QUADRATIC ISOGENY PRIMES

USER'S GUIDE

BARINDER S. BANWAIT

ABSTRACT. This document gives an overview of the implementation of *Quadratic Isogeny Primes*, a software package which computes, for a given quadratic field K which is not imaginary quadratic of class number one, a tight superset of the set of *isogeny primes for* K, namely, the primes p for which there exists an elliptic curve over K admitting a K-rational p-isogeny. This *user's guide* is to be considered a companion to the author's article *Explicit isogenies of prime degree over quadratic fields* which explains the theory upon which the algorithms and implementation are based. Some extended examples explaining how one may exactly compute the set of isogeny primes from the superset output by the program for specific Ks is also included here to illustrate the utility of the package.

1. Introduction

A key notion in the study of elliptic curves is that of an **isogeny**, defined as a surjective morphism $\phi : E \to E'$ between elliptic curves which maps the identity element of E to that of E'; or equivalently, precisely those surjective morphisms of curves which induce a group homomorphism between the underlying group of geometric points on E and E'. The **degree of** ϕ is defined as its degree when viewed as a morphism of curves - that is, the degree of the extension of function fields induced by the pullback of ϕ - and ϕ is said to be **separable** if this field extension is separable. If N is the degree of ϕ, one says that ϕ is an N-isogeny. If E and E' are both defined over the same field K, ϕ is said to be K**-rational** if it commutes with the natural Galois action on E and E'. It follows from the basic Galois theory of elliptic function fields that a separable K-rational N-isogeny can be decomposed as a chain of K-rational p-isogenies for p prime, so the isogenies of prime degree carry an elevated significance.

We henceforth take K to be a number field - so in particular all isogenies are separable. It is a corollary of a result of Shafarevich from 1962 [**?**] that an elliptic curve E/K not admitting any complex multiplication over K admits only finitely many K-rational isogenies of prime degree. In the base case of $K = \mathbb{Q}$, no elliptic curve admits complex multiplication over $\mathbb{Q}$, so every rational elliptic curve admits only finitely many prime degree isogenies.

In a seminal paper involving a delicate analysis of the Néron model of the Eisenstein quotient of the Jacobian of the modular curve $X_0(N)$, Mazur succeeded in 1978 in proving an *explicit uniform version* of the above corollary of Shafarevich for $K = \mathbb{Q}$.

2010 *Mathematics Subject Classification.* 11G05 (primary), 11Y60, 11G15. (secondary).

Theorem 1.1 (Mazur, Theorem 1 in [**?**])**.** *Let $E/\mathbb{Q}$ be an elliptic curve possessing a $\mathbb{Q}$-rational p-isogeny. Then p belongs to the set*

$$\mathsf{IsogPrimeDeg}(\mathbb{Q}) := \{2,3,5,7,11,13,17,19,37,43,67,163\}.$$

It is worth stressing that, prior to this theorem, it was not even known that the set $\mathsf{IsogPrimeDeg}(\mathbb{Q})$ of primes p for which there exists an elliptic curve over $\mathbb{Q}$ possessing a $\mathbb{Q}$-rational p-isogeny was even finite!

Naturally one is motivated to determine the uniform set $\mathsf{IsogPrimeDeg}(K)$ of **isogeny primes for** K for other number fields. In general $\mathsf{IsogPrimeDeg}(K)$ may not be finite due to the possible existence of elliptic curves admitting complex multiplication over K. If such curves exist over K, then necessarily $\mathsf{IsogPrimeDeg}(K)$ is infinite, and it is a theorem of Momose [**?**, Theorem B] that, for quadratic number fields, this is the only way which $\mathsf{IsogPrimeDeg}(K)$ can be infinite. From the arithmetic theory of elliptic curves possessing complex multiplication, this means that, for K a quadratic number field, finiteness of $\mathsf{IsogPrimeDeg}(K)$ is equivalent to K not being one of the nine imaginary quadratic fields of class number one.

In our recent paper [**?**] we provide - assuming the Generalised Riemann Hypothesis - the first instances of the determination of $\mathsf{IsogPrimeDeg}(K)$ for $K \neq \mathbb{Q}$ since Mazur's work.

Theorem 1.2 ([**?**])**.** *Assuming GRH, we have the following.*

$$\begin{aligned}\mathsf{IsogPrimeDeg}(\mathbb{Q}(\sqrt{7})) &= \mathsf{IsogPrimeDeg}(\mathbb{Q})\\ \mathsf{IsogPrimeDeg}(\mathbb{Q}(\sqrt{-10})) &= \mathsf{IsogPrimeDeg}(\mathbb{Q})\\ \mathsf{IsogPrimeDeg}(\mathbb{Q}(\sqrt{5})) &= \mathsf{IsogPrimeDeg}(\mathbb{Q}) \cup \{23,47\}.\end{aligned}$$

This is actually obtained as a corollary of the following more general algorithmic result.

Algorithm 1.3. *Let K be a quadratic field which is not imaginary quadratic of class number 1. Then there is an algorithm which computes a superset of $\mathsf{IsogPrimeDeg}(K)$ as the union of three sets:*

$$\begin{aligned}\mathsf{IsogPrimeDeg}(K) \subseteq \mathsf{PreTypeOneTwoPrimes}(K) &\cup \mathsf{TypeOnePrimes}(K)\\ &\cup \mathsf{TypeTwoPrimes}(K).\end{aligned}$$

The termination of the algorithm relies on the Generalised Riemann Hypothesis.

It is the Sage [**?**] and PARI/GP [**?**] implementation of this algorithm that constitutes the software package *Quadratic Isogeny Primes*, and the present document is to be considered the *user's guide* for this package.

This document is organised as follows. In Section 2 we give an overview of the algorithm, which is based on studying the *isogeny types*. The cases of isogenies of **Type 1** and **Type 2** require separate handling; these constitute Section 4 and Section 5 respectively. Finally in Section 8 we illustrate the use of the package to determine $\mathsf{IsogPrimeDeg}(K)$ in the specific case of $K = \mathbb{Q}(\sqrt{5})$.

2. Overview of the algorithm

This section gives an overview of *Quadratic Isogeny Primes*. More details may be found in [**?**]. Since the phrase 'quadratic field which is not imaginary quadratic of class number one' arises frequently, we give these objects the name of

isogeny-finite quadratic fields, in light of Momose's theorem on the finiteness of $\mathsf{IsogPrimeDeg}(K)$ mentioned in the Introduction.

Let E/K be an elliptic curve over a number field possessing a K-rational p-isogeny for some prime $p \geq 5$; write V for the kernel of the isogeny, a one-dimensional G_K-stable module, where $G_K := \mathrm{Gal}(\overline{K}/K)$ denotes the absolute Galois group of K, and denote by λ the *isogeny character*:

$$\lambda : G_K \longrightarrow \mathrm{Aut}\, V(\overline{K}) \cong \mathbb{F}_p^\times.$$

The study of the isogeny character was initiated by Mazur [?, Section 5], and developed by Momose, who provided the following classification, in which θ_p denotes the mod-p cyclotomic character of G_K.

Theorem 2.1 (Momose, Theorem 1 in [?]). *Let K be a number field. Then there exists an effective constant $C_0 = C_0(K)$ such that for any prime $p > C_0$, and for any elliptic curve admitting a K-rational p-isogeny, the isogeny character λ falls into one of the following three types:*

Type 1. λ^{12} *or* $(\lambda\theta_p^{-1})^{12}$ *is unramified.*

Type 2. $\lambda^{12} = \theta_p^6$ *and* $p \equiv 3 \pmod 4$.

Type 3. *K contains the Hilbert class field H_L of an imaginary quadratic field L. The rational prime p splits in L:*

$$p\mathcal{O}_L = \mathfrak{p}\bar{\mathfrak{p}}.$$

For any prime $\mathfrak{q}$ of K prime to $\mathfrak{p}$, with Frobenius automorphism $\sigma_\mathfrak{q}$,

$$\lambda^{12}(\sigma_\mathfrak{q}) = \alpha^{12} \pmod{\mathfrak{p}}$$

for any $\alpha \in K^\times$ with $\alpha\mathcal{O}_L = \mathrm{Nm}_{K/L}(\mathfrak{q})$.

Therefore, for K an isogeny-finite quadratic field, Type 3 does not arise, so Momose's theorem may be reinterpreted in this case as saying that, outside of a finite set $\mathsf{PreTypeOneTwoPrimes}(K)$, the isogeny character must be of Type 1 or Type 2. While Momose did not make his constant C_0 effective, our algorithm constructs a tight superset of $\mathsf{PreTypeOneTwoPrimes}(K)$, which the rest of this section illustrates.

The first step towards the proof of Momose's theorem is a description of how λ^{12} acts on ideals of K coprime to p (identifying λ with a character of the ideal group $I_K(p)$).

Lemma 2.2 (Momose, Lemma 1 of *loc. cit.*). *Assume that K is Galois over $\mathbb{Q}$, and that p is unramified in K. Then for a fixed prime $\mathfrak{p}$ of K lying over p, there exist integers a_σ satisfying $0 \leq a_\sigma \leq 12$, for $\sigma \in \mathrm{Gal}(K/\mathbb{Q})$ such that*

$$\lambda^{12}((\alpha)) \equiv \alpha^\epsilon \pmod{\mathfrak{p}}$$

for $\epsilon = \sum_\sigma a_\sigma \sigma$ and $\alpha \in K^\times$ prime to p.

Thus, λ^{12} acts (modulo $\mathfrak{p}$) via an element ϵ of the group ring $\mathbb{Z}[\mathrm{Gal}(K/\mathbb{Q})]$, for which there are only finitely many possibilities. In our case of quadratic K, we may denote ϵ as a pair (a, b) of integers, referring to $a \cdot \mathrm{id} + b \cdot \sigma$, with σ being the non-trivial Galois automorphism.

Lemma 2.3. *The possible values of the group ring character ϵ for a quadratic field K are as follows:*

(1) **Quadratic** *ϵ: the* 4 *pairs* $(12a, 12b)$ *for* $a, b \in \{0, 1\}$*;*
(2) **Quartic** *ϵ: the* 12 *pairs* $(4a, 4b)$ *for* $a, b \in \{0, 1, 2, 3\}$*, excluding the* 4 *quadratic pairs.*
(3) **Sextic** *ϵ: the* 5 *pairs* $(6a, 6b)$ *for* $a, b \in \{0, 1, 2\}$*, excluding the* 4 *quadratic pairs;*

Proof. This follows from Remark 1 of *loc. cit.*, where the possible values for the unordered set $\{a, b\}$ are given as

- $\{0, 12\}$;
- $\{0, 4, 8, 12\}$ (possible only if $j(E) \equiv 0 \pmod{\mathfrak{p}}$ and $p \equiv 2 \pmod 3$);
- $\{0, 6, 12\}$ (possible only if $j(E) \equiv 1728 \pmod{\mathfrak{p}}$ and $p \equiv 3 \pmod 4$).

□

It follows from Lemma 2.2 that, if $\epsilon = (0, 0)$ or $(12, 12)$, then λ is of Type 1. Furthermore, Momose shows (Lemma 2 of *loc. cit.*) that, if $\epsilon = (6, 6)$, then λ is of Type 2. These two types will be dealt with separately in the subsequent two sections of this paper, so for the rest of this section we exclude them from consideration.

The 18 remaining possiblities for ϵs are implemented in a global Python dictionary, with keys corresponding to the ϵs, and values giving the type. Here is a snapshot:

```
EPSILONS_PRE_TYPE_1_2 = {

    (0,12): 'quadratic',
    (12,0): 'quadratic',

    (0,4): 'quartic',
    (0,8): 'quartic',
```

Keeping track of the type of ϵ allows us to impose further conditions on any possible isogeny primes that arise from the algorithm, due to the extra restrictions in the quartic and sextic types shown in the proof of Lemma 2.3:

```
def filter_ABC_primes(K, prime_list, eps_type):

    if eps_type == 'quadratic':
        # no additional restrictions
        return prime_list

    elif eps_type == 'quartic':
        # prime must split or ramify in K, and be congruent to 2 mod 3
        output_list = []

        for p in prime_list:
            if p%3 == 2:
                if not K.ideal(p).is_prime():
                    output_list.append(p)
        return output_list

    elif eps_type == 'sextic':
        # prime must split or ramify in K, and be congruent to 3 mod 4
        output_list = []
        for p in prime_list:
            if p%4 == 3:
                if not K.ideal(p).is_prime():
                    output_list.append(p)
        return output_list

    else:  # should never happen
        raise ValueError("type must be quadratic, quartic, or sextic")
```

The main idea for computing a tight superset of $\mathsf{PreTypeOneTwoPrimes}(K)$ is that of **auxiliary primes**; these are prime ideals $\mathfrak{q}$ of K of residue degree 1 lying over a rational prime q different to p, the assumed isogeny degree. In short, each auxiliary prime gives rise to a non-zero integer which p must divide; and by taking several auxiliary primes and taking the greatest common divisor of the resulting integers, one can significantly reduce the size of $\mathsf{PreTypeOneTwoPrimes}(K)$.

More precisely, we argue as follows. Let E/K be an elliptic curve over an isogeny-finite quadratic field K admitting a K-rational p-isogeny, where the isogeny character acts via the group ring character ϵ not of Type 1 or 2; and let $\mathfrak{q}$ be an auxiliary prime as above. Considering the reduction type of E at $\mathfrak{q}$, we know that either E has potentially multiplicative reduction at $\mathfrak{q}$, or it has potentially good reduction at $\mathfrak{q}$.

In the potentially multiplicative reduction case, it follows - essentially from Tate's algorithm - that p must divide one of the following two non-zero integers (non-zero because ϵ is not of Type 1):

$$A(\epsilon, \mathfrak{q}) := \mathrm{Nm}_{K/\mathbb{Q}}(\alpha^\epsilon - 1) \quad \text{or} \quad B(\epsilon, \mathfrak{q}) := \mathrm{Nm}_{K/\mathbb{Q}}(\alpha^\epsilon - q^{12h_K}),$$

where h_K denotes the class number of K, and α is a generator of the ideal $\mathfrak{q}^{h_K}$. We refer to primes in the support of these two integers as **Type A** and **Type B** primes, respectively. Observe that each of these integers depends only on ϵ and $\mathfrak{q}$.

In the potentially good reduction case, Momose shows (Lemma 2 of *loc. cit.*) that $\alpha^\epsilon \neq \beta^{12h_K}$ for any root β of the characteristic polynomial of Frobenius of any elliptic curve over $\mathbb{F}_q$; consequently p must divide the following non-zero integer depending only on ϵ and $\mathfrak{q}$:

$$C(\epsilon, \mathfrak{q}) := \mathrm{lcm}(\{\mathrm{Nm}_{K(\beta)/\mathbb{Q}}(\alpha^\epsilon - \beta^{12h_K}) \mid \beta \text{ is a Frobenius root over } \mathbb{F}_q\}).$$

We refer to primes dividing this integer as **Type C** primes. Note that this LCM is being taken over all characteristic polynomials of Frobenius for elliptic curves over $\mathbb{F}_q$.

We write $ABC(\epsilon, \mathfrak{q})$ for the union of the Type A, B, and C primes arising as above, together with the rational prime q lying under $\mathfrak{q}$; we append this single rational prime since the possible isogeny primes of type A, B or C were deemed to be distinct from q. For Aux a finite set of auxiliary primes, we may thus define the set of **pre-Type 1 or 2** primes for K as

$$\mathsf{PreTypeOneTwoPrimes}(K) := \bigcup_{\epsilon} \bigcap_{\mathfrak{q} \in \mathsf{Aux}} ABC(\epsilon, \mathfrak{q}),$$

where the union is taken over the 18 pairs in Lemma 2.3 excluding $(0, 12)$, $(12, 0)$, and $(6, 6)$.

3. PreTypeOneTwoPrimes(K)

This section makes some remarks on the Sage implementation of PreTypeOneTwoPrimes(K), which is carried out by the function `get_pre_type_one_two_primes`. The requested number of auxiliary primes is taken, which in the imaginary quadratic case are further checked to be non-principal ideals:

```
def get_pre_type_one_two_primes(K, aux_prime_count=3, loop_curves=False):

    if K.is_totally_real():
        aux_primes = K.primes_of_degree_one_list(aux_prime_count)
    else:
        it = K.primes_of_degree_one_iter()
        aux_primes = []
        while len(aux_primes) < aux_prime_count:
            aux_prime_candidate = next(it)
            if not aux_prime_candidate.is_principal():
                aux_primes.append(aux_prime_candidate)
```

The main part of the function is the following block, which computes, for each auxiliary prime $\mathfrak{q}$, a dictionary, whose keys are the 18 εs, and the value for each ε is the integer $ABC(\varepsilon, \mathfrak{q})$ defined just before **??**, i.e., the lowest common multiple of the four integers $A(\varepsilon, \mathfrak{q})$, $B(\varepsilon, \mathfrak{q})$, $C(\varepsilon, \mathfrak{q})$, and q.

```
for q in aux_primes:
    q_class_group_order = C_K(q).multiplicative_order()
    # these will be dicts with keys the epsilons, values sets of primes
    AB_primes_dict = get_AB_primes(K,q,epsilons, q_class_group_order)
    C_primes_dict = get_C_primes(K, q, epsilons, q_class_group_order, loop_curve
    unified_dict = {}
    q_rat = Integer(q.norm())
    assert q_rat.is_prime()
    for eps in epsilons:
        unified_dict[eps] = lcm([q_rat, AB_primes_dict[eps], C_primes_dict[eps]]
    tracking_dict[q] = unified_dict
```

The computation of the integers $A(\varepsilon, \mathfrak{q})$, $B(\varepsilon, \mathfrak{q})$, and $C(\varepsilon, \mathfrak{q})$ are delegated to the inner methods `get_AB_primes` and `get_C_primes`.

The first of these methods is as follows:

```
def get_AB_primes(K, q, epsilons, q_class_group_order):

    output_dict_AB = {}
    alphas = (q ** q_class_group_order).gens_reduced()
    assert len(alphas) == 1, "q^q_class_group_order not principal, which is very bad"
    alpha = alphas[0]
    rat_q = ZZ(q.norm())
    assert rat_q.is_prime(), "somehow the degree 1 prime is not prime"
    for eps in epsilons:
        alpha_to_eps = group_ring_exp(alpha,eps)
        A = (alpha_to_eps - 1).norm()
        B = (alpha_to_eps - (rat_q ** (12 * q_class_group_order))).norm()
        output_dict_AB[eps] = lcm(A,B)
    return output_dict_AB
```

get_C_primes requires one to loop over all possible characteristic polynomials of Frobenius of elliptic curves defined over residue_field. This set of polynomials is contained in the set of **Weil polynomials** of degree 2 whose complex roots have absolute value equal to the square-root of the cardinality of residue_field. Such polynomials are implemented in Sage as weil_polynomials, and as such, obtaining this potentially larger set of polynomials is faster than looping over all elliptic curves over residue_field and computing the frobenius polynomials. The boolean flag loop_curves switches between these two sets of polynomials.

```
if loop_curves:
    frob_polys_to_loop = get_weil_polys(residue_field)
else:
    frob_polys_to_loop = R.weil_polynomials(2, residue_field.cardinality())
```

While looping through all polynomials in frob_polys_to_loop, the possible roots β are extracted, and for each β and ε, the quantity $\mathrm{Nm}_{K(\beta)/\mathbb{Q}}(\alpha^{\varepsilon} - \beta^{12\,\mathrm{ord}_{C_K}(\mathfrak{q})})$ is computed and stored in a "growing LCM" integer which itself is ultimately stored in a dictionary with key ε:

```
if frob_poly.is_irreducible():
    frob_poly_root_field = frob_poly.root_field('a')
    _, K_into_KL, L_into_KL, _ = K.composite_fields(frob_poly_root_field, 'c',
                                 both_maps=True)[0]
else:
    frob_poly_root_field = IntegerRing()
roots_of_frob = frob_poly.roots(frob_poly_root_field)
betas = [r for r,e in roots_of_frob]

for beta in betas:
    if beta in K:
        for eps in epsilons:
            N = (group_ring_exp(alpha, eps)
                    - beta ** (12*q_class_group_order)).absolute_norm()
            N = ZZ(N)
            output_dict_C[eps] = lcm(output_dict_C[eps], N)
    else:
        for eps in epsilons:
            N = (K_into_KL(group_ring_exp(alpha, eps))
                    - L_into_KL(beta ** (12*q_class_group_order))).absolute_norm
            N = ZZ(N)
            output_dict_C[eps] = lcm(output_dict_C[eps], N)
return output_dict_C
```

This allows one to only compute factorisations of Weil polynomials of all elliptic curves over $\kappa(\mathfrak{q})$ once for each $\mathfrak{q}$, rather than every time for each ε.

The dictionary thus created is collapsed and inverted to yield a dictionary with keys the εs and values the GCDs, taken over all $\mathfrak{q} \in \mathsf{Aux}$, of the integers $ABC(\varepsilon, \mathfrak{q})$:

```
    tracking_dict_inv_collapsed = {}
    for eps in epsilons:
        q_dict = {}
        for q in aux_primes:
            q_dict[q] = tracking_dict[q][eps]
        q_dict_collapsed = gcd(list(q_dict.values()))
        tracking_dict_inv_collapsed[eps] = q_dict_collapsed
```

The function then ends by taking the prime divisors of the integers thus far obtained, passing them through the `filter_ABC_primes` method shown above, and finally taking the union over all declared εs, completing the determination of $\mathsf{PreTypeTwoPrimes}$:

```
    final_split_dict = {}

    for eps_type in set(epsilons.values()):
        eps_type_tracking_dict_inv = {eps:ZZ(tracking_dict_inv_collapsed[eps])
                                  for eps in epsilons if epsilons[eps] == eps_type}
        eps_type_output = lcm(list(eps_type_tracking_dict_inv.values()))
        eps_type_output = eps_type_output.prime_divisors()
        eps_type_output = filter_ABC_primes(K, eps_type_output, eps_type)
        final_split_dict[eps_type] = set(eps_type_output)

    output = set.union(*(val for val in final_split_dict.values()))
    output = list(output)
    output.sort()
    return output
```

4. $\mathsf{TypeOnePrimes}(K)$

The determination of $\mathsf{TypeOnePrimes}$ is very similar to $\mathsf{PreTypeOneTwoPrimes}$: one uses auxiliary primes $\mathfrak{q}$ to compute non-zero integers which an isogeny prime must divide, and one takes the GCD of the resulting integers. More precisely, for an elliptic curve E/K over an isogeny-finite quadratic field admitting a K-rational p-isogeny of Type 1, as before one considers the two types of potential reduction E may have at $\mathfrak{q}$.

If E has potentially multiplicative reduction at $\mathfrak{q}$, then Momose shows that $p-1$ divides $12h_K$.

If E has potentially good reduction at $\mathfrak{q}$, then Momose shows that p must divide the non-zero integer:

$$D(\mathfrak{q}) := \operatorname{lcm}(\{1 + N(\mathfrak{q})^{12h_K} - \beta^{12h_K} - \bar{\beta}^{12h_K} \mid \beta \text{ is a Frobenius root over } \kappa(\mathfrak{q})\}). \tag{4.1}$$

This however makes two assumptions

(1) The possible isogeny prime p is unramified in K.
(2) A certain morphism

$$f : X^{(2)}_{/S} \to \tilde{J}_{/S}$$

from the symmetric square of the modular curve $X_0(p)$ to the Eisenstein quotient of $J_0(p)$ with base $S = \operatorname{Spec}\mathbb{Z}[1/p]$ is a *formal immersion along* (∞, ∞) away from characteristics 2, 3, and 5.

We do not discuss the notion of formal immersion here; it suffices to say that Kamienny proved [**?**, Proposition 3.2] that the second assumption above is satisfied whenever $p \geq 73$. The first assumption gives the restriction that the possible isogeny primes are deemed not to divide the discriminant Δ_K of K. One therefore has the following superset for the Type 1 primes (again, for Aux a finite set of auxiliary primes):

$$\begin{aligned}\mathsf{TypeOnePrimes}(K) = \mathsf{PrimesUpTo}(71) &\cup \{p : p \mid \Delta_K\} \\ &\cup \{p : (p-1) \mid 12h_K\} \cup \left(\bigcap_{\mathfrak{q} \in \mathsf{Aux}} \{p : p \mid qD(\mathfrak{q})\} \right).\end{aligned}$$

The function which computes the Type 1 primes begins as follows:

```
P_2 = 73
Q_2 = 7
def get_type_1_primes(K, aux_prime_count=3, loop_curves=False):
    """Compute the type 1 primes"""

    h_K = K.class_number()
    C_K = K.class_group()
    aux_primes = [Q_2]
    prime_to_append = Q_2
    for _ in range(1,aux_prime_count):
        prime_to_append = next_prime(prime_to_append)
        aux_primes.append(prime_to_append)
```

`aux_prime_count` is a parameter denoting the number of auxiliary primes to take, and `Q_2` is the global value of 7 denoting what Momose calls $q_{(2)}$; the code thus far takes `aux_prime_count` number of rational primes larger than or equal to 7. `loop_curves` is a boolean parameter to be described below.

We initialise a dictionary to store the integers $D(\mathfrak{q})$ for each q, and then begin the loop over all such auxiliary primes, initially computing various entities associated with the residue field and order of $\mathfrak{q}$ in the class group:

```
    D_dict = {}
    R = PolynomialRing(Rationals(), 'x')

    for q in aux_primes:
        frak_q = K.primes_above(q)[0]
        residue_field = frak_q.residue_field(names='z')
        residue_field_card = residue_field.cardinality()
        frak_q_class_group_order = C_K(frak_q).multiplicative_order()
        exponent = 12 * frak_q_class_group_order
```

At this point we need to loop over all possible characteristic polynomials of Frobenius of elliptic curves defined over `residue_field`. This set of polynomials is contained in the set of **Weil polynomials** of degree 2 whose complex roots have absolute value equal to the square-root of the cardinality of `residue_field`. Such polynomials are implemented in Sage as `weil_polynomials`, and as such, obtaining this potentially larger set of polynomials is faster than looping over all elliptic curves over `residue_field` and computing the frobenius polynomials. The boolean flag `loop_curves` switches between these two sets of polynomials.

```
running_D = q
if loop_curves:
    weil_polys = get_weil_polys(residue_field)
else:
    weil_polys = R.weil_polynomials(2, residue_field_card)

for wp in weil_polys:
    D = get_D(wp, residue_field_card, exponent)
    D = Integer(D)
    if D != 0:
        # else we can ignore since it doesn't arise from an elliptic curve
        running_D = lcm(running_D, D)
D_dict[q] = running_D
```

D_dict[q] is then a dictionary with keys the auxiliary primes $\mathfrak{q}$, and values the integers $D(\mathfrak{q})$ defined in Equation (4.1).

The computation of $D(\mathfrak{q})$ then proceeds via computing the roots of the characteristic polynomial of Frobenius:

```
def get_D(frob_poly, residue_field_card, exponent):
    """This computes the integer D"""

    if frob_poly.is_irreducible():
        frob_poly_root_field = frob_poly.root_field('a')
    else:
        frob_poly_root_field = IntegerRing()
    roots_of_frob = frob_poly.roots(frob_poly_root_field)
    if len(roots_of_frob) == 1:
        assert roots_of_frob[0][1] == 2
        beta = roots_of_frob[0][0]
        return 1 + residue_field_card ** exponent - 2 * beta ** exponent
    else:
        beta, beta_bar = [r for r,e in roots_of_frob]
        return 1 + residue_field_card ** exponent - beta ** exponent - beta_bar
```

The code for this section then concludes by taking the greatest common divisor of $D(\mathfrak{q})$ for the various $\mathfrak{q}$ the program considered, finding the prime divisors of this GCD, and appending the other sets of primes shown in **??**:

```
    output = gcd(list(D_dict.values()))
    output = set(output.prime_divisors())
    output = output.union(set(prime_range(P_2)))
    Delta_K = K.discriminant().abs()
    output = output.union(set(Delta_K.prime_divisors()))
    third_set = [1+d for d in (12*h_K).divisors()]  # p : (p-1)|12h_K
    output = output.union(set([p for p in third_set if p.is_prime()]))
    output = list(output)
    output.sort()
    return output
```

5. TypeTwoPrimes(K)

Momose gives a necessary condition that an isogeny prime of Type 2 must satisfy, which in [?] is expressed as follows.

Condition CC. Let K be an isogeny-finite quadratic field, and E/K an elliptic curve admitting a K-rational p-isogeny, with p of Type 2. Let q be a rational prime $< p/4$ such that $q^2 + q + 1 \not\equiv 0 \pmod{p}$. Then the following implication holds:

if q splits or ramifies in K, then q does not split in $\mathbb{Q}(\sqrt{-p})$.

This condition may be expressed concretely via the legendre symbols $\left(\frac{D}{q}\right)$ and $\left(\frac{q}{p}\right)$, so may be checked for any prime p as follows:

```
def satisfies_condition_CC(K,p):
    for q in prime_range(p/4):
        if (q**2 + q + 1) % p != 0:
            if not K.ideal(q).is_prime():
                if legendre_symbol(q,p) == 1:  # i.e. not inert
                    return False
    return True
```

The question is then of how far we need to check: can we find an upper bound on Type 2 primes?

It was Larson and Vaintrob who found an effective bound on Type 2 primes assuming GRH, and modulo the determination of an effectively computable absolute constant c_7 (which alas is not effectively computed!), which appears in [?, Corollary 6.3]. Tracing through their proof, and combining it with the best possible bounds in the Effective Chebotarev Density Theorem due to Bach and Sorenson [?, Theorem 5.1], we are able to offer a modest improvement on their bound which removes the dependence on c_7 in our case.

Proposition 5.1. *Assume GRH. Let K be an isogeny-finite quadratic field, and E/K an elliptic curve possessing a K-rational p-isogeny, for p a Type 2 prime. Then p satisfies*

$$p \leq (16 \log p + 16 \log(12\Delta_K) + 26)^4.$$

In particular, there are only finitely many primes p as above.

See [?, Proposition 4.4] for the proof. For $K = \mathbb{Q}(\sqrt{5})$, this bound on Type 2 primes is approximately 5.65×10^{10}. The algorithm therefore needs to check all primes up to this large bound.

The Sage script has implemented this check; but we found that Sage quickly ran into memory overflow errors when attempting this check up to the bound given by the above proposition. We therefore limited the Sage script to only check Type 2 primes up to 1000, and instead developed an optimised PARI/GP script with parallel threading to carry out the check up to the Type 2 bound. The search range is broken into blocks of size 100,000; the primes in these blocks are then checked to satisfy condition CC via a *parallel for loop*:

```
blockSize=100000;
export(blockSize)

checktypetwo(pBeg) =
{
    my(p,cond);
    forprime(p = pBeg*blockSize, (pBeg+1)*blockSize-1,
             cond=custom_congruence_condition(p,D);
             if(cond,print_satisfiesCC(p)));
}
export(checktypetwo)
```

```
howMany=floor(typetwobound/blockSize);

parapply(checktypetwo,[0..howMany]);
```

Remark 5.2. The largest Type 2 prime we have encountered for any isogeny-finite quadratic field is 163. There is a connection between Type 2 primes and an anaglogue of the class number one problem, as discussed in Goldfeld's Appendix to Mazur's paper [**?**].

We may then check all primes up to the Type 2 bound by running the following with `bound=None`:

```
def get_type_2_primes(K, bound=None):
    """Compute a list containing the type 2 primes"""

    # First get the bound
    if bound is None:
        bound = get_type_2_bound(K)
        print("type_2_bound = {}".format(bound))

    # We need to include all primes up to 25
    # see Larson/Vaintrob's proof of Theorem 6.4
    output = set(prime_range(25))

    for p in pari.primes(25, bound):
        p_int = Integer(p)
        if p_int % 4 == 3:  # Type 2 primes necessarily congruent to 3 mod 4
            if satisfies_condition_CC(K,p_int):
                output.add(p_int)
    return output
```

However, on our version of Sage, executing this with `bound=None` quickly yields a PARI memory error. The issue is precomputing very large sets of prime numbers, which a typical architecture may not be able to handle.

We therefore translate the above routing directly into PARI/GP code. The following shows the code in the case of $K = \mathbb{Q}(\sqrt{-5})$; the other cases are similar.

```
D=-5;  \\ change this to desired value
typetwobound=56546719183;  \\ change this to corresponding bound
export(D)

\\check if condition CC is satisfied
satisfiesCC(p) =
{
  forprime(q = 7,p/4,
    if((q^2 + q + 1) % p != 0,
      if(kronecker(D,q) != -1,
        \\ don't memoize the next call
        if(kronecker(q,p) == 1,
          return(0)
        );
      );
    );
  );
  return(1);
}
export(satisfiesCC)

\\print to stdout if p satisfies condition CC
print_satisfiesCC(p) =
{
  if(satisfiesCC(p),
    print(p," is a type 2 prime")
  );
}
export(print_satisfiesCC)

\\ for D=-6,-5,6,10
congruence_condition_main(p) =
{
    if(p%24 == 19,
      x=p%5;
      if((x==2)||(x==3),
        return(1);
      return(0);
      );
    return(0);
    );
}
export(congruence_condition_main)

blockSize=100000;
export(blockSize)

checktypetwo(pBeg) =
{
    my(p,cond);
    forprime(p = pBeg*blockSize, (pBeg+1)*blockSize-1,
              cond=custom_congruence_condition(p,D);
              if(cond,print_satisfiesCC(p)));
}
export(checktypetwo)

howMany=floor(typetwobound/blockSize);

parapply(checktypetwo,[0..howMany]);
```

Note that here we have computed by hand the Kronecker symbol $\left(\frac{D}{q}\right)$ for $q = 2, 3, 5$; this allows us to start the q loop at 7, and to restrict to primes in certain congruence classes modulo 24 and 5[1]. The other cases of K are similarly optimised.

The above code checks all primes in blocks of size 100,000; this is to mitigate the overhead required in the communication between the primary and secondary threads (see [?, Section 2.4] for more on this overhead issue).

We initialise PARI/GP to precompute the requisite number of primes:

[1]every little helps!

```
gp --primelimit 56546719183 --stacksize 20000000
```

Running this PARI/GP code revealed that 163 was the largest Type 2 prime witnessed for the 9 number fields shown in **??** in the Introduction.

Remark 5.3. The above code requires a version of PARI/GP which allows for parallel computing. Running it for `D=-5` took just over an hour of wall time with 8 threads running on an Intel Core i5 processor.

We remark that Joe Buhler also ran a computation several decades ago for Type 2 primes for imaginary quadratic fields, checking all primes up to $32,768$, as reported by Goldfeld in the Appendix to Mazur's paper. In the table in *loc. cit.*, as in our case, 163 was the largest prime observed for the fields not of class number one.

6. The DLMV bound

In this section we present the Sage code for calculating the DLMV bound for an isogeny-finite quadratic field K.

We begin by computing the bound on TypeTwoPrimes, using Sage's `find_root` function.

```
GENERIC_UPPER_BOUND = 10 ** 30
def get_type_2_bound(K):

    # The Bach and Sorenson parameters
    A = 4
    B = 2.5
    C = 5

    n_K = K.degree()
    delta_K = K.discriminant().abs()

    D = 2 * A * n_K
    E = 4 * A * log(delta_K) + 2 * A * n_K * log(12) + 4 * B * n_K + C + 1

    R = PolynomialRing(Rationals(), 'x')
    x = R.gen()
    f = x - (D*log(x) + E) ** 4

    try:
        bound = find_root(f,10,GENERIC_UPPER_BOUND)
        return ceil(bound)
    except RuntimeError:
        warning_msg = ("Warning: Type 2 bound for quadratic field with "
        "discriminant {} failed. Returning generic upper bound").format(delta_K)
        print(warning_msg)
        return GENERIC_UPPER_BOUND
```

This searches for a root of the function

$$p - (16 \log p + 16 \log(12\Delta_K) + 26)^4$$

in the range $[10, 10^{30}]$, and if it doesn't find one, returns a generic bound of 10^{30}; for all of our examples, a root was found in this interval.

Using this function, we obtain the following implementation of the DLMV bound. We make one minor improvement: David took $C(K, 2(\Delta_K^{Ah_K}))$. However, since we are in any case assuming GRH, we may replace the $2(\Delta_K^{Ah_K})$ with $(4\log|\Delta_K|^{h_K} + 5h_K + 5)^2$, using Theorem 5.1 of [?] (applied with $E = H_K$, the Hilbert class field of K.)

```
def DLMV(K):
    """Compute the DLMV bound"""

    # First compute David's C_0

    Delta_K = K.discriminant().abs()
    h_K = K.class_number()
    R_K = K.regulator()
    r_K = K.unit_group().rank()
    delta_K = log(2)/(r_K + 1)
    C_1_K = r_K ** (r_K + 1) * delta_K**(-(r_K - 1)) / 2
    C_2_K = exp(24 * C_1_K * R_K)
    CHEB_DEN_BOUND = (4*log(Delta_K**h_K) + 5*h_K + 5)**2
    C_0 = ((CHEB_DEN_BOUND**(12*h_K))*C_2_K + CHEB_DEN_BOUND**(6*h_K))**4

    # Now the Type 1 and 2 bounds

    type_1_bound = (1 + 3**(12 * h_K))**2
    type_2_bound = get_type_2_bound(K)

    return max(C_0, type_1_bound, type_2_bound)
```

7. Verification Testing

In order to have some testing of the code, we require examples of isogeny primes larger than 71 over specific quadratic fields. From the survey article [?], together with one example from Box's Section 4.7 [?], we have the six examples shown in Table 7.1

Prime	Is Isogeny Prime over	Found by
73	$\mathbb{Q}(\sqrt{-127})$	Galbraith [?]
73	$\mathbb{Q}(\sqrt{-31})$	Box [?]
103	$\mathbb{Q}(\sqrt{5 \cdot 577})$	Galbraith (*loc. cit.*)
137	$\mathbb{Q}(\sqrt{-31159})$	Galbraith (*loc. cit.*)
191	$\mathbb{Q}(\sqrt{61 \cdot 229 \cdot 145757})$	Elkies [?]
311	$\mathbb{Q}(\sqrt{11 \cdot 17 \cdot 9011 \cdot 23629})$	Galbraith (*loc. cit.*)

Table 7.1. Instances of isogeny primes in the literature which are rational over isogeny-finite quadratic fields.

These examples serve as the basis of a unit testing framework, which has been implemented in `test_quadratic_isogeny_primes.py`. Moreover, for each case, we check that $\mathsf{IsogPrimeDeg}(K)$ contains $\mathsf{IsogPrimeDeg}(\mathbb{Q})$. Here is a snapshot of the tests:

```
AUX_PRIME_COUNT = 2
class TestQuadraticIsogenyPrimes(unittest.TestCase):

    def test_73(self):
        K = QuadraticField(-127)
        superset = get_isogeny_primes(K, AUX_PRIME_COUNT)
        self.assertTrue(set(superset).issuperset(EC_Q_ISOGENY_PRIMES))
        self.assertIn(73, superset)

    def test_103(self):
        K = QuadraticField(5 * 577)
        superset = get_isogeny_primes(K, AUX_PRIME_COUNT)
        self.assertTrue(set(superset).issuperset(EC_Q_ISOGENY_PRIMES))
        self.assertIn(103, superset)

    def test_311(self):
        K = QuadraticField(11*17*9011*23629)
        superset = get_isogeny_primes(K, AUX_PRIME_COUNT)
        self.assertTrue(set(superset).issuperset(EC_Q_ISOGENY_PRIMES))
        self.assertIn(311, superset)
```

Running all tests takes about a minute on an old laptop.

8. An example: $\mathsf{IsogPrimeDeg}(\mathbb{Q}(\sqrt{5}))$

We illustrate how the *Quadratic Isogeny Primes* package may be used to exactly determine the set of isogeny primes $\mathsf{IsogPrimeDeg}(K)$ for a given isogeny-finite quadratic field. In this final section we show this for $K = \mathbb{Q}(\sqrt{5})$.

Having cloned the Git repository, the main file in the package is `quadratic_isogeny_primes.py`, which takes one required argument - the integer D for which $K = \mathbb{Q}(\sqrt{D})$ - and several optional arguments, including `--aux_prime_count`, which determines the number of auxiliary primes to take. Increasing this number will reduce the size of the final superset, but will take longer to run. In this example we take 6 auxiliary primes:

```
sage quadratic_isogeny_primes.py 5 --aux_prime_count 6
```

Running this on an old laptop takes about 20 seconds, and shows that the superset for $\mathsf{IsogPrimeDeg}(K)$ is the following:

```
superset = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 163]
```

Note that we are also warned about the following:

```
WARNING: Only checking Type 2 primes up to 1000.
```

As explained in Section 5, Sage encounters a memory error when attempting to check all primes up to the bound on Type 2 primes, which may be attempted by running the script with the option `--rigorous`; this results in the following (truncated) error:

```
type_2_bound = 56546719183
Traceback (most recent call last):
...
cypari2.handle_error.PariError: the PARI stack overflows (current size: 1000000;
 maximum size: 2596085760)
```

To check all primes up to this bound of 56546719183, we navigate to the `gp_scripts` folder, and edit the file `partype2primes.gp` for our desired D and the Type 2 bound:

```
D=5;  \\ change this to desired value
typetwobound=56546719183;
export(D)
```

We initialise PARI/GP with the requisite number of precomputed primes:

```
gp --primelimit 56546719183
```

The user may wish to confirm that their version of PARI/GP has been configured for parallel computing; if so, the number of available threads is declared upon starting the PARI/GP calculator:

```
parisize = 400003072, primelimit = 56546719183, nbthreads = 8
```

One then loads the script `partype2primes.gp` and executes the main 'parallel for loop', which checks all primes up to 56546719183 for whether they are indeed of Type 2; if so, they are printed on screen. Doing this for `D = 5` took just under 70 minutes on an old laptop:

```
? read("partype2primes.gp");
? parapply(checktypetwo,[0..howMany]);
3 is a type 2 prime
7 is a type 2 prime
23 is a type 2 prime
43 is a type 2 prime
47 is a type 2 prime
67 is a type 2 prime
163 is a type 2 prime
cpu time = 9h, 18min, 26,461 ms, real time = 1h, 9min, 55,918 ms.
?
```

Therefore, conditional upon the Generalised Riemann Hypothesis, the superset found by `quadratic_isogeny_primes.py` is indeed a superset, and no Type 2 primes larger than 1000 need to be considered. Since an elliptic curve over $\mathbb{Q}$ may be considered as an elliptic curve over any number field K, we have $\mathsf{IsogPrimeDeg}(\mathbb{Q}) \subseteq \mathsf{IsogPrimeDeg}(K)$, so we need to decide whether or not the following are isogeny primes for $\mathbb{Q}(\sqrt{5})$:

$$\{23, 29, 31, 41, 47, 53, 59, 61, 71, 73, 79\}.$$

This is equivalent to asking, for each p in this list, whether or not the modular curve $X_0(p)$ admits a non-cuspidal $\mathbb{Q}(\sqrt{5})$-rational point.

Such questions are notoriously difficult; but fortunately, many of these ps we need to consider are such that the genus of $X_0(p)$ is fairly small; and most importantly, there has been in recent years much progress in the study of *quadratic points on low-genus modular curves*. Of particular significance are the works of Bruin and Najman [?], Özman and Siksek [?], and most recently Box [?]; taken together, these three works give a complete determination of the quadratic points on $X_0(N)$ when it has genus 2, 3, 4 or 5. In essence, for each such N, there are only finitely many quadratic points which do not correspond to elliptic $\mathbb{Q}$-curves; and these finitely many points are determined explicitly. All of these works utilise Magma [?] in a significant way.

By combining these results with earlier work of Özman in determining local solubility of quadratic twists of $X_0(N)$ [?], together with a fair amount of Sage and Magma computation - notably the Chabauty package - one is able to decide that, out of the primes in the above set, only the primes 23 and 47 are isogeny primes for $\mathbb{Q}(\sqrt{5})$, thereby proving the final assertion of Theorem 1.2. For full details of this, the reader is referred to the final section - *Weeding out the Pretenders* - of [?].

Barinder S. Banwait, Harish-Chandra Research Institute, Prayagraj, India
Email address: barinder.s.banwait@gmail.com