5
CPU time: 0.00 s, Wall time: 0.00 s
5 4
working on p=7
CPU time: 0.43 s, Wall time: 0.43 s
7 4
working on p=11
CPU time: 0.59 s, Wall time: 0.61 s
11 4
working on p=13
CPU time: 0.68 s, Wall time: 0.69 s
13 4
working on p=17
CPU time: 0.79 s, Wall time: 0.80 s
17 4
working on p=19
CPU time: 0.82 s, Wall time: 0.83 s
19 4
working on p=23
ignoring p=23 (for now)
working on p=29
CPU time: 1.10 s, Wall time: 1.17 s
29 4
you should be morally convinced by now
CPU time: 4.52 s, Wall time: 4.66 s
```

```
d = delta_qexp(M.sturm_bound()+1)
```

What do the matrices look like on this 4-dimensional space?

```
V5
```

```
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 507 for
Gamma_0(529) of weight 12 with sign 1 over Finite Field of size 23
```

```
show(V5.hecke_matrix(13))
```

$$\begin{pmatrix} 22 & amp;0 & amp;0 & amp;0 \\ 0 & amp;22 & amp;0 & amp;0 \\ 0 & amp;0 & amp;22 & amp;0 \\ 0 & amp;0 & amp;0 & amp;22 \end{pmatrix}$$

I bet they are all diagonal except for T_{23}.

```
T23 = V5.hecke_matrix(23)
show(T23)
show(T23.fcp())
```

$$\begin{pmatrix} 6 & amp;4 & amp;0 & amp;14 \\ 3 & amp;2 & amp;0 & amp;7 \\ 16 & amp;3 & amp;0 & amp;22 \\ 20 & amp;21 & amp;0 & amp;16 \end{pmatrix}$$

$$(x + 22) \cdot x^{3}$$

```
d[23] % 23
```

```
1
```

Use a trick to reduce complexity dramatically

```
%time T2 = M.dual_hecke_matrix(2)
```

```
CPU time: 0.01 s, Wall time: 0.01 s
```

```
%time V2dual = (T2 - d[2]).kernel()
```

```
CPU time: 0.06 s, Wall time: 0.06 s
```

```
V2dual.dimension()
```

```
4
```

```
B = V2dual.free_module().basis()
# we need 4 easy-to-compute with basis elements that dot nonzero with our\
    basis.
guess = [M.gen(i).element() for i in [0..3]]
A = matrix(4,4, [B[i].dot_product(guess[j]) for i in range(4) for j in \
   range(4)])
show(A)
```

$$\begin{pmatrix} 1 & amp;0 & amp;0 & amp;0 \\ 0 & amp;1 & amp;0 & amp;0 \\ 0 & amp;0 & amp;1 & amp;0 \\ 0 & amp;0 & amp;0 & amp;1 \end{pmatrix}$$

```
# we got super lucky in this case.  The above means that computing
```

```
# the action of Tp on M.gen(0),..., M.gen(3) will give the same matrix
# as computing the action on our 4-dimensional subspace.
# But surprisingly it is massively easier...
# In general would have to invert a matrix and multiply by that.
# This should really be implemented in general in Sage, but evidently isn\
   't...
```

```
Tp = M.hecke_operator(37)
%time x = Tp.apply_sparse(M.gen(0)) # fast even if "37" is large...
```

```
CPU time: 0.02 s, Wall time: 0.02 s
```

```
def fastT(p):
    Tp = M.hecke_operator(p)
    C = [Tp.apply_sparse(M.gen(i)).element() for i in range(4)]  # the \
       real work
    return matrix(4,4,[b.dot_product(c) for b in B for c in C])
```

```
fastT(2).fcp()
```

```
(x + 1)^4
```

```
# compare with
V2.hecke_matrix(2).fcp()
```

```
(x + 1)^4
```

```
# try bigger
%time show(fastT(37))
```

$$\begin{pmatrix} 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \end{pmatrix}$$

```
CPU time: 0.12 s, Wall time: 0.16 s
```

```
%time show(V2.hecke_matrix(37))
```

$$\begin{pmatrix} 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \\ 0 & amp;0 & amp;0 & amp;0 \end{pmatrix}$$

```
CPU time: 1.25 s, Wall time: 1.32 s
```

```
show(fastT(23).fcp())
```

$$(x + 22) \cdot x^3$$

```
show(V2.hecke_matrix(23).fcp())
```

$$(x + 22) \cdot x^3$$

So now we do the hard computation above for primes up to the Sturm bound, but using our fastT:

```
%time  # so get overall time
d = delta_qexp(M.sturm_bound()+1)
V = M
```

```
p = 2
while p <= M.sturm_bound():
    print "doing p=%s"%p; sys.stdout.flush()
    if p != 23:
        %time Tp = fastT(p)
        if Tp != d[p]:  # the right scalar matrix
            print "FAIL at p=%s"%p
    else:
        print "ignoring p=23"
    p = next_prime(p)
```

```
doing p=2
CPU time: 0.09 s, Wall time: 0.09 s
doing p=3
CPU time: 0.08 s, Wall time: 0.11 s
doing p=5
CPU time: 0.09 s, Wall time: 0.10 s
doing p=7
CPU time: 0.09 s, Wall time: 0.09 s
doing p=11
CPU time: 0.08 s, Wall time: 0.08 s
doing p=13
CPU time: 0.08 s, Wall time: 0.09 s
doing p=17
CPU time: 0.08 s, Wall time: 0.08 s
doing p=19
CPU time: 0.10 s, Wall time: 0.10 s
doing p=23
ignoring p=23
doing p=29
CPU time: 0.08 s, Wall time: 0.09 s
doing p=31
CPU time: 0.09 s, Wall time: 0.09 s
doing p=37
CPU time: 0.10 s, Wall time: 0.10 s
doing p=41
CPU time: 0.10 s, Wall time: 0.10 s
doing p=43
CPU time: 0.10 s, Wall time: 0.10 s
doing p=47
CPU time: 0.10 s, Wall time: 0.10 s
doing p=53
CPU time: 0.10 s, Wall time: 0.10 s
doing p=59
CPU time: 0.10 s, Wall time: 0.11 s
doing p=61
CPU time: 0.12 s, Wall time: 0.12 s
doing p=67
CPU time: 0.11 s, Wall time: 0.11 s
doing p=71
CPU time: 0.11 s, Wall time: 0.12 s
doing p=73
```

```
CPU time: 0.12 s, Wall time: 0.12 s
doing p=79
CPU time: 0.12 s, Wall time: 0.13 s
doing p=83
CPU time: 0.13 s, Wall time: 0.14 s
doing p=89
CPU time: 0.12 s, Wall time: 0.13 s
doing p=97
CPU time: 0.19 s, Wall time: 0.20 s
doing p=101
CPU time: 0.13 s, Wall time: 0.13 s
doing p=103
CPU time: 0.13 s, Wall time: 0.14 s
doing p=107
CPU time: 0.16 s, Wall time: 0.17 s
doing p=109
CPU time: 0.14 s, Wall time: 0.14 s
doing p=113
CPU time: 0.19 s, Wall time: 0.20 s
doing p=127
CPU time: 0.23 s, Wall time: 0.25 s
doing p=131
CPU time: 0.14 s, Wall time: 0.14 s
doing p=137
CPU time: 0.14 s, Wall time: 0.15 s
doing p=139
CPU time: 0.15 s, Wall time: 0.15 s
doing p=149
CPU time: 0.21 s, Wall time: 0.22 s
doing p=151
CPU time: 0.16 s, Wall time: 0.16 s
doing p=157
CPU time: 0.15 s, Wall time: 0.15 s
doing p=163
CPU time: 0.16 s, Wall time: 0.16 s
doing p=167
CPU time: 0.16 s, Wall time: 0.16 s
doing p=173
CPU time: 0.16 s, Wall time: 0.16 s
doing p=179
CPU time: 0.16 s, Wall time: 0.16 s
doing p=181
CPU time: 0.16 s, Wall time: 0.17 s
doing p=191
CPU time: 0.16 s, Wall time: 0.17 s
doing p=193
CPU time: 0.17 s, Wall time: 0.17 s
doing p=197
CPU time: 0.17 s, Wall time: 0.17 s
doing p=199
CPU time: 0.17 s, Wall time: 0.18 s
```

```
doing p=211
CPU time: 0.18 s, Wall time: 0.18 s
doing p=223
CPU time: 0.18 s, Wall time: 0.19 s
doing p=227
CPU time: 0.18 s, Wall time: 0.19 s
doing p=229
CPU time: 0.24 s, Wall time: 0.24 s
doing p=233
CPU time: 0.24 s, Wall time: 0.25 s
doing p=239
CPU time: 0.20 s, Wall time: 0.20 s
doing p=241
CPU time: 0.20 s, Wall time: 0.20 s
doing p=251
CPU time: 0.19 s, Wall time: 0.20 s
doing p=257
CPU time: 0.20 s, Wall time: 0.21 s
doing p=263
CPU time: 0.21 s, Wall time: 0.22 s
doing p=269
CPU time: 0.25 s, Wall time: 0.26 s
doing p=271
CPU time: 0.20 s, Wall time: 0.21 s
doing p=277
CPU time: 0.23 s, Wall time: 0.23 s
doing p=281
CPU time: 0.29 s, Wall time: 0.30 s
doing p=283
CPU time: 0.22 s, Wall time: 0.23 s
doing p=293
CPU time: 0.21 s, Wall time: 0.23 s
doing p=307
CPU time: 0.22 s, Wall time: 0.23 s
doing p=311
CPU time: 0.22 s, Wall time: 0.22 s
doing p=313
CPU time: 0.22 s, Wall time: 0.23 s
doing p=317
CPU time: 0.29 s, Wall time: 0.30 s
doing p=331
CPU time: 0.24 s, Wall time: 0.25 s
doing p=337
CPU time: 0.26 s, Wall time: 0.26 s
doing p=347
CPU time: 0.24 s, Wall time: 0.24 s
doing p=349
CPU time: 0.26 s, Wall time: 0.27 s
doing p=353
CPU time: 0.25 s, Wall time: 0.26 s
doing p=359
```

```
CPU time: 0.25 s, Wall time: 0.30 s
doing p=367
CPU time: 0.26 s, Wall time: 0.33 s
doing p=373
CPU time: 0.25 s, Wall time: 0.26 s
doing p=379
CPU time: 0.26 s, Wall time: 0.26 s
doing p=383
CPU time: 0.37 s, Wall time: 0.38 s
doing p=389
CPU time: 0.46 s, Wall time: 0.49 s
doing p=397
CPU time: 0.37 s, Wall time: 0.45 s
doing p=401
CPU time: 0.31 s, Wall time: 0.31 s
doing p=409
CPU time: 0.33 s, Wall time: 0.34 s
doing p=419
CPU time: 0.30 s, Wall time: 0.30 s
doing p=421
CPU time: 0.37 s, Wall time: 0.38 s
doing p=431
CPU time: 0.28 s, Wall time: 0.29 s
doing p=433
CPU time: 0.28 s, Wall time: 0.28 s
doing p=439
CPU time: 0.28 s, Wall time: 0.29 s
doing p=443
CPU time: 0.30 s, Wall time: 0.31 s
doing p=449
CPU time: 0.30 s, Wall time: 0.31 s
doing p=457
CPU time: 0.30 s, Wall time: 0.31 s
doing p=461
CPU time: 0.29 s, Wall time: 0.30 s
doing p=463
CPU time: 0.29 s, Wall time: 0.30 s
doing p=467
CPU time: 0.29 s, Wall time: 0.31 s
doing p=479
CPU time: 0.30 s, Wall time: 0.31 s
doing p=487
CPU time: 0.30 s, Wall time: 0.31 s
doing p=491
CPU time: 0.38 s, Wall time: 0.39 s
doing p=499
CPU time: 0.33 s, Wall time: 0.34 s
doing p=503
CPU time: 0.36 s, Wall time: 0.37 s
doing p=509
CPU time: 0.32 s, Wall time: 0.33 s
```

```
doing p=521
CPU time: 0.39 s, Wall time: 0.40 s
doing p=523
CPU time: 0.40 s, Wall time: 0.41 s
doing p=541
CPU time: 0.31 s, Wall time: 0.32 s
doing p=547
CPU time: 0.50 s, Wall time: 0.60 s

CPU time: 21.27 s, Wall time: 22.28 s
```

So in less than minute total CPU time weve verified your first claim using exactly the right algorithms

Some stuff involving new subspaces I left around New subspaces with modular symbols behave funny in characteristic p.

```
set_verbose(2)
%time Mn = M.new_subspace()
```

```
%time V2new = V2.new_subspace()
```

```
V2new
```
```
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 507 for
Gamma_0(529) of weight 12 with sign 1 over Finite Field of size 23
```

```
V2new.hecke_operator(23).fcp()
```
```
verbose 1 (6534: free_module.py, echelon_coordinates) mod-p multiply of 1 x 2 matrix by 2
x 507 matrix modulo 23
verbose 1 (6534: free_module.py, echelon_coordinates) mod-p multiply of 1 x 2 matrix by 2
x 507 matrix modulo 23
verbose 1 (579: matrix_morphism.py, characteristic_polynomial) _charpoly_linbox...
x^2
```

```
%time D = V2new.dual_free_module(bound=5)
```