```
load "hu.m";

function IdealConjugate(I, K)
// Input: Z_K-Ideal I; Field K

// Output: Z_K-Ideal which is the complex conugate of I

    gens := [];
    for g in Generators(I) do
        Append(~gens, ComplexConjugate(K ! g));
    end for;

    return ideal<Integers(K)|gens>;

end function;


function ReduceByIsometry(Lattices)
// Input: List of lattices

// Output: Reduced list for which the elements are
pairwise non-isometric

    LatticesReduced := [* *];
    Minima := [* *];
    NumShortest := AssociativeArray();
    SizeAuto := AssociativeArray();


    for i in [1..#Lattices] do
        L := Lattices[i];

        min_computed := false;
        minimum := 0;

        shortest_computed := false;
        shortest := 0;

        auto_computed := false;
        auto := 0;

        for j in [1..#LatticesReduced] do
            M := LatticesReduced[j];

            if not min_computed then
                min_computed := true;
                minimum := Min(L);
```

```
            end if;

            if not IsDefined(Minima, j) then
                Minima[j] := Min(M);
            end if;

            if minimum ne Minima[j] then
                continue;
            end if;


            if not shortest_computed then
                shortest_computed := true;
                shortest := #ShortestVectors(L);
            end if;

            if not IsDefined(NumShortest, j) then
                NumShortest[j] := #ShortestVectors(M);
            end if;

            if shortest ne NumShortest[j] then
                continue;
            end if;


            if not auto_computed then
                auto_computed := true;
                auto := #AutomorphismGroup(L);
            end if;

            if not IsDefined(SizeAuto, j) then
                SizeAuto[j] := #AutomorphismGroup(M);
            end if;

            if auto ne SizeAuto[j] then
                continue;
            end if;


            if IsIsometric(L, M) then
                continue i;
            end if;
        end for;

        Append(~LatticesReduced, Lattices[i]);

        NewIndex := #LatticesReduced;
```

```
        if min_computed then
            Minima[NewIndex] := minimum;
        end if;

        if shortest_computed then
            NumShortest[NewIndex] := shortest;
        end if;

        if auto_computed then
            SizeAuto[NewIndex] := auto;
        end if;

    end for;

    return LatticesReduced;
end function;


function ExtremalMinimum(l, n)
// Input: Square-free l in N; n in N

// Output: Minimum that a l-modular lattice of
dimension n must have at least

    if l eq 1 then k := 24;
    elif l eq 2 then k := 16;
    elif l eq 3 then k := 12;
    elif l eq 5 then k := 8;
    elif l eq 6 then k := 8;
    elif l eq 7 then k := 6;
    elif l eq 11 then k := 4;
    elif l eq 14 then k := 4;
    elif l eq 15 then k := 4;
    elif l eq 23 then k := 2;
    end if;

    return 2 + 2*Floor(n/k);
end function;


HermiteBounds := [1, 1.1547, 1.2599, 1.1412, 1.5157,
1.6654, 1.8115, 2, 2.1327, 2.2637, 2.3934, 2.5218,
2.6494, 2.7759, 2.9015, 3.0264, 3.1507, 3.2744,
3.3975, 3.5201, 3.6423, 3.7641, 3.8855, 4.0067,
4.1275, 4.2481, 4.3685, 4.4887, 4.6087, 4.7286,
4.8484, 4.9681, 5.0877, 5.2072, 5.3267, 5.4462];


```

```
136 function GenSymbol(L)
137 // Input: Positive definite Numberfield Lattice L of square-free level
138
139 // Output: Genus symbol of L in the form [S_1, n, <2, n_2, epsilon_2, S_2, t_2>, <3, n_3, epsilon_3>, <5,...>, ...] for all primes dividing Det(L)
140     Symbol := [* *];
141
142     Rat := RationalsAsNumberField();
143     Int := Integers(Rat);
144
145     LNF := NumberFieldLatticeWithGram(Matrix(Rat, GramMatrix(L)));
146     _, Grams2, Vals2 := JordanDecomposition (LNF,ideal<Int|2>);
147
148     // Checks if all diagonal entries of the 1-component of the 2-adic jordan decomposition are even
149     if Vals2[1] ne 0 or (Vals2[1] eq 0 and &and ([Valuation(Rationals() ! (Grams2[1][i][i]), 2) ge 1 : i in [1..NumberOfRows(Grams2[1])]])) then
150         Append(~Symbol, 2);
151     else
152         Append(~Symbol, 1);
153     end if;
154
155     Append(~Symbol, Dimension(L));
156
157
158     for p in PrimeDivisors(Integers() ! (Determinant (L))) do
159         _, Gramsp, Valsp := JordanDecomposition(LNF, ideal<Int|p>);
160
161         if Valsp[1] eq 0 then
162             G := Matrix(Rationals(), 1/p * Gramsp[2]);
163         else
164             G := Matrix(Rationals(), 1/p * Gramsp[1]);
165         end if;
166
167         sym := <p, NumberOfRows(G)>;
168
169         det := Determinant(G);
170         det := Integers() ! (det * Denominator(det)^2);
```

```
171
172         if p eq 2 then
173             if IsDivisibleBy(det+1, 8) or IsDivisibleBy
    (det-1, 8) then
174                 Append(~sym, 1);
175             else
176                 Append(~sym, -1);
177             end if;
178
179             if &and([Valuation(Rationals() ! (G[i]
    [i]), 2) ge 1 : i in [1..sym[2]]]) then
180                 Append(~sym, 2);
181             else
182                 Append(~sym, 1);
183             end if;
184
185             if sym[4] eq 2 then
186                 Append(~sym, 0);
187             else
188                 Append(~sym, Integers() ! (Trace(G)*
    Denominator(Trace(G))^2) mod 8);
189             end if;
190         else
191             Append(~sym, LegendreSymbol(det, p));
192         end if;
193
194         Append(~Symbol, sym);
195     end for;
196
197     return Symbol;
198
199 end function;
200
201
202 function ToZLattice(L)
203 // Input: Numberfield lattice L
204
205 // Output: L as Z-lattice
206     B:= Matrix(ZBasis(L`Module));
207     G:= B * L`Form * InternalConjugate(L, B);
208     Form:= Matrix( Ncols(G), [ AbsoluteTrace(e) : e in
    Eltseq(G) ] );
209     Form:=IntegralMatrix(Form);
210
211     LZ := LatticeWithGram(LLLGram(Matrix(Integers
    (),Form)));
212
213     return LZ;
```

```
214 end function;
215
216
217 function MiPoQuotient(sigma, L, p);
218 // Input : Automorphism sigma of L; Lattice L
219
220 // Output: Minimal polynomial of the operation of
    sigma on the partial dual quotient L^(#, p) / L
221
222     sigma := Matrix(Rationals(), sigma);
223     L := CoordinateLattice(L);
224     LD := PartialDual(L, p : Rescale := false);
225     _,phi := LD / L;
226     MiPo := PolynomialRing(GF(p)) ! 1;
227
228     B := [];
229
230     for i in [1..Rank(LD)] do
231
232         b := LD.i;
233         if b in sub<LD|L,B> then
234             continue;
235         end if;
236         Append(~B,b);
237
238         dep := false;
239         C := [Eltseq(phi(b))];
240         while not dep do
241             b := b*sigma;
242             Append(~C, Eltseq(phi(b)));
243             Mat := Matrix(GF(p),C);
244             if Dimension(Kernel(Mat)) gt 0 then
245                 dep := true;
246                 coeff := Basis(Kernel(Mat))[1];
247                 coeff /:= coeff[#C];
248                 coeff := Eltseq(coeff);
249                 MiPo := LCM(MiPo, Polynomial(GF(p),
    coeff));
250             else
251                 Append(~B, b);
252             end if;
253         end while;
254     end for;
255
256     return MiPo;
257
258 end function;
259
```

```
function IsModular(L, l)
// Input: Lattice L; l in N

// Output: true iff L is a l-modular lattice

    return IsIsometric(L, LatticeWithGram(l*GramMatrix
(Dual(L:Rescale:=false))));

end function;

function IsStronglyModular(L,l)
// Input: Lattice L; l in N

// Output: true iff L is a strongly l-modular lattice

    return &and[IsIsometric(L, LatticeWithGram
(m*GramMatrix(PartialDual(L, m : Rescale:=false)))) :
m in [m : m in Divisors(l) | Gcd(m, Integers() ! (l/
m)) eq 1]];

end function;
```