The AGM-$X_0(N)$ Algorithm

David R. Kohel

The AGM-$X_0(N)$ Algorithm

The AGM-$X_0(N)$ Algorithm

David R. Kohel

§1 Introduction

This document describes the AGM-$X_0(N)$ elliptic curve point counting algorithm, generalizing the algorithms of Satoh and Mestre. The algorithm computes a p-adic lift of the modular invariants of an elliptic curves $E/\mathbf{F}_{p^m}$ and determines the action of Frobenius on the 1-dimensional space of differentials. In fact the action of Frobenius is determined on a generically parametrized curve, and the sign is determined subsequently on the original curve. At present only the top-level intrinsic `AGMTrace` is described, but the individual components of this algorithm appear in the appendix.

§2 Trace of Frobenius

The intrinsic `AGMTrace` returns the trace of Frobenius as determined by its action on a p-adic lift of the elliptic curve.

`AGMTrace(E)`

`AGMTrace(E,N)`

Given an ordinary elliptic curve over a field of characteristic p in the set $\{2, 3, 5, 7, 13\}$ returns the trace of Frobenius on E determined by an AGM-$X_0(N)$ recursion to determine the canonical p-adically lift of the invariants of E on $X_0(N)$.

§3 Examples of trace computations

The following example produces the timing data reported in Kohel [Koh03].

Example E1

First we set up a procedure to print the timings which we will use for several runs of the AGM-$X_0(N)$ algorithm.

```
> procedure AGMTimings(p,m,N,num)
>     tyme := Cputime();
>     prec := ((m+1) div 2)+1+(4 div p);
>     R := pAdicQuotientRing(p,prec);
>     S := CyclotomicUnramifiedExtension(R,m);
>     print "One-time setup:", Cputime(tyme);
>     FF := GF(p,m);
```

```
>     k := 0;
>     tyme := Cputime();
>     while k lt num do
>         j := Random(FF);
>         if j notin {FF|0,1} then
>             E := EllipticCurveWithjInvariant(j);
>             t := AGMTrace(E,N : ExtensionRing := S);
>             k +:= 1;
>         end if;
>     end while;
>     printf "Average time: %o secs\n", Cputime(tyme)/num;
> end procedure;
```

Then we can determine the average algorithmic performance for this algorithm over various fields of small characteristic. Over the field $\mathbf{F}_{2^{163}}$ one obtains the following data.

```
> p := 2; m := 163;
> for N in [2,4,8,16] do
>     print "N =", N;
>     AGMTimings(p,m,N,16);
> end for;
N = 2
One-time setup: 0.059
Average time: 0.4869375 secs
N = 4
One-time setup: 0.059
Average time: 0.4813125 secs
N = 8
One-time setup: 0.059
Average time: 0.4650625 secs
N = 16
One-time setup: 0.059
Average time: 0.5663125 secs
```

For the middle range of elliptic curves for cryptography, one might take the field $\mathbf{F}_{2^{193}}$.

```
> p := 2; m := 193;
> for N in [2,4,8,16] do
>     print "N =", N;
>     AGMTimings(p,m,N,16);
> end for;
N = 2
One-time setup: 0.079
Average time: 0.615625 secs
N = 4
One-time setup: 0.081
Average time: 0.61625 secs
N = 8
One-time setup: 0.079
Average time: 0.62375 secs
```

```
N = 16
One-time setup: 0.081
Average time: 0.7456875 secs
```

Finally, for higher security (or paranoia) a base field $\mathbf{F}_{2^{239}}$ might be employed. One finds that determining the number of points on such curves requires, on average, less than a second using $X_0(2)$, $X_0(4)$, or $X_0(8)$.

```
> p := 2; m := 239;
> for N in [2,4,8,16] do
>     print "N =", N;
>     AGMTimings(p,m,N,16);
> end for;
N = 2
One-time setup: 0.129
Average time: 0.9143125 secs
N = 4
One-time setup: 0.129
Average time: 0.9268125 secs
N = 8
One-time setup: 0.141
Average time: 0.9411875 secs
N = 16
One-time setup: 0.13
Average time: 1.126875 secs
```

The timing data for other small characteristics can be determined similarly.

---

§4 Bibliography

[**Koh03**] David R. Kohel. The AGM-$X_0(N)$ Heegner point lifting algorithm and elliptic curve point counting. In *Proceedings of Asiacrypt 2003*, Berlin, 2003. Springer.

A1 AGM Package

agm_heegner.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                            David Kohel                             //
//                AGM-X0(N) Heegner Class Polynomials                 //
//                                                                    //
////////////////////////////////////////////////////////////////////////
declare verbose Heegner, 3;
import "frobenius_representation.m" : AGMFrobeniusNorm;
function BalancedMod(t,m)
    t mod:= m;
    if t gt ((m+1) div 2) then
        t -:= m;
    end if;
    return t;
end function;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function MyRoots(f)
    R := BaseRing(Parent(f));
    F := ResidueClassField(R);
    f0 := PolynomialRing(F)!f;
    return [ HenselLift(f,R!r0[1]) : r0 in Roots(f0) ];
end function;
function PhiX0(N,p,X)
    P := PolynomialRing(Parent(X));
    Y := P.1;
    case N:
    when 16:
        case p:
        when 3:
            return X^4 - 16*X^3*Y^3 + 6*X^2*Y^2 - X*Y + Y^4;
        when 5:
            return X^6 - 256*X^5*Y^5 + 10*X^5*Y + 15*X^4*Y^2
                - 20*X^3*Y^3 + 15*X^2*Y^4 + 10*X*Y^5 - X*Y + Y^6;
        when 7:
            return X^8 - 4096*X^7*Y^7 + 224*X^7*Y^3 - 1792*X^6*Y^6
                + 140*X^6*Y^2 - 448*X^5*Y^5 + 14*X^5*Y + 70*X^4*Y^4
                + 224*X^3*Y^7 - 28*X^3*Y^3 + 140*X^2*Y^6 - 7*X^2*Y^2
                + 14*X*Y^5 - X*Y + Y^8;
        when 11:
```

```
    return
        X^12 - 1048576*X^11*Y^11 + 90112*X^11*Y^7 - 1584*X^11*Y^3
        - 1441792*X^10*Y^10 + 70400*X^10*Y^6 + 1298*X^10*Y^2
        - 1036288*X^9*Y^9 + 65472*X^9*Y^5 - 99*X^9*Y
        - 298496*X^8*Y^8 + 19151*X^8*Y^4 + 90112*X^7*Y^11
        - 74272*X^7*Y^7 + 4092*X^7*Y^3 + 70400*X^6*Y^10
        - 7876*X^6*Y^6 + 275*X^6*Y^2 + 65472*X^5*Y^9
        - 4642*X^5*Y^5 + 22*X^5*Y + 19151*X^4*Y^8
        - 1166*X^4*Y^4 - 1584*X^3*Y^11 + 4092*X^3*Y^7
        - 253*X^3*Y^3 + 1298*X^2*Y^10 + 275*X^2*Y^6
        - 22*X^2*Y^2 - 99*X*Y^9 + 22*X*Y^5 - X*Y + Y^12;
when 17:
    return X^18 - 4294967296*X^17*Y^17 + 570425344*X^17*Y^13
        - 23396352*X^17*Y^9 + 287232*X^17*Y^5 - 306*X^17*Y
        + 3422552064*X^16*Y^14 - 529203200*X^16*Y^10
        + 19253248*X^16*Y^6 + 28441*X^16*Y^2
        + 10267656192*X^15*Y^15 - 2560229376*X^15*Y^11
        + 132596736*X^15*Y^7 - 793968*X^15*Y^3
        + 3422552064*X^14*Y^16 - 3873767424*X^14*Y^12
        + 194865152*X^14*Y^8 + 2120308*X^14*Y^4
        + 570425344*X^13*Y^17 - 5421268992*X^13*Y^13
        + 139394560*X^13*Y^9 + 12298888*X^13*Y^5
        + 1122*X^13*Y - 3873767424*X^12*Y^14
        - 312160256*X^12*Y^10 + 33457156*X^12*Y^6
        + 75208*X^12*Y^2 - 2560229376*X^11*Y^15
        - 419776512*X^11*Y^11 + 27917808*X^11*Y^7
        + 517956*X^11*Y^3 - 529203200*X^10*Y^16
        - 312160256*X^10*Y^12 + 9441902*X^10*Y^8
        + 761192*X^10*Y^4 - 23396352*X^9*Y^17
        + 139394560*X^9*Y^13 - 17290156*X^9*Y^9
        + 544510*X^9*Y^5 - 357*X^9*Y + 194865152*X^8*Y^14
        + 9441902*X^8*Y^10 - 1219376*X^8*Y^6 -
        8075*X^8*Y^2 + 132596736*X^7*Y^15 +
        27917808*X^7*Y^11 - 1639752*X^7*Y^7 -
        39066*X^7*Y^3 +
        19253248*X^6*Y^16 +
        33457156*X^6*Y^12 - 1219376*X^6*Y^8 -
        59109*X^6*Y^4 + 287232*X^5*Y^17 + 12298888*X^5*Y^13 +
        544510*X^5*Y^9 -
        82722*X^5*Y^5 + 34*X^5*Y + 2120308*X^4*Y^14 +
        761192*X^4*Y^10 - 59109*X^4*Y^6 +
        204*X^4*Y^2 - 793968*X^3*Y^15 + 517956*X^3*Y^11
        - 39066*X^3*Y^7 + 612*X^3*Y^3 +
        28441*X^2*Y^16 + 75208*X^2*Y^12 -
```

```
                8075*X^2*Y^8 + 204*X^2*Y^4 - 306*X*Y^17
                + 1122*X*Y^13 - 357*X*Y^9 + 34*X*Y^5 - X*Y + Y^18;
        end case;
    end case;
    assert false;
end function;
function ExtendPrimeGenerators(Q,p)
    D := Discriminant(Q);
    h := ClassNumber(Q);
    m := Conductor(Q);
    pp := PrimeForm(Q,p);
    if Order(pp) eq h then return [], []; end if;
    prms := [];
    ords := [];
    Sfrms := {@ pp^i : i in [0..Order(pp)-1] @};
    Qfrms := Sfrms;
    while #Qfrms lt h do
        p := NextPrime(p);
        while KroneckerSymbol(D,p) eq -1 or m mod p eq 0 do
            p := NextPrime(p);
        end while;
        qq := Q!1;
        pp := PrimeForm(Q,p);
        for i in [1..Order(pp)] do
            qq *:= pp;
            if qq notin Qfrms then
                assert i ne Order(pp);
                Qfrms join:= {@ ff*qq : ff in Sfrms @};
            else
                if i ne 1 then
                    Append(~prms,p);
                    Append(~ords,i);
                end if;
                break i;
            end if;
        end for;
        Sfrms := Qfrms;
    end while;
    return prms, ords;
end function;


intrinsic HeegnerClassPolynomial(N::RngIntElt,x::FldFinElt
    : DefaultPrecision := 100) -> RngUPolElt
```

```
{}
require x ne 0 : "Argument 2 must be nonzero.";
FF := Parent(x);
n := Degree(FF);
p := Characteristic(FF);
prec := DefaultPrecision;
v1 := AGMLift(N,x,prec);
Nrm := AGMFrobeniusNorm(N,p,v1);
t := BalancedMod(Integers()!Nrm,p^(n-1));
D := t^2-4*#FF;
vprint Heegner, 2 : "D =", D;
Q := QuadraticForms(t^2-4*#FF);
prms, ords := ExtendPrimeGenerators(Q,p);
vprint Heegner, 3 : "prms =", prms;
vprint Heegner, 3 : "ords =", ords;
Gelts := [v1];
Relts := {@ x^p^k : k in [0..n-1] @};
// "Relts =", Relts;
for i in [1..#prms] do
    q := prms[i];
    vprint Heegner, 3 : "q =", q;
    vprint Heegner, 3 : "o =", ords[i];
    for j in [1..(ords[i] div 2)] do
        Celts := [];
        for v1 in Gelts do
            for u1 in MyRoots(PhiX0(N,q,v1)) do
                u0 := FF!u1;
                if u0 notin Relts then
                    // "New u0 =", u0;
                    Append(~Celts,u1);
                    Relts join:= {@ u0^p^k : k in [0..n-1] @};
                    // "Relts =", Relts;
                else
                end if;
            end for;
        end for;
        Gelts cat:= Celts;
    end for;
end for;
Hrts := [];
for v1 in Gelts do
    assert N eq 16;
    v2 := FrobeniusImage(v1);
    x1 := v2;
```

```
        y1 := v1*(4*v2^2+1);
        // x2 is twice the A-L image:
        x2 := (-2*v1+1) div (2*v1+1);
        y2 := (-2*v1+1)*(2*v2+1) div ((2*v1+1)*(-2*v2+1));
        // zz := (y2+y1) div (2*x1+x2);
        zz := (y2+y1-2) div (2*x1+x2-2);
        // y1+y2 = 2*ww^3 - 3*ww + 2;
        // 2*x1+x2 = 2*ww^2 - 1;
        // y1^2 - (2*zz^3-3*zz+2)*y1 - (2*zz^3-4*zz^2+3*zz-1);
        if GetVerbose("Heegner") ge 3 then
            print "Valuations:";
            // Verify that all give zero:
            Valuation((4*v2^2+1)*v1^2-v2);
            Valuation((4*x1^2+1)*x1-y1^2);
            Valuation((x2^2+1)*x2-2*y2^2);
        end if;
        Append(~Hrts,zz);
    end for;
    S := Eltseq(&*[ MinimalPolynomial(zz) : zz in Hrts ]);
    R := ChangePrecision(U,Precision(U)-32) where U := Universe(S);
    f := PolynomialRing(Rationals())!
        RationalReconstruction([R!c : c in S]);
    return f, D, Hrts;
end intrinsic;


intrinsic HeegnerDivisor(N::RngIntElt,x::FldFinElt : Precision := 100)
    -> RngUPolElt
    {}
    f, D := HeegnerClassPolynomial(N,x : DefaultPrecision := Precision);
    H := NumberField(f); z := H.1;
    w := Sqrt(D*(4*z^4-12*z^2+16*z-7));
    // return f, Parent(f)!Eltseq(w), D;
    // K<t> := QuadraticField(D);
    P<x> := PolynomialRing(Rationals());
    // E := HyperellipticCurve(4*x^3+x);
    C := HyperellipticCurve(D*(4*x^4-12*x^2+16*x-7));
    J := Jacobian(C);
    // f := PolynomialRing(K)!f;
    return J![f,Parent(f)!Eltseq(w)];
end intrinsic;
/*
A2<x,y> := AffineSpace(QQ,2);
E := Curve(A2,y^2-4*x^3-x);
```

```
C := Curve(A2,y^2-(4*x^4-12*x^2+16*x-7)/4);
z := (-y+1)/(-2*x+1);
w := (z^2-3/2+1/z)-y/z;
m := map< E->C | [z,w] >;
m0 := Extend(ProjectiveClosure(m));
_, n0 := IsInvertible(m0);
P2<X,Y,Z> := Ambient(Domain(m0));
E0 := Domain(m0);
C0 := Codomain(m0);
i := Extend(map< E0->E0 |
    [(Y-2*X)/(Y+2*X)/2,(X*Z^2-4*X^3)/(Z*(4*X^2+4*X*Y+Y^2)),1] >);
*/
```

agm_lift.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                            David Kohel                             //
//                     AGM-X0(N) Lifting Algorithms                   //
//                                                                    //
////////////////////////////////////////////////////////////////////////
import "analytic_frobenius_lift.m" : AnalyticAGMLift;
import "modular_correspondences.m" : PhiX0, DxPhiX0, DyPhiX0;
forward AGMLift, LinearAGMLift, BlockAGMLift;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function MagmaWordSize(p)
    // Returns n such that p^n < 2^30.
    case p:
    when 2: return 29;
    when 3: return 18;
    when 5: return 12;
    when 7: return 10;
    when 13: return 8;
    end case;
end function;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


intrinsic AGMLift(N::RngIntElt,x::FldFinElt,prec::RngIntElt :
    Cyclotomic := true,
    ExtensionRing := false) -> RngPadResExtElt
    {}
    F := Parent(x);
    n := Degree(F);
    p := Characteristic(F);
    require p in {2,3,5,7,13} : // and n gt 1 :
        "Argument must be defined over a nonprime " *
        "finite field of characteristic 2, 3, 5, 7, or 13.";
    require N eq 1 or N mod p eq 0 :
        "Argument 1 must be zero modulo the characteristic.";
    require x ne 0 :
        "Argument 2 must be nonzero.";
    word := Min(MagmaWordSize(p),prec);
    tyme := Cputime();
    if Type(ExtensionRing) eq BoolElt then
```

```
        Z := pAdicQuotientRing(p,prec);
        if Cyclotomic then
            S := CyclotomicUnramifiedExtension(Z,n);
        else
            S := UnramifiedExtension(Z,n);
        end if;
    else
        S := ChangePrecision(ExtensionRing,prec);
    end if;
    vprint AGM, 2: "Cyclotomic setup  ", Cputime(tyme);
    // Do an analytic list using initial segments of the
    // analytic Frobenius; can add a precision argument.
    if N gt 1 then
        analt := word;
        R := ChangePrecision(S,analt);
        tyme := Cputime();
        x := AnalyticAGMLift(N,R!x);
    else
        analt := 0;
    end if;
    vprint AGM, 2: "Analytic lift:    ", Cputime(tyme);
    // Currently the linear lifting phase is done with
    // the precomputed analytic Frobenius.
    if analt lt word then
        R := ChangePrecision(S,word);
        tyme := Cputime();
        x := LinearAGMLift(N,R!x);
        vprint AGM, 2: "Linear lift:      ", Cputime(tyme);
    end if;
    // Once the initial lift is complete to one word block,
    // lift using a blockwise algorithm.
    tyme := Cputime();
    x := BlockAGMLift(N,S,x);
    vprint AGM, 2: "Total block lift: ", Cputime(tyme);
    return x;
end intrinsic;
function LinearAGMLift(N,x)
    // Lift x linearly to a root of the modular polynomial,
    // up to the precision of the residue class ring.
    S := Parent(x);
    p := Prime(S);
    FF := ResidueClassField(S);
    j := 1;
    repeat
```

```
        Delta := PhiX0(N,p,x,FrobeniusImage(x));
        j := Valuation(Delta);
        if j eq 1 then
            vprint AGM, 2: "Initial valuation:", j;
        end if;
        x +:= p^j*S!Root(FF!(Delta div p^j),p);
    until Delta eq 0;
    vprint AGM, 2: "Final valuation:  ", j;
    return x;
end function;
function BlockAGMLift(N,S,x);
    // S is a ring of the desired final precision, and R = Parent(x)
    // is a ring whose precision is truncated to optimize operations
    // on computer words.
    R := Parent(x);
    p := Prime(S);
    n := Degree(S);
    prec := Precision(S);
    word := Precision(R);
    FF := ResidueClassField(S);
    tyme := Cputime();
    y := FrobeniusImage(x);
    DX := DxPhiX0(N,p,x,y);
    DY := DyPhiX0(N,p,x,y);
    m := 1;
    q := p^word;
    prec0 := word;
    vprint AGM, 2: "Block lift init:  ", Cputime(tyme);
    while true do
        Rm := ChangePrecision(S,Min(prec,(m+1)*word));
        x := Rm!x;
        ntyme := Cputime();
        vprintf AGM, 2: "  Eval at prec %o: ", (m+1)*word;
        Rx := R!(PhiX0(N,p,x,FrobeniusImage(x)) div q);
        vprint AGM, 2: Cputime(ntyme);
        for i in [0..word-1] do
            if m*word+i eq prec then return x; end if;
            dx := R!Root(FF!Rx,p);
            Rx +:= DX*dx + DY*FrobeniusImage(dx);
            Rx div:= p;
            x +:= q*(Rm!dx);
            q *:= p;
        end for;
        vprint AGM, 2: "Block lift step:  ", Cputime(tyme);
```

```
        m +:= 1;
    end while;
    return x;
end function;
```

agm_frobenius_trace.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                           David Kohel                              //
//                    AGM-X0(N) Trace Computation                     //
//                                                                    //
////////////////////////////////////////////////////////////////////////
declare verbose AGM, 3;
import "frobenius_representation.m" : AGMFrobeniusNorm;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function VerifyTraceCharacter(E,t)
    // Replace this with an analysis of the character
    // of Frobenius on some the torsion module.
    P := Random(E);
    q := #BaseRing(E);
    if (q+1-t)*P ne E!0 then
        assert (q+1+t)*P eq E!0;
        return false;
    end if;
    return true;
end function;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function AGMInitialValues(j,p,n)
    prec := ((n+1) div 2)+1+(4 div p);
    case p:
    when 2: x := 1/j;
    when 3: x := 1/j;
    when 5: x := 1/j;
    when 7: x := 1/(j+1);
    when 13: x := 1/(j-5);
    end case;
    return x, prec;
end function;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


intrinsic AGMTrace(E::CrvEll : Cyclotomic := true) -> RngIntElt
    {}
    K := BaseField(E);
    p := Characteristic(K);
```

```
    n := Degree(K);
    require p in {2,3,5,7,13} and n gt 1 :
        "Argument must be defined over a nonprime " *
        "finite field of characteristic 2, 3, 5, 7, or 13.";
    return AGMTrace(E,p);
end intrinsic;


intrinsic AGMTrace(E::CrvEll,N::RngIntElt :
    Cyclotomic := true,
    ExtensionRing := false) -> RngIntElt
    {}
    K := BaseField(E);
    p := Characteristic(K);
    n := Degree(K);
    require p in {1,2,3,5,7,13} and n gt 1 :
        "Argument must be defined over a nonprime " *
        "finite field of characteristic 2, 3, 5, 7, or 13.";
    require N eq 1 or N mod p eq 0 :
        "Argument 2 must be zero modulo the characteristic.";
    j := jInvariant(E);
    if N gt 1 then
        x, prec := AGMInitialValues(j,p,n);
    else
        x := j; prec := ((n+1) div 2)+1+(4 div p);
    end if;
    vprint AGM, 2: "Final precision:  ", prec;
    run_tyme := Cputime();
    x := AGMLift(N,x,prec :
        Cyclotomic := Cyclotomic,
        ExtensionRing := ExtensionRing);
    vprint AGM, 1: "Lifting total:    ", Cputime(run_tyme);
    nrm_tyme := Cputime();
    Nrm := AGMFrobeniusNorm(N,p,x);
    vprint AGM, 1: "Exp(Trace(Log())):", Cputime(nrm_tyme);
    if p eq 2 then prec -:= 1; end if;
    m := p^prec;
    t := (Integers()!Nrm) mod m;
    if t gt m-t then t -:= m; end if;
    vprint AGM, 1: "Total trace time: ",Cputime(run_tyme);
    vprint AGM, 1: "Frobenius trace:  ", t;
    bool := VerifyTraceCharacter(E,t);
    return bool select t else -t;
end intrinsic;
```

modular_correspondences.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                          David Kohel                               //
//              Modular Correspondences for the AGM                   //
//                                                                    //
////////////////////////////////////////////////////////////////////////
forward PhiX0;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


intrinsic AGMModularCorrespondence(N::RngIntElt,p::RngIntElt)
    -> RngMPolElt
    {The modular correspondencee on X_0(N) use for AGM recursion.}
    P := PolynomialRing(Integers(),2 : Global := true);
    return PhiX0(N,p,P.1,P.2);
end intrinsic;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function PhiX0(N,p,x,y)
    if N eq 1 then
        return Evaluate(ClassicalModularPolynomial(p),[x,y]);
    end if;
    case [N,p]:
    when [2,2]:
        return x^2-(4096*y+48)*x*y-y;
    when [4,2]:
        return x^2-(16*(16*x*y+x+y)+1)*y;
    when [8,2]:
        return x^2*(4*y+1)^2-y;
    when [16,2]:
        return x^2*(4*y^2+1)-y;
    when [3,3]:
        xy := x*y;
        return x^3 - 531441*xy^2*y - 26244*xy^2
            - 270*x*xy - 729*xy*y - 36*xy - y;
    when [9,3]:
        return x^3 - 729*x^2*y^3 - 243*x^2*y^2 - 27*x^2*y
            - 243*x*y^3 - 81*x*y^2 - 9*x*y - 27*y^3 - 9*y^2 - y;
    when [5,5]:
        return x^5 - 244140625*x^4*y^5 - 58593750*x^4*y^4
            - 4921875*x^4*y^3 - 162500*x^4*y^2 - 1575*x^4*y
```

```
        - 1953125*x^3*y^4 - 468750*x^3*y^3 - 39375*x^3*y^2
        - 1300*x^3*y - 15625*x^2*y^3 - 3750*x^2*y^2
        - 315*x^2*y - 125*x*y^2 - 30*x*y - y;
when [25,5]:
    return x^5 - 625*x^4*y^5 - 625*x^4*y^4 - 375*x^4*y^3
        - 125*x^4*y^2 - 25*x^4*y - 625*x^3*y^5 - 625*x^3*y^4
        - 375*x^3*y^3 - 125*x^3*y^2 - 25*x^3*y - 375*x^2*y^5
        - 375*x^2*y^4 - 225*x^2*y^3 - 75*x^2*y^2 - 15*x^2*y
        - 125*x*y^5 - 125*x*y^4 - 75*x*y^3 - 25*x*y^2 - 5*x*y
        - 25*y^5 - 25*y^4 - 15*y^3 - 5*y^2 - y;
when [7,7]:
    return x^7 - 13841287201*x^6*y^7 - 7909306972*x^6*y^6
        - 1856265922*x^6*y^5 - 224003696*x^6*y^4 - 14201915*x^6*y^3
        - 422576*x^6*y^2 - 4018*x^6*y - 282475249*x^5*y^6
        - 161414428*x^5*y^5 - 37882978*x^5*y^4 - 4571504*x^5*y^3
        - 289835*x^5*y^2 - 8624*x^5*y - 5764801*x^4*y^5
        - 3294172*x^4*y^4 - 773122*x^4*y^3 - 93296*x^4*y^2
        - 5915*x^4*y - 117649*x^3*y^4 - 67228*x^3*y^3
        - 15778*x^3*y^2 - 1904*x^3*y - 2401*x^2*y^3
        - 1372*x^2*y^2 - 322*x^2*y - 49*x*y^2
        - 28*x*y - y;
when [13,13]:
    return x^13 - 23298085122481*x^12*y^13
        - 46596170244962*x^12*y^12 - 44804009850925*x^12*y^11
        - 27020264402404*x^12*y^10 - 11283187332872*x^12*y^9
        - 3409754413780*x^12*y^8 - 758378576462*x^12*y^7
        - 123855918940*x^12*y^6 - 14548002326*x^12*y^5
        - 1174999540*x^12*y^4 - 59916584*x^12*y^3
        - 1623076*x^12*y^2 - 15145*x^12*y
        - 1792160394037*x^11*y^12 - 3584320788074*x^11*y^11
        - 3446462296225*x^11*y^10 - 2078481877108*x^11*y^9
        - 867937487144*x^11*y^8 - 262288801060*x^11*y^7
        - 58336813574*x^11*y^6 - 9527378380*x^11*y^5
        - 1119077102*x^11*y^4 - 90384580*x^11*y^3
        - 4608968*x^11*y^2 - 124852*x^11*y
        - 137858491849*x^10*y^11 - 275716983698*x^10*y^10
        - 265112484325*x^10*y^9 - 159883221316*x^10*y^8
        - 66764422088*x^10*y^7 - 20176061620*x^10*y^6
        - 4487447198*x^10*y^5 - 732875260*x^10*y^4
        - 86082854*x^10*y^3 - 6952660*x^10*y^2 - 354536*x^10*y
        - 10604499373*x^9*y^10 - 21208998746*x^9*y^9
        - 20393268025*x^9*y^8 - 12298709332*x^9*y^7
        - 5135724776*x^9*y^6 - 1552004740*x^9*y^5
        - 345188246*x^9*y^4 - 56375020*x^9*y^3
```

```
            - 6621758*x^9*y^2 - 534820*x^9*y - 815730721*x^8*y^9
            - 1631461442*x^8*y^8 - 1568712925*x^8*y^7
            - 946054564*x^8*y^6 - 395055752*x^8*y^5
            - 119384980*x^8*y^4 - 26552942*x^8*y^3 - 4336540*x^8*y^2
            - 509366*x^8*y - 62748517*x^7*y^8 - 125497034*x^7*y^7
            - 120670225*x^7*y^6 - 72773428*x^7*y^5 - 30388904*x^7*y^4
            - 9183460*x^7*y^3 - 2042534*x^7*y^2 - 333580*x^7*y
            - 4826809*x^6*y^7 - 9653618*x^6*y^6 - 9282325*x^6*y^5
            - 5597956*x^6*y^4 - 2337608*x^6*y^3 - 706420*x^6*y^2
            - 157118*x^6*y - 371293*x^5*y^6 - 742586*x^5*y^5
            - 714025*x^5*y^4 - 430612*x^5*y^3 - 179816*x^5*y^2
            - 54340*x^5*y - 28561*x^4*y^5 - 57122*x^4*y^4
            - 54925*x^4*y^3 - 33124*x^4*y^2 - 13832*x^4*y
            - 2197*x^3*y^4 - 4394*x^3*y^3 - 4225*x^3*y^2
            - 2548*x^3*y - 169*x^2*y^3 - 338*x^2*y^2
            - 325*x^2*y - 13*x*y^2 - 26*x*y - y;
    end case;
    error "N must be in {2,3,4,5,7,8,9,13,16,25} not " * Sprint(N);
end function;
function DxPhiX0(N,p,x,y)
    if N eq 1 then
Phi := ClassicalModularPolynomial(p);
return Evaluate(Derivative(Phi,1),[x,y]);
    end if;
    case [N,p]:
    when [2,2]:
        return 2*(x-(2048*y+24)*y);
    when [4,2]:
        return 2*(x-(128*y+8)*y);
    when [8,2]:
        return 2*x*(4*y+1)^2;
    when [16,2]:
        return 2*x*(4*y^2+1);
    when [3,3]:
        return 3*x^2 - 1062882*x*y^3 - 52488*x*y^2
            - 540*x*y - 729*y^2 - 36*y;
    when [9,3]:
        return 3*x^2 - 1458*x*y^3 - 486*x*y^2 - 54*x*y
            - 243*y^3 - 81*y^2 - 9*y;
    when [5,5]:
        return 5*x^4 - 976562500*x^3*y^5 - 234375000*x^3*y^4
            - 19687500*x^3*y^3 - 650000*x^3*y^2 - 6300*x^3*y
            - 5859375*x^2*y^4 - 1406250*x^2*y^3 - 118125*x^2*y^2
            - 3900*x^2*y - 31250*x*y^3 - 7500*x*y^2 - 630*x*y
```

```
        - 125*y^2 - 30*y;
when [25,5]:
    return 5*x^4 - 2500*x^3*y^5 - 2500*x^3*y^4 - 1500*x^3*y^3
        - 500*x^3*y^2 - 100*x^3*y - 1875*x^2*y^5 - 1875*x^2*y^4
        - 1125*x^2*y^3 - 375*x^2*y^2 - 75*x^2*y - 750*x*y^5
        - 750*x*y^4 - 450*x*y^3 - 150*x*y^2 - 30*x*y - 125*y^5
        - 125*y^4 - 75*y^3 - 25*y^2 - 5*y;
when [7,7]:
    return 7*x^6 - 83047723206*x^5*y^7 - 47455841832*x^5*y^6
        - 11137595532*x^5*y^5 - 1344022176*x^5*y^4
        - 85211490*x^5*y^3 - 2535456*x^5*y^2 - 24108*x^5*y
        - 1412376245*x^4*y^6 - 807072140*x^4*y^5 - 189414890*x^4*y^4
        - 22857520*x^4*y^3 - 1449175*x^4*y^2 - 43120*x^4*y
        - 23059204*x^3*y^5 - 13176688*x^3*y^4 - 3092488*x^3*y^3
        - 373184*x^3*y^2 - 23660*x^3*y - 352947*x^2*y^4
        - 201684*x^2*y^3 - 47334*x^2*y^2 - 5712*x^2*y
        - 4802*x*y^3 - 2744*x*y^2 - 644*x*y - 49*y^2 - 28*y;
when [13,13]:
    return 13*x^12 - 279577021469772*x^11*y^13
        - 559154042939544*x^11*y^12 - 537648118211100*x^11*y^11
        - 324243172828848*x^11*y^10 - 135398247994464*x^11*y^9
        - 40917052965360*x^11*y^8 - 9100542917544*x^11*y^7
        - 1486271027280*x^11*y^6 - 174576027912*x^11*y^5
        - 14099994480*x^11*y^4 - 718999008*x^11*y^3
        - 19476912*x^11*y^2 - 181740*x^11*y
        - 19713764334407*x^10*y^12 - 39427528668814*x^10*y^11
        - 37911085258475*x^10*y^10 - 22863300648188*x^10*y^9
        - 9547312358584*x^10*y^8 - 2885176811660*x^10*y^7
        - 641704949314*x^10*y^6 - 104801162180*x^10*y^5
        - 12309848122*x^10*y^4 - 994230380*x^10*y^3
        - 50698648*x^10*y^2 - 1373372*x^10*y
        - 1378584918490*x^9*y^11 - 2757169836980*x^9*y^10
        - 2651124843250*x^9*y^9 - 1598832213160*x^9*y^8
        - 667644220880*x^9*y^7 - 201760616200*x^9*y^6
        - 44874471980*x^9*y^5 - 7328752600*x^9*y^4
        - 860828540*x^9*y^3 - 69526600*x^9*y^2 - 3545360*x^9*y
        - 95440494357*x^8*y^10 - 190880988714*x^8*y^9
        - 183539412225*x^8*y^8 - 110688383988*x^8*y^7
        - 46221522984*x^8*y^6 - 13968042660*x^8*y^5
        - 3106694214*x^8*y^4 - 507375180*x^8*y^3 - 59595822*x^8*y^2
        - 4813380*x^8*y - 6525845768*x^7*y^9
        - 13051691536*x^7*y^8 - 12549703400*x^7*y^7
        - 7568436512*x^7*y^6 - 3160446016*x^7*y^5
        - 955079840*x^7*y^4 - 212423536*x^7*y^3 - 34692320*x^7*y^2
```

```
            - 4074928*x^7*y - 439239619*x^6*y^8 - 878479238*x^6*y^7
            - 844691575*x^6*y^6 - 509413996*x^6*y^5 - 212722328*x^6*y^4
            - 64284220*x^6*y^3 - 14297738*x^6*y^2 - 2335060*x^6*y
            - 28960854*x^5*y^7 - 57921708*x^5*y^6 - 55693950*x^5*y^5
            - 33587736*x^5*y^4 - 14025648*x^5*y^3 - 4238520*x^5*y^2
            - 942708*x^5*y - 1856465*x^4*y^6 - 3712930*x^4*y^5
            - 3570125*x^4*y^4 - 2153060*x^4*y^3 - 899080*x^4*y^2
            - 271700*x^4*y - 114244*x^3*y^5 - 228488*x^3*y^4
            - 219700*x^3*y^3 - 132496*x^3*y^2 - 55328*x^3*y
            - 6591*x^2*y^4 - 13182*x^2*y^3 - 12675*x^2*y^2
            - 7644*x^2*y - 338*x*y^3 - 676*x*y^2 - 650*x*y
            - 13*y^2 - 26*y;
    end case;
    error "N must be in {2,3,4,5,7,8,9,13,16,25} not " * Sprint(N);
end function;
function DyPhiX0(N,p,x,y)
    if N eq 1 then
Phi := ClassicalModularPolynomial(p);
return Evaluate(Derivative(Phi,2),[x,y]);
    end if;
    case [N,p]:
    when [2,2]:
        return -(8192*x*y+48*x+1);
    when [4,2]:
        return -(512*x*y+16*x+32*y+1);
    when [8,2]:
        return (32*y+8)*x^2-1;
    when [16,2]:
        return 8*x^2*y-1;
    when [3,3]:
        return -(1594323*(x*y)^2 + 52488*x^2*y
            + 270*x^2 + 1458*x*y + 36*x + 1);
    when [9,3]:
        return -(2187*x^2*y^2 + 486*x^2*y + 27*x^2
            + 729*x*y^2 + 162*x*y + 9*x + 81*y^2 + 18*y + 1);
    when [5,5]:
        return -1220703125*x^4*y^4 - 234375000*x^4*y^3 - 14765625*x^4*y^2
            - 325000*x^4*y - 1575*x^4 - 7812500*x^3*y^3 - 1406250*x^3*y^2
            - 78750*x^3*y - 1300*x^3 - 46875*x^2*y^2 - 7500*x^2*y
            - 315*x^2 - 250*x*y - 30*x - 1;
    when [25,5]:
        return -3125*x^4*y^4 - 2500*x^4*y^3 - 1125*x^4*y^2 - 250*x^4*y
            - 25*x^4 - 3125*x^3*y^4 - 2500*x^3*y^3 - 1125*x^3*y^2
            - 250*x^3*y - 25*x^3 - 1875*x^2*y^4 - 1500*x^2*y^3
```

```
        - 675*x^2*y^2 - 150*x^2*y - 15*x^2 - 625*x*y^4 - 500*x*y^3
        - 225*x*y^2 - 50*x*y - 5*x - 125*y^4 - 100*y^3 - 45*y^2
        - 10*y - 1;
when [7,7]:
    return -96889010407*x^6*y^6 - 47455841832*x^6*y^5
        - 9281329610*x^6*y^4 - 896014784*x^6*y^3
        - 42605745*x^6*y^2 - 845152*x^6*y - 4018*x^6
        - 1694851494*x^5*y^5 - 807072140*x^5*y^4
        - 151531912*x^5*y^3 - 13714512*x^5*y^2
        - 579670*x^5*y - 8624*x^5 - 28824005*x^4*y^4
        - 13176688*x^4*y^3 - 2319366*x^4*y^2 - 186592*x^4*y
        - 5915*x^4 - 470596*x^3*y^3 - 201684*x^3*y^2
        - 31556*x^3*y - 1904*x^3 - 7203*x^2*y^2
        - 2744*x^2*y - 322*x^2 - 98*x*y - 28*x - 1;
when [13,13]:
    return -302875106592253*x^12*y^12 - 559154042939544*x^12*y^11
        - 492844108360175*x^12*y^10 - 270202644024040*x^12*y^9
        - 101548685995848*x^12*y^8 - 27278035310240*x^12*y^7
        - 5308650035234*x^12*y^6 - 743135513640*x^12*y^5
        - 72740011630*x^12*y^4 - 4699998160*x^12*y^3
        - 179749752*x^12*y^2 - 3246152*x^12*y - 15145*x^12
        - 21505924728444*x^11*y^11 - 39427528668814*x^11*y^10
        - 34464622962250*x^11*y^9 - 18706336893972*x^11*y^8
        - 6943499897152*x^11*y^7 - 1836021607420*x^11*y^6
        - 350020881444*x^11*y^5 - 47636891900*x^11*y^4
        - 4476308408*x^11*y^3 - 271153740*x^11*y^2 - 9217936*x^11*y
        - 124852*x^11 - 1516443410339*x^10*y^10
        - 2757169836980*x^10*y^9 - 2386012358925*x^10*y^8
        - 1279065770528*x^10*y^7 - 467350954616*x^10*y^6
        - 121056369720*x^10*y^5 - 22437235990*x^10*y^4
        - 2931501040*x^10*y^3 - 258248562*x^10*y^2
        - 13905320*x^10*y - 354536*x^10 - 106044993730*x^9*y^9
        - 190880988714*x^9*y^8 - 163146144200*x^9*y^7
        - 86090965324*x^9*y^6 - 30814348656*x^9*y^5
        - 7760023700*x^9*y^4 - 1380752984*x^9*y^3
        - 169125060*x^9*y^2 - 13243516*x^9*y - 534820*x^9
        - 7341576489*x^8*y^8 - 13051691536*x^8*y^7
        - 10980990475*x^8*y^6 - 5676327384*x^8*y^5
        - 1975278760*x^8*y^4 - 477539920*x^8*y^3 - 79658826*x^8*y^2
        - 8673080*x^8*y - 509366*x^8 - 501988136*x^7*y^7
        - 878479238*x^7*y^6 - 724021350*x^7*y^5 - 363867140*x^7*y^4
        - 121555616*x^7*y^3 - 27550380*x^7*y^2 - 4085068*x^7*y
        - 333580*x^7 - 33787663*x^6*y^6 - 57921708*x^6*y^5
        - 46411625*x^6*y^4 - 22391824*x^6*y^3 - 7012824*x^6*y^2
```

```
            - 1412840*x^6*y - 157118*x^6 - 2227758*x^5*y^5
            - 3712930*x^5*y^4 - 2856100*x^5*y^3 - 1291836*x^5*y^2
            - 359632*x^5*y - 54340*x^5 - 142805*x^4*y^4
            - 228488*x^4*y^3 - 164775*x^4*y^2 - 66248*x^4*y - 13832*x^4
            - 8788*x^3*y^3 - 13182*x^3*y^2 - 8450*x^3*y - 2548*x^3
            - 507*x^2*y^2 - 676*x^2*y - 325*x^2 - 26*x*y - 26*x - 1;
    end case;
    error "N must be in {2,3,4,5,7,8,9,13,16,25} not " * Sprint(N);
end function;
```

frobenius_representation.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                          David Kohel                               //
//                  Parametrized Representations                      //
//                     of Frobenius Isogeny                           //
//                     and Norm computation                           //
//                                                                    //
////////////////////////////////////////////////////////////////////////
function AGMFrobeniusNorm(N,p,x)
    case [N,p]:
    when [2,2]:
        y := FrobeniusImage(x);
        // Square of Frobenius:
        // Num := (256*y+1)*(-256*y*(256*y+1)+16*x+1);
        // Den := (256*x+1)*(512*y*(64*y+1)-8*x+1);
        // Using the fact that Norm(256*y+1) = Norm(256*x+1):
        Num := (-256*y*(256*y+1)+16*x+1);
        Den := (512*y*(64*y+1)-8*x+1);
        return Exp(Trace(Log(Num div Den)) div 2);
    when [4,2]:
        // Square of Frobenius:
        // Num := 32*y+1;
        // Den := 8*x+1;
        // But since Norm(32*y+1) = Norm(32*x+1):
        return Exp(Trace(Log((1+32*x) div (1+8*x))) div 2);
    when [8,2]:
        // Minimizing degrees:
        // Num := 2*(4*y+1)*x-1;
        // Den := 2*(4*y+1)*x+1;
        // or (exploiting cancellation):
        // Num := (-4*x+1)*(4*y+1);
        // Den := 4*y-1;
        // But since Norm(-4*x+1) = Norm(-4*y+1):
        return Exp(Trace(Log(1+4*x)));
    when [16,2]:
        // Use that the square of x is the function of level 8:
        return Exp(Trace(Log(1+4*x^2)));
    when [3,3]:
        // Square of Frobenius:
        Num := (3*x+1)*(-19683*x^2-486*x+1);
        Den := (243*x+1)*(-27*x^2+18*x+1);
        return Exp(Trace(Log(Num div Den)) div 2);
```

```
    when [9,3]:
        // Square of Frobenius:
        g := (27*x^2+9*x+1);
        Num := (3*x+1)*(27*x^2+1)*(-243*(81*x^2*g^2+2*x*g)+1);
        Den := (-27*x^2+1)*(243*g*x+1)*(27*x^2+g^2);
        return Exp(Trace(Log(Num div Den)) div 2);
    when [5,5]:
        // Square of Frobenius:
        x2 := 5^2*x; x3 := 5*x2;
        Num := (5*(x + 2)*x + 1)*(-x3^2 - 4*x3 + 1);
        Den := (5*(x2 + 2)*x2 + 1)*(-x^2 + 4*x + 1);
        // Log doesn't converge, so we have to call Norm:
        return Sqrt(Norm(Num div Den));
    when [25,5]:
        // Square of Frobenius...
        x := 25*x^5 + 25*x^4 + 15*x^3 + 5*x^2 + x;
        x2 := 5^2*x; x3 := 5*x2;
        Num := (5*(x + 2)*x + 1)*(-x3^2 - 4*x3 + 1);
        Den := (5*(x2 + 2)*x2 + 1)*(-x^2 + 4*x + 1);
        // Log doesn't converge, so we have to call Norm:
        return Sqrt(Norm(Num div Den));
    when [7,7]:
        // Square of Frobenius...
        x2 := 7^2*x;
        Num := (x^2 + 5*x + 1)*(-7^7*x^4 - (2*x2^2 + 9*x2 + 10)*x2 + 1);
        Den := (x2^2 + 5*x2 + 1)*(-7*x^4 + 7*(10*x^2 + 9*x + 2)*x + 1);
        // Log doesn't converge, so we have to call 'Norm'.
        return Sqrt(Norm(Num div Den));
    when [13,13]:
        // Square of Frobenius...
        Num := (x^4 + 19*x^3 + 20*x^2 + 7*x + 1)*
            (-((13*x)^6 + 10*(13*x)^5 + 46*(13*x)^4
            + 108*(13*x)^3 + 122*(13*x)^2 + 38*(13*x)) + 1);
        Den := ((13*x)^4 + 7*(13*x)^3 + 20*(13*x)^2 + 19*(13*x) + 1)*
            (-x^6 + 38*x^5 + 122*x^4 + 108*x^3 + 46*x^2 + 10*x + 1);
        // Log doesn't converge, so we have to call 'Norm'.
        return Sqrt(Norm(Num div Den));
    end case;
end function;
```

analytic_frobenius_lift.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                            David Kohel                             //
//                AGM-X0(N) Analytic Lifting Functions                //
//                                                                    //
////////////////////////////////////////////////////////////////////////
import "modular_correspondences.m" : PhiX0;
function AnalyticAGMProduct(N,n)
    case N:
    when 2:
        c :=
            [
            [ 1 ],
            [],
            [],
            [],
            [ -48 ],
            [],
            [],
            [],
            [ 2304 ],
            [],
            [],
            [],
            [ -4096 ],
            [],
            [],
            [],
            [ 5701632 ],
            [],
            [],
            [],
            [ -28311552 ],
            [],
            [],
            [],
            [],
            [ 1845493760 ],
            [],
            [],
            [ -113548197888 ],
            []
```

```
        ];
    e :=
        [
        [ 2 ],
        [],
        [],
        [],
        [ 1 ],
        [],
        [],
        [],
        [ 2 ],
        [],
        [],
        [],
        [ 3 ],
        [],
        [],
        [],
        [ 4 ],
        [],
        [],
        [],
        [ 5 ],
        [],
        [],
        [],
        [],
        [ 6 ],
        [],
        [],
        [ 7 ],
        []
        ];
when 3:
    c :=
        [
        [ 1 ],
        [],
        [ -36 ],
        [ 1026 ],
        [],
        [ 9720 ],
        [],
```

```
        [ 1051947 ],
        [],
        [ 9998964, 93927276 ],
        [],
        [],
        [ 10237679424, 1204441407729 ],
        [],
        [ 10485875125584 ],
        [ 199707657158790 ],
        [],
        []
       ];
    e :=
        [
        [ 3 ],
        [],
        [ 1 ],
        [ 2 ],
        [],
        [ 3 ],
        [],
        [ 4 ],
        [],
        [ 5, 6 ],
        [],
        [],
        [ 7, 8 ],
        [],
        [ 9 ],
        [ 10 ],
        [],
        []
       ];
when 4:
    c :=
        [
        [ 1 ],
        [],
        [],
        [],
        [ -16, 240 ],
        [],
        [],
        [],
```

```
    [],
    [ -3584, -3584 ],
    [],
    [],
    [ 13029376 ],
    [ -8192 ],
    [],
    [],
    [ 23003136, 79364096 ],
    [],
    [],
    [],
    [ 16130244608 ],
    [ -1142947840 ],
    [],
    [],
    [ -329521299456, 172678069616640 ],
    [],
    [],
    [],
    [ 255525589614592, 2189113423822848 ],
    []
   ];
e :=
    [
    [ 2 ],
    [],
    [],
    [],
    [ 1, 2 ],
    [],
    [],
    [],
    [],
    [ 3, 4 ],
    [],
    [],
    [ 6 ],
    [ 5 ],
    [],
    [],
    [ 7, 8 ],
    [],
    [],
```

```
        [],
        [ 10 ],
        [ 9 ],
        [],
        [],
        [ 11, 12 ],
        [],
        [],
        [],
        [ 13, 14 ],
        []
        ];
when 5:
    c :=
        [
        [ 1 ],
        [ -30, 585 ],
        [ -9400, -146850 ],
        [ -713250, 87438500 ],
        [ 2760783750, 28269399375 ],
        [ 113402862500, 644012784375 ],
        [ -124730895375000, 5676373418500000 ],
        [],
        [ 221235630208593750, 10012876431465234375,
        1659106750825671875000, 11240689969552722656250 ],
        [ 4983813291973640625000, -313287979085919113281250 ],
        [ 75752133511878023018554687500 ],
        [ 12291236670671272851562500, 13418481251050945595410156250,
        139001822619438194402929687500 ],
        [ 17217226985354097968847656250,
        30223876486571136479207275390625 ]
        ];
    e :=
        [
        [ 5 ],
        [ 1, 2 ],
        [ 3, 4 ],
        [ 5, 6 ],
        [ 7, 8 ],
        [ 9, 10 ],
        [ 11, 12 ],
        [],
        [ 13, 14, 15, 16 ],
        [ 17, 18 ],
```

```
        [ 20 ],
        [ 19, 21, 22 ],
        [ 23, 24 ]
        ];
when 7:
    c := [
        [ 1 ],
        [ -28, 462, -5824, 61705 ],
        [ -575064, -11275390, -86909144, -316615264 ],
        [ -508741716, 330450650768, 12968142035296, 227091056598987 ],
        [ 2324017066686112, 14985679520285362, 60912373421904872,
        147255139154249112 ],
        [ 182623317826756856, 337333362400574890,
        -336227891797005243448, 96002763911702873300320 ],
        [ 8339600561672138823755992, 31707515829400198190137 4130,
        665024726557488578401552 7568, 866295702850447286955172 29193 ],
        [ 74215353034550116536449 8002096,
        96426570810766552867631791 45652,
        85456538772622182827099502935920,
        54352793708026008760670656687 0219 ],
        [ 25525639016245897110215583681 27224,
        85660062312066793965376168092 85466,
        21460074411631763619805320105341400 ],
        [ -74353587302217238357004100314 0302156,
        21812089830725587966846557464054 1335264,
        20799643274577677024954943091829131470468,
        86766102876656007338466411835009 7855544832 ]
        ];
    e := [
        [ 7 ],
        [ 1, 2, 3, 4 ],
        [ 5, 6, 7, 8 ],
        [ 9, 10, 11, 12 ],
        [ 13, 14, 15, 16 ],
        [ 17, 18, 19, 20 ],
        [ 21, 22, 23, 24 ],
        [ 25, 26, 27, 28 ],
        [ 29, 30, 31 ],
        [ 32, 33, 34, 35 ]
        ];
when 8:
    c :=
        [
        [ 1 ],
```

```
    [],
    [],
    [ 8 ],
    [ 80 ],
    [],
    [],
    [],
    [ 256 ],
    [ 8704 ],
    [],
    [],
    [ 45056, 536576 ],
    [],
    [],
    [],
    [ 155648000 ],
    [ 7733248 ],
    [],
    [],
    [ 1395654656, 19840106496 ],
    [],
    [],
    [],
    [ 292376543232, 3755328143360 ],
    [],
    [],
    [],
    [ 796413906649088 ],
    [ 53647899623424 ]
   ];
e :=
    [
    [ 2 ],
    [],
    [],
    [ 2 ],
    [ 4 ],
    [],
    [],
    [],
    [ 6 ],
    [ 8 ],
    [],
    [],
```

```
            [ 10, 12 ],
            [],
            [],
            [],
            [ 16 ],
            [ 14 ],
            [],
            [],
            [ 18, 20 ],
            [],
            [],
            [],
            [ 22, 24 ],
            [],
            [],
            [],
            [ 28 ],
            [ 26 ]
           ];
    when 9:
        c := [
            [ 1 ],
            [],
            [ -9, -252 ],
            [ 54, 649674 ],
            [ 5265 ],
            [ 486, 33048, 2925234, 98492517 ],
            [],
            [ -284310, 95057955, 746944318962 ],
            [ 2721811167, 1628537873607 ],
            [ 167507325289521 ],
            [ 19730337615, -14931128309481, 11007827988715881 ],
            [ 9071347984480242, 2485532229845377635 ],
            [ 138174660, 20255812587593298, 295146811991132679753, 7376345866076732091
],
            [ 139584431011691609073 ],
            [ 5816757370191771585663, 67252077270180213803113166l ],
            [ 3617112905878878247401252, 816011585988223471214101560, 33595131748304142
],
            [ 38394987838149953214747 ],
            [ 628432084308651544694743283493 0 ]
           ];
        e := [
            [ 3 ],
```

```
        [],
        [ 1, 3 ],
        [ 2, 6 ],
        [ 4 ],
        [ 3, 5, 7, 9 ],
        [],
        [ 6, 8, 12 ],
        [ 10, 13 ],
        [ 15 ],
        [ 11, 14, 16 ],
        [ 18, 19 ],
        [ 9, 17, 21, 24 ],
        [ 20 ],
        [ 22, 27 ],
        [ 25, 26, 30 ],
        [ 23 ],
        [ 31 ]
        ];
when 13:
    c := [
        [ 1 ],
        [ -26, 351, -3224, 22165, -117546, 468013, -1169792,
        25898470, -168725687, 687023064, -1695680194, 28091872042 ],
        [ -561249, -30714549086, -428755206802, -4348371100359,
        -27548699217598, 138599607472950, 5699009810586170,
        774062321245391178, 5865092557705744415, 36612906614488201010,
        199897137248937103144, 1073213480542420436570,
        6391816768712682760310, 125068087775895091246991 ],
        [ -14592474, -182405925, -1430062452, -7763196168,
        -93805559458, 75504441118991144, 2200526218950753839 1190,
        -257135994332195516650242, -10810977852741121511604408,
        -11188292478371389233004425 0, -80353410365207377621581 5712,
        -32790917475712945035614 5872, -22808909807711906534620 8549420,
        -24461310309115652082 0818723583986,
        -22721516863939025459394880905014,
        -18057235896457026656504111127723 2,
        -94458632785394170316897067895448 8,
        88903106957210822910771591297159 29 ],
        [ -62962662616179134931066230299,
        84468487524789674697884520025239774,
        12654528206097887251125424488683959 44,
        10509687491896464238618209585062264 4052,
        79591519251650288540660109240339770 7850,
        57871908107551641789749345248239761 85746,
```

```
            4418684756169062755894431210897144252528 6,
            2629969114995633471649764311561003668103 30,
            1491656965814972965067096015934281457566292,
            6577926766003034643223732295138493650871915,
            2437333479198792746732246925293120273005283 6,
            1051447475676456084887323044635429322894648030,
            4649273459273183568054296791398698410112784151,
            3358630889155424443932621432592439053381935683 17 ],
            [ -8190011440026, -102375143000325, -802621121122548,
            -500849535399421806, -5602515736989734550,
            -50180322585622327020, -384700687816611454938,
            -1985289527849976953280, 3556892761053419670444 9,
            1244971827736624107312774539592662487 2,
            7846446644480074797134883938605023539695325 0,
            3484324022164501540218937429892176740428015663502,
            3228057676775850000351702348631746874748399540753 5,
            2431623829762729680478019600768890181068883881103 22,
            2468567555951427903437981014222633798219550871263014,
            8343367042644536152194459449954749959367354915734854,
            5116416754865601111696907110245490969309627404268511 8,
            5388038055778825522174986499637091264305122246274353 22,
            4366442535501802160567114904189209277059757389395188452,
            2405703786382373836423382478056086182455870321352451 9308,
            1312487346557553116046773803004808886210267811714293122668,
            1335062538892533420663421162256488639290797125325190103 0694 ],
            [ 1591876259698826176446565661104122313613091247 2,
            1253841551990769970674236563185436694582553627040625231 48,
            1255825623754636398665827280958806306550031669020965292433 82,
            1187994766617686170923884925463060093553937651358759492327867,
            3876648359514974206543730534694307944139673169619390911242924 5,
            1513367947773126176078764943715489398658155275729647557006864 70,
            1613261884872758009323123814985837104442050905611599659637512 17,
            -1450241384095559997111977359805479641205791924199911953309962 8764,
            -2296007845852934684182667042977251469947066484721576237689632 02926,
            -2219333166762269885765731227365978457225718644265341399568147 1763863,
            -1818295046431280575929027944065360120322008356615947956488327 33995404,
            -1093080915341626529519979256122704837855796380732815322245578 2631140876,
            -6945508733386577766384167023798784586450979496091519053715424 8297742270
],
            [ 8012867809735825188696812780984982880270885271766806246564 216,
            -1653257431023191790205461606347968046194030989729463675838957 750100,
            -1402222137310569105273969905568592721472196587622783430131738 927611326,
            -2895628829182632743297390992330448973572364580032440956345342 56990882972,
            -2939664153291167308519766512347811821628686405897586878611458 26404856983,
```

```
            169502350102081009099044224320558202520968044966024449355662844340701910146
            464881234024221657924610856977912133058071596415546127000742575675383017636
            364756944034342188979434623299748125317687575270170824398910114592768246814
            389989594211230911145756147549312845118286569550071439720363388929096396897
            367265535931991771765741451244108345946685278124773676071557435562211541947
            307240408083265770339256329940585505026470573108898886684416432079986803077
            249083581460587794305361906585401360155538508174314145972030546347121583857
            184444694663896679786334474708141814969071092067827437520063173958301604127
            128376972358534496070826219595539511260766538381805261033213743528098866757
            850072368794668539680053107181885096422472044043225552751443193746020390297
            561007670970061218075662469976855818328417420352477382720587963399729441597
            313803444549564604687286250052349369953378310740595165004392278101395173407
]
            ];
        e := [
            [ 13 ],
            [ 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 14 ],
            [ 8, 13, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 28 ],
    [ 9, 10, 11, 12, 14, 20, 27, 29, 30, 31, 32, 33, 35, 36,
    37, 38, 39, 40 ],
            [ 34, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 53, 54, 56 ],
            [ 17, 18, 19, 22, 23, 24, 25, 26, 28, 43, 52, 57, 58, 59,
            60, 61, 62, 63, 64, 65, 67, 68 ],
            [ 55, 66, 69, 70, 72, 73, 74, 75, 76, 78, 79, 81, 82 ],
    [ 71, 77, 80, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93,
    94, 95, 96 ]
            ];
    when 16:
        c := [
            [ 1 ],
            [],
            [ 4 ],
            [],
            [],
            [ 32 ],
            [ 192 ],
            [],
            [ 2816 ],
            [],
            [ 25600 ],
            [],
            [ 323584 ],
            [],
            [ 4014080 ],
```

```
    [],
    [ 56950784 ],
    [],
    [],
    [],
    [ 705691648 ],
    [],
    [ 134490357760 ],
    [ 10007609344 ],
    [ 1935771959296 ],
    [],
    [ 26845089103872 ],
    [],
    [],
    [ 392648594554880 ]
   ];
e := [
    [ 2 ],
    [],
    [ 4 ],
    [],
    [],
    [ 8 ],
    [ 12 ],
    [],
    [ 16 ],
    [],
    [ 20 ],
    [],
    [ 24 ],
    [],
    [ 28 ],
    [],
    [ 32 ],
    [],
    [],
    [],
    [ 36 ],
    [],
    [ 44 ],
    [ 40 ],
    [ 48 ],
    [],
    [ 52 ],
```

```
        [],
        [],
        [ 56 ]
        ];
when 25:
    c := [
        [ 1 ],
        [ -5, 10, 120, -590 ],
        [ -50, -825, -1850, 600350, 1789925, -75602150, 30599407150 ],
        [ -250, -500, 9500, 1162750, -131141500, -711533250,
        88854843500, 547482543000, 2945432282375, -5419168578752750 ],
        [ 145000, 3466250, -2243352500, 2134552092500,
        -1343389838750, -117801884126875, 22320405957891250,
        2354651797998768125, -3146290673447264103750 ],
        [ -206250, -487500, 32850000, -3356096875, -12966018750,
        -96725446190625, -210371136987500, 635341643534375,
        248232922664700000, 873097481037978125, 25183988334948209375,
        16324247392478565625, 3093377194810720394581250,
        -72993127663064170744820625O ],
        [ -320581968843750, -1122653576257022843750,
        -1078635505491249848437 5, 4073464036663680593750,
        5343076936974475432812 5, 9906955823889378570156 5,
        6797997018766297053703125, -5118035585142557538968750,
        -11957239251023997710282 5, 2913817233770711323305890625,
        38221891900163528998137671875,
        -56585991122401121177117769312500 ],
        [ 23380000000, 7289375000, 3753241909062500,
        1545802017998507812 5, -44763981430326730025779687 0,
        -36926847529747391899389117 875,
        41943231706968976741475335156 5,
        -97672288216766004295638833835937 5,
        -74422002786644745497811899853906 5 ],
        [ -51562500, -121875000, -22310323772424061229687500,
        -25654506963228988674765625 , -11517057238844638577548437500,
        16946401892402674735976713281250,
        -55592994259461614591480730468 5,
        -29061288956814997709299355950390625,
        14858255657520967617580660005871093 5,
        34182755359310855082164880551578125 0,
        -10886626036107465014026467970296355468 5,
        -12833875401238665275415958375549359375 0,
        -21389099482793312995347213013208991406250,
        58287884173964373746377847950510909186328125 ],
        [ 714914062500, -5646234375000, 8843164574218750,
```

```
1179711034601562500, -41614781597347758951652269687500000,
1142480310261941930444391772960156250,
850530917727852646453755713194726562 5,
-434755648584230316896273796028984375 00,
109158915554451280164020245689663861328125,
933772846239491604288645877371729707031 25,
147193052361300880742046636125994917335937500,
-250667394372732349618686576801769626683593750,
-627853647663107414068238941199559565583027343 75,
347747900302784592892600149491520548933098906250 00 ],
[ 983506948558554687500,
788092376006036279721154639941406 25,
-555563667273422115359797653559651367 1875,
-937354496815926107282142677475757812 5000,
515893111452181231142748855238257037109 3750,
141762403786600352201692539157320577929 68750,
-521696584030315794343296655369097612870 1171875,
-125980487922971974854525848180818508719 43359375,
-366613093330280465309758223926890691735058 59375,
214578119233685330671918938536431814041445312500,
-693711230900453082525911664108321192601307 6171875,
-227717002285835443818633098226864176802798 53515625,
-962616893939181614606023811479300588711186 7744140625,
458590375737329349108774774268042941557692850 7246093750 ],
[ 7768759375000000, -1319860856505047167968750,
-623373647254362984375000 00,
270477908367736615422067129390014921438715 8203125,
-701315114546910884460886473316101844133119 482421875,
-262445595956207576763101611228855006136650 6542968750,
251570029133934822617155138489538549464791966898437500 00,
902138359716658663193639140397507971958158944420898437 50,
-380980434714730251340037985907937658972314701755009765 6250,
302212093763633471574131978864572605083339943246811826 1718750 ],
[ 12901183593750000, -169495174804687500,
-2252608576171875000, -3366697511718750000,
535407744490312557537555603408067361268505859 3750,
-140835397709545098550522404564153493082050766601562 50,
-630957916600072064675133226574599813350736276855468 75,
614898823002865867974470698011327464550644829101562 50,
117757677569485543203618281193520620480137178051757 8125,
416844255334636856248791340111914675242577839594726 56250,
-257365141075474877259422697720209682784119928609755 85937500,
-696869389853328807324686253102950491763025630463959 96093750,
142561052381432127958066594397928422216143727945791 43310546875,
```

```
            421559768577467074098952148283074053761414395315613093261718 75,
            -173076699853589294766932376614735752547766101732250389062500 0000,
            937413216460312510178951656744275339552779666290698684943847656 25
]
            ];
        e := [
            [ 5 ],
            [ 1, 2, 5, 10 ],
            [ 4, 6, 7, 11, 12, 15, 20 ],
            [ 5, 8, 9, 14, 16, 17, 21, 22, 25, 30 ],
            [ 10, 13, 19, 23, 24, 31, 32, 35, 40 ],
            [ 11, 12, 15, 18, 20, 26, 28, 29, 33, 34, 36, 37, 45, 50 ],
            [ 27, 39, 41, 42, 43, 44, 46, 47, 51, 52, 55, 60 ],
            [ 21, 22, 30, 38, 53, 56, 57, 61, 62 ],
            [ 16, 17, 48, 49, 54, 58, 59, 63, 66, 67, 71, 72, 75, 80 ],
            [ 24, 25, 31, 35, 64, 68, 69, 70, 76, 77, 81, 82, 85, 90 ],
            [ 40, 65, 73, 74, 78, 79, 83, 84, 86, 87, 91, 92, 95, 100 ],
            [ 32, 45, 50, 88, 93, 94, 101, 102, 105, 110 ],
            [ 33, 34, 36, 37, 89, 96, 97, 98, 99, 104, 106, 107,
            111, 112, 115, 120 ]
            ];
    else
        assert false;
    end case;
    return c[1..n], e[1..n];
end function;
function AnalyticAGMLift(N,x)
    R := Parent(x);
    prec := Precision(R);
    _, p := IsPrimePower(N);
    cffs, exps := AnalyticAGMProduct(N,prec);
    P := PolynomialRing(Integers()); X := P.1;
    for i in [1..prec] do
        x := ChangePrecision(R,Max(2,i))!x;
        xpow := [x];
        for j in [1..Max(&cat exps[1..i])-1] do
            Append(~xpow,x*xpow[j]);
        end for;
        y := x^p;
        for j in [2..i] do
            if #cffs[j] ne 0 then
y *:= 1 + &+[
    cffs[j][k]*xpow[exps[j][k]] : k in [1..#exps[j]] ];
            end if;
```

```
        end for;
        x := y;
        vprint AGM, 3 :
            "Analytic val:", Valuation(PhiX0(N,x,FrobeniusImage(x)));
    end for;
    return x;
end function;
```

analytic_decomposition.m

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                            David Kohel                             //
//                AGM-X0(N) Analytic Lifting Functions                //
//                                                                    //
////////////////////////////////////////////////////////////////////////
import "modular_correspondences.m" : PhiX0;
function ProductRepresentation(f,p,n)
    // f is a power series over the integers
    S := Parent(f); t := S.1;
    P := PolynomialRing(BaseRing(S)); x := P.1;
    polys := [P|];
    for i in [0..n-1] do
        m := AbsolutePrecision(f);
vals := [ j : j in [0..m-1] |
    Valuation(Coefficient(f,j),p) le i ];
        g := &+[ Coefficient(f,j)*x^j : j in vals ];
        Append(~polys,g);
        f /:= Evaluate(g,t);
    end for;
    return polys;
end function;


intrinsic AnalyticAGMFrobenius(N::RngIntElt,prec::RngIntElt) -> SeqEnum
    {}
    S<x> := LaurentSeriesRing(Rationals());
    AssertAttribute(S,"Precision",prec);
    P<y> := PolynomialRing(S);
    return Roots(PhiX0(N,p,x,y))[1][1];
end intrinsic;


intrinsic AnalyticAGMPowerProduct(N::RngIntElt,prec::RngIntElt) -> SeqEnum
    {Return a sequence of polynomials, the initial k
    elements of which have product which approximates
    the analytic p^i correspondence on X_0(N).}
    bool, p := IsPrimePower(N);
    require bool : "Argument 1 must be a prime power.";
    return AnalyticAGMPowerProduct(N,1,prec);
end intrinsic;
```

```
intrinsic AnalyticAGMPowerProduct(
    N::RngIntElt,i::RngIntElt,prec::RngIntElt) -> SeqEnum
    {Return a sequence of polynomials, the initial k
    elements of which have product which approximates
    the analytic p^i correspondence on X_0(N).}
    bool, p := IsPrimePower(N);
    require bool : "Argument 1 must be a prime power.";
    S<x> := LaurentSeriesRing(Rationals());
    AssertAttribute(S,"Precision",N*(i*prec+1));
    P<y> := PolynomialRing(S);
    for j in [1..i] do
        yx := Roots(PhiX0(N,p,x,y))[1][1];
        if j ne i then x := yx; end if;
    end for;
    return ProductRepresentation(yx,p,prec);
end intrinsic;
```

class_polynomial.m

```
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
function AGMInitialValues(j,p,n)
    prec := ((n+1) div 2)+1+(4 div p);
    case p:
    when 2: x := 1/j;
    when 3: x := 1/j;
    when 5: x := 1/j;
    when 7: x := 1/(j+1);
    when 13: x := 1/(j-5);
    end case;
    return x, prec;
end function;
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////


intrinsic AGMClassPolynomial(E::CrvEll,N::RngIntElt :
    Precision := 0) -> RngUPolElt
    {}
    require Type(BaseRing(E)) eq FldFin :
        "Argument 1 must be defined over a finite field.";
    p := Characteristic(BaseRing(E));
    require N in {2,4,8,16,3,9,5,25,7,13} and N mod p eq 0 :
        "Argument 2 must be in {2,4,8,16,3,9,5,25,7,13} " *
        "and be divisible by the characteristic.";
    n := Degree(BaseRing(E));
    j := jInvariant(E);
    x, prec := AGMInitialValues(j,p,n);
    prec := Max([Precision,prec,16]);
    x := AGMLift(N,x,prec);
    case N:
    when 2:
        x1 := 2^12*x;
        x2 := 1 div x;
    when 4:
        x1 := 2^8*x;
        x2 := 1 div x;
    when 8:
        x1 := 4*x;
        x2 := (1-4*x) div (1+4*x);
    when 16:
```

```
        x1 := 2*x;
        x2 := (1-2*x) div (1+2*x);
    when 3:
        x1 := 729*x;
        x2 := 1 div x;
    when 9:
        x1 := 27*x;
        x2 := 1 div x;
    else
        require false : "Level not yet treated.";
    end case;
    ZZ := Integers();
    f := Eltseq(MinimalPolynomial(x1) * MinimalPolynomial(x2));
    M := LLL(VerticalJoin(
        Matrix(#f,[ ZZ!f[i] : i in [1..#f] ]),
        p^(prec-4)*IdentityMatrix(ZZ,#f)));
    return PolynomialRing(ZZ)!Eltseq(M[1]);
end intrinsic;
```