A MAGMA SOURCE FOR DEALING ARITHMETIC ASPECTS OF $X_0^*(N)$

1. A Magma programme to compute arithmetic aspects of $X_0^*(N)$, with N square-free

Programme in Magma V2-23.9, (and working in Magma Calculator Online in October 2018) to compute the number of elements of the set $X_0^*(N)(\mathbb{F}_{p^n})$, obtain $Q_p(k)$ with k odd in [1], and given E an elliptic curve over $\mathbb{Q}$ with conductor $M|N$ to compute two times the number of $\mathbb{F}_{p^n}$-points of E, always $p \nmid N$ prime. Here always N is square-free integer.

To compute $Q_p(k)$ is needed to compute $P_p(k) \in \{0,1\}$, see also the details of such elements in the paper "Bielliptic modular curves $X_0^*(N)$ with N square-free levels by Francesc Bars and Josep González.

Input: Introduce in the first line N a square-free level, p a prime with $p \nmid N$, and the a_p-coefficient of the q-expansion of an elliptic curve E over $\mathbb{Q}$ of conductor M with $M|N$ such that at level M all the Atkin-Lehner involution acts as $+1$ (this elliptic curve appears as a 1-dimensional factor in the Jacobian decomposition of $\mathrm{Jac}(X_0^*(N))$ over $\mathbb{Q}$).

Output:

(1) PointsOfXzerostarGFp:=$[\#X_0^*(N)(\mathbb{F}_p), \ldots, \#X_0^*(N)(\mathbb{F}_{p^{20}})]$,
(2) N,
(3) `ValueofP_p` $:=[P_p(1), P_p(2), \ldots, P_p(20)]$,
(4) k, (is the biggest odd integer ≤ 20 such that $P_p(k) = 1 \in \{0,1\}$),
(5) $Q_p(k)$.
(6) DoubleNumberPointsofEprimep:=$[2 \cdot \#E(\mathbb{F}_p), \ldots, 2 \cdot \#E(\mathbb{F}_{p^{20}})]$.

If one wish to replace 20 for another integer, one can modify 20 in the next programme source with a positive integer where he expects that $Q_p(M)$ will increase, or $2 \cdot \#E(\mathbb{F}_{p^n})$ will be smaller than $\#X_0^*(N)(\mathbb{F}_{p^n})$.

Remember that $\mathrm{Aut}(X_0^*(N))$ is trivial if $Q_p(k)$ for some k odd and p prime with $p \nmid N$ the quantity $Q_p(k) > 2g_N^* + 2$ following [1], where g_N^* denotes de genus of $X_0^*(N)$.

Magma code:

We use level $N = 555$, $p = 2$ and $E = 185a$ (where $a_2(185a) = -2$, from Cremona tables).

```
N=:555;

p=:2;

a_p_E:=-2;


invariant_eigenforms := [* *]; number_fields:=[* *];

for conductor in Divisors(N) do
```

```
  for decomposition_factor in
          NewformDecomposition(CuspidalSubspace(ModularSymbols(conductor,2,1))) do
    eigenform := Eigenform(decomposition_factor, 20);
    number_field_of_eigenform := Parent(Coefficient(eigenform, 3));

    all_atkin_lehners_act_as_identity := true;
    for prime_dividing_conductor in PrimeDivisors(conductor) do
      if AtkinLehner(decomposition_factor, prime_dividing_conductor) ne
                  IdentityMatrix(Rationals(), Dimension(decomposition_factor)) then
        all_atkin_lehners_act_as_identity := false;
      end if;
    end for;

    if all_atkin_lehners_act_as_identity then
      invariant_eigenforms:= Append(invariant_eigenforms, eigenform);
      number_fields:= Append(number_fields, number_field_of_eigenform);
    end if;
  end for;
end for;


C:=ComplexField(100);

R<x>:=PolynomialRing(C);

FrobpolynJacobian:=0*x+1;

RootsFrobactJacob:=[* *];

for j in [1 .. #number_fields] do
   if Degree(number_fields[j]) eq 1 then
       Rootsellipticfactor:=Roots(x^2-Coefficient(invariant_eigenforms[j],p)*x+p,C);
       RootsFrobactJacob:=Append(RootsFrobactJacob,Rootsellipticfactor);
       FrobpolynJacobian:=FrobpolynJacobian*(x^2-Coefficient(invariant_eigenforms[j],p)*x+p);
   else
      dd:=Degree(number_fields[j]);
      u:=Roots(DefiningPolynomial(number_fields[j]),C);
      for m in [1 .. #u] do
         f := hom< number_fields[j] -> C | u[m][1]>;
         cc2:=Roots(x^2-f(Coefficient(invariant_eigenforms[j],p))*x+p,C);
         RootsFrobactJacob:=Append(RootsFrobactJacob,cc2);
         FrobpolynJacobian:=FrobpolynJacobian*(x^2-f(Coefficient(invariant_eigenforms[j],p))*x+p);
      end for;
  end if;
end for;

PointsOfXzerostarGFp:=[* *];

for nn in [1 .. 20] do
   SumofpowerofFrobpolyn:=0;
```

```
   for i in [1 .. # RootsFrobactJacob] do
      for j in [1..2] do
        if RootsFrobactJacob[i][j][2] gt 0 then
         SumofpowerofFrobpolyn:=
         SumofpowerofFrobpolyn+(RootsFrobactJacob[i][j][2])*(RootsFrobactJacob[i][j][1])^(nn) ;
        else
          SumofpowerofFrobpolyn:=SumofpowerofFrobpolyn;
        end if;
      end for;
   end for;

  PointsXzerostarpnn:=Round(1+p^(nn)-SumofpowerofFrobpolyn);

  PointsOfXzerostarGFp:=Append(PointsOfXzerostarGFp,PointsXzerostarpnn);
end for;

PointsOfXzerostarGFp;

N;


ValueofP_p:=[* *];

for aaa in [1..20] do
 sumdivisorsMUbyPointszerostar:=0;
    for kk in Divisors(aaa) do
       vv:=aaa/kk;
       vv:=Numerator(vv);
       sumdivisorsMUbyPointszerostar:=
       sumdivisorsMUbyPointszerostar+(MoebiusMu(vv))*(PointsOfXzerostarGFp[kk]);
     end for;

  vvv:=sumdivisorsMUbyPointszerostar/aaa;
  Rr:=Integers(2);
  P_p_aaa:=Rr!vvv;
  ValueofP_p:=Append(ValueofP_p,P_p_aaa);
end for;

ValueofP_p;




Q_p_odd:=0;

odd_number:=0;

for t in [1..#ValueofP_p] do
  if ValueofP_p[t] eq 1 then
     tred:=Rr!t;
     if tred eq 1 then
```

```
     Q_p_odd:=Q_p_odd+t;
     odd_number:=t;
     else
     Q_p_odd:=Q_p_odd;
     end if;
  else
     Q_p_odd:=Q_p_odd;
  end if;
end for;

odd_number;

Q_p_odd;



DoubleNumberPointsofEprimep:=[* *];

RootsofFrobactE:=Roots(x^2-a_p_E*x+p,C);

for i in [1..20] do

Twotimesp_i_points_E:=2*(p^i+1-Round(RootsofFrobactE[1][1]^i+
p^i/RootsofFrobactE[1][1]^i));

DoubleNumberPointsofEprimep:=Append(DoubleNumberPointsofEprimep,Twotimesp_i_points_E);

end for;

DoubleNumberPointsofEprimep;
```

2. A Magma programme to compute the genus of $X_0^*(N)$

Here we provide Magma code, (which was working on Magma Calculator Online, October 2018), to compute the genus of $X_0^*(N)$ for arbitrary N, moreover we introduce different functions which may be usual for different arithmetic aspects with respect to quotients of $X_0(N)$ by subgroups generated by Atkin-Lehner involutions.

The general setting is: fix $N \geq 2$ an integer and consider the modular curve $X_0(N)$, and consider $W(N)$ the group generated by all the Atkin-Lehner involutions of $X_0(N)$, group of order 2^r where r is the number of different primes that divides N, and fix W a subgroup of $W(N)$ of order 2^s, we recall that $X_0^*(N)$ denotes $X_0(N)/W(N)$, $X_0^+(N) := X_0(N)/ < w_N >$, $X_0^W(N) := X_0(N)/W$. Denote by $g_{X_0(N)}$ the genus of $X_0(N)$ and g_W the genus of $X_0^W(N)$. Then the following formula is well-known [2],

$$2g_{X_0(N)} - 2 = 2^r(2g_{W(N)} - 2) + \sum_{1<d|N} \nu(N,d)$$

where $\nu(N,d)$ denotes the number of fixed points of the involution w_d in $X_0(N)$.

More in general, W are given by certain Atkin-Lehner involutions, and the genus formula for $X_0^W(N)$ is given by

$$2g_{X_0(N)} - 2 = 2^s(2g_W - 2) + \sum_{w_d \in W} \nu(N, d).$$

We introduce different Magma functions: the function $fixedpointsALinvsmall(m, n)$ computes $\nu(m, n)$ when $n \in \{1, 2, 3, 4\}$, the function $fixedpointsALinvbig(m, n)$ computes $\nu(m, n)$ when $n \geq 5$, (here we use Kluit formulae in [3] to compute $\nu(m, n)$, named in the Magma source by

`nu_n{`

), the function $generexoN(m)$ computes the genus of $X_0(m)$, and finally the provide ad-hoc magma instructions for the genus of $X_0^*(m)$ where we run for $m = 4 * 255$, where as output gives the different $\nu(N, n)$ the genus of $X_0(N)$ and $X_0^*(N)$ (named "genusxoNstar"). Of course with very small modifications we could obtain the genus of $X_0(N)/W$ with W a subgroup generated by certain Atkin-Lehner involutions of $X_0(N)$, following the previous formula for computing its genus introduced in the beginning of this section.

MAGMA CODE:

```
fixedpointsALinvsmall:=function(m, n)

if n eq 3 then
        q:=Factorization(Numerator(m/3));
        nu_3:=2;
        for d in [1 .. #q] do
          if q[d][1] eq 2 then
              if q[d][2] gt 2 then
               nu_3:=0;
              else
              end if;
          else
            nu_3:=nu_3*(1+LegendreSymbol(-n,q[d][1]));
          end if;
        end for;
        return nu_3;
else
        if n eq 1 then
        else
            if n eq 2 then
              q:=PrimeDivisors(Numerator(m/n));
              nu_2factorminus1:=1;
              nu_2factorminus2:=1;
              for d in [1 .. #q] do
                 nu_2factorminus1:=nu_2factorminus1*(1+LegendreSymbol(-1,q[d]));
                 nu_2factorminus2:=nu_2factorminus2*(1+LegendreSymbol(-2,q[d]));
              end for;
              return nu_2factorminus1+nu_2factorminus2;
            else
               if n eq 4 then
```

```
                q:=PrimeDivisors(Numerator(m/n));
                l:=Divisors(Numerator(m/n));
                nu_4factorminus1:=1; nu_4sumeuler:=0;
                for d in [1..#q] do
                  nu_4factorminus1:=nu_4factorminus1*(1+LegendreSymbol(-1,q[d]));
                end for;
                for dd in [1..#l] do
                   ss:=Numerator(m/(n*l[dd]));
                   ssh:=GCD(l[dd],ss);
                   nu_4sumeuler:=nu_4sumeuler+EulerPhi(ssh);
                end for;
                return nu_4factorminus1+nu_4sumeuler;
              end if;
            end if;
          end if;
end if;
end function;



fixedpointsALinvbig:=function(m, n)

PD:=PrimeDivisors(Numerator(m/n)); D:=Divisors(Numerator(m/n));

n_mod2:=n mod 2; n_mod4:=n mod 4;


if n_mod2 eq 1 then
     if n_mod4 eq 3 then
          if 2 in PD then
              if 8 in D then
                  nu_nodd3mod4_8:=2*(ClassNumber(-n)+ClassNumber(-4*n))*(1+KroneckerSymbol(-n,2));
                    if #PD eq 1 then
                    else
                        for p1 in PD do
                         p1_mod2:=p1 mod 2;
                         if p1_mod2 eq 1 then
                           nu_nodd3mod4_8:=nu_nodd3mod4_8*(1+LegendreSymbol(-n,p1));
                         end if;
                        end for;
                    end if;
                    return nu_nodd3mod4_8;
              else
                  if 4 in D then
                     nu_nodd3mod4_4:=
                           (2*ClassNumber(-4*n)+2*(1+KroneckerSymbol(-n,2))*ClassNumber(-n));
                     if #PD eq 1 then
                       return nu_nodd3mod4_4;
                     else
                       for p2 in PD do
```

```
                    p2_mod2:=p2 mod 2;
                      if p2_mod2 eq 1 then
                       nu_nodd3mod4_4:=nu_nodd3mod4_4*(1+LegendreSymbol(-n,p2));
                      end if;
                    end for;
                    return nu_nodd3mod4_4;
                  end if;
               else
                    nu_nodd3mod4_2:=ClassNumber(-4*n)+3*ClassNumber(-n);
                    if #PD eq 1 then
                       return nu_nodd3mod4_2;
                    else
                       for p3 in PD do
                          p3_mod2:=p3 mod 2;
                          if p3_mod2 eq 1 then
                            nu_nodd3mod4_2:=nu_nodd3mod4_2*(1+LegendreSymbol(-n,p3));
                          end if;
                        end for;
                        return nu_nodd3mod4_2;
                    end if;
               end if;
          end if;
     else
          if #PD eq 0 then
            nu_nodd3mod4_no2:=ClassNumber(-n)+ClassNumber(-4*n);
            return nu_nodd3mod4_no2;
          else
            nu_nodd3mod4_no2:=ClassNumber(-n)+ClassNumber(-4*n);
             for j in [1..#PD] do
               nu_nodd3mod4_no2:=nu_nodd3mod4_no2*(1+LegendreSymbol(-n,PD[j]));
             end for;
             return nu_nodd3mod4_no2;
          end if;
     end if;
else
     if #PD eq 0 then
        nu_nodd1mod4:=ClassNumber(-4*n);
        return nu_nodd1mod4;
     else
        if 2 in PD then
             if 4 in D then
               return 0;
             else
                nu_nodd1mod4_2:=ClassNumber(-4*n);
                PD2:=PrimeDivisors(Numerator(m/(2*n)));
                 for k in [1..#PD2] do
                  nu_nodd1mod4_2:=nu_nodd1mod4*(1+LegendreSymbol(-n,PD2[k]));
                 end for;
              return nu_nodd1mod4_2;
```

```
                end if;
            else
                nu_nodd1mod4_no2:=ClassNumber(-4*n);
                  for i in [1 .. #PD] do
                   nu_nodd1mod4_no2:=nu_nodd1mod4_no2*(1+LegendreSymbol(-n,PD[i]));
                  end for;
                return nu_nodd1mod4_no2;
            end if;
          end if;
     end if;
 else
    nu_neven:=ClassNumber(-4*n);
    if #PD eq 0 then
      return nu_neven;
    else
      for u in [1 .. #PD] do
        nu_neven:=nu_neven*(1+LegendreSymbol(-n,PD[u]));
      end for;
      return nu_neven;
    end if;
 end if;
end function;



generexoN:=function(b)

  m:=PrimeDivisors(b);
  l:=Divisors(b);
  factor_b:=Factorization(b);
 psiEulerindex:=b;
 order4elliptic:=1;
 order3elliptic:=1;
 cusps:=0;

 for x in [1 .. #m] do
 psiEulerindex:=psiEulerindex*(1+1/m[x]);
 end for;

   for y in [1..#m] do
     if factor_b[y][1] eq 2 then
         order3elliptic:=0;
        if factor_b[y][2] gt 1 then
          order4elliptic:=0;
        end if;
     else
        if factor_b[y][1] eq 3 then
            if factor_b[y][2] gt 1 then
              order3elliptic:=0;
              order4elliptic:=order4elliptic*(1+LegendreSymbol(-1,factor_b[y][1]));
```

```
            else
              order3elliptic:=order3elliptic*(1+LegendreSymbol(-3,factor_b[y][1]));
              order4elliptic:=order4elliptic*(1+LegendreSymbol(-1,factor_b[y][1]));
            end if;
         else
            order4elliptic:=order4elliptic*(1+LegendreSymbol(-1,factor_b[y][1]));
            order3elliptic:=order3elliptic*(1+LegendreSymbol(-3,factor_b[y][1]));
         end if;
      end if;
    end for;

 for a in [1 .. #l] do
   n1:=Numerator(b/l[a]);
   t1:=GCD(l[a],n1);
   cusps:=cusps+EulerPhi(t1);
  end for;

genus:=1+(psiEulerindex/12)-(order4elliptic/4)-(order3elliptic/3)-(cusps/2);
return genus;
 end function;
```

The ad-hoc subroutine for computing the genus of $X_0^*(N)$ with $N = 4 * 255$

```
N:=4*255;
FixedpointsALinvolutions:=[* *];
Dd:=Divisors(N);

for i in [1..#Dd] do
   u:=GCD(Dd[i],Numerator(N/Dd[i]));
   if u eq 1 then
      if Dd[i] eq 1 then
      else
         if Dd[i] gt 4 then
            nu_Ddi:=fixedpointsALinvbig(N,Dd[i]);
            FixedpointsALinvolutions:=Append(FixedpointsALinvolutions,nu_Ddi);
         else
            nu_Ddi:=fixedpointsALinvsmall(N,Dd[i]);
            FixedpointsALinvolutions:=Append(FixedpointsALinvolutions,nu_Ddi);
         end if;
      end if;
    end if;
end for;

CountAllFixedPointsALinvolutions:=0;

for u in FixedpointsALinvolutions do
  CountAllFixedPointsALinvolutions:=CountAllFixedPointsALinvolutions+u;
end for;
```

```
genusxoN:=generexoN(N);
genusxoNstar:=1+2^(-#PrimeDivisors(N))*(genusxoN-1-(CountAllFixedPointsALinvolutions/2));
genusxoNstar;
```

Acknowledgements. We thank referees for their comments, especially those that have contributed to improve the presentation of the Magma source presented here.

References

[1] J. González. Constraints on the automorphism group of a curve. *J. Théor. Nombres Bordeaux*, 29(2):535–548, 2017.
[2] J. González and J.-C. Lario. Rational and elliptic parametrizations of **Q**-curves. *J. Number Theory*, 72(1):13–31, 1998.
[3] P. G. Kluit. On the normalizer of $\Gamma_0(N)$. pages 239–246. Lecture Notes in Math., Vol. 601, 1977.