

```

1  load "hu.m"; // Program by David Lorch for constructing lattices
   with given elementary divisors
2
3  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];
4
5
6  function IdealConjugate(I, K)
7  // Input: Z_K-Ideal I; Field K
8
9  // Output: Z_K-Ideal which is the complex conjugate of I
10
11  gens := [];
12  for g in Generators(I) do
13    Append(~gens, ComplexConjugate(K ! g));
14  end for;
15
16  return ideal<Integers(K)|gens>;
17
18 end function;
19
20
21 function ReduceByIsometry(Lattices)
22 // Input: List of lattices
23
24 // Output: Reduced list for which the elements are pairwise non-
   isometric
25
26 LatticesReduced := [* *];
27 Minima := [* *];
28 NumShortest := AssociativeArray();
29 SizeAuto := AssociativeArray();
30
31
32 for i in [1..#Lattices] do
33   L := Lattices[i];
34
35   min_computed := false;
36   minimum := 0;
37
38   shortest_computed := false;
39   shortest := 0;
40
41   auto_computed := false;
42   auto := 0;
43
44   for j in [1..#LatticesReduced] do
45     M := LatticesReduced[j];
46
47     if not min_computed then
48       min_computed := true;
49       minimum := Min(L);

```

```

50     end if;
51
52     if not IsDefined(Minima, j) then
53         Minima[j] := Min(M);
54     end if;
55
56     if minimum ne Minima[j] then
57         continue;
58     end if;
59
60
61     if not shortest_computed then
62         shortest_computed := true;
63         shortest := #ShortestVectors(L);
64     end if;
65
66     if not IsDefined(NumShortest, j) then
67         NumShortest[j] := #ShortestVectors(M);
68     end if;
69
70     if shortest ne NumShortest[j] then
71         continue;
72     end if;
73
74
75     if not auto_computed then
76         auto_computed := true;
77         auto := #AutomorphismGroup(L);
78     end if;
79
80     if not IsDefined(SizeAuto, j) then
81         SizeAuto[j] := #AutomorphismGroup(M);
82     end if;
83
84     if auto ne SizeAuto[j] then
85         continue;
86     end if;
87
88
89     if IsIsometric(L, M) then
90         continue i;
91     end if;
92 end for;
93
94 Append(~LatticesReduced, Lattices[i]);
95
96 NewIndex := #LatticesReduced;
97 if min_computed then
98     Minima[NewIndex] := minimum;
99 end if;
100
101 if shortest_computed then
102     NumShortest[NewIndex] := shortest;
103 end if;
104
105 if auto_computed then

```

```

106         SizeAuto[NewIndex] := auto;
107     end if;
108
109 end for;
110
111 return LatticesReduced;
112 end function;
113
114
115 function ExtremalMinimum(l, n)
116 // Input: Square-free l in N; n in N
117
118 // Output: Minimum that a l-modular lattice of dimension n must
119 // have at least
120
121 if l eq 1 then k := 24;
122 elif l eq 2 then k := 16;
123 elif l eq 3 then k := 12;
124 elif l eq 5 then k := 8;
125 elif l eq 6 then k := 8;
126 elif l eq 7 then k := 6;
127 elif l eq 11 then k := 4;
128 elif l eq 14 then k := 4;
129 elif l eq 15 then k := 4;
130 elif l eq 23 then k := 2;
131 end if;
132
133 return 2 + 2*Floor(n/k);
134 end function;
135
136 function GenSymbol(L)
137 // Input: Positive definite Numberfield Lattice L of square-free
138 // level
139
140 // Output: Genus symbol of L in the form [S_1, n, <2, n_2,
141 // epsilon_2, S_2, t_2>, <3, n_3, epsilon_3>, <5,...>, ...] for all
142 // primes dividing Det(L)
143 Symbol := [* *];
144
145 Rat := RationalAsNumberField();
146 Int := Integers(Rat);
147
148 LNF := NumberFieldLatticeWithGram(Matrix(Rat, GramMatrix(L)));
149 _, Grams2, Vals2 := JordanDecomposition(LNF, ideal<Int|2>);
150
151 // Checks if all diagonal entries of the 1-component of the 2-
152 // adic jordan decomposition are even
153 if Vals2[1] ne 0 or (Vals2[1] eq 0 and &and([Valuation(Rationals
154 () ! (Grams2[1][i][i]), 2) ge 1 : i in [1..NumberOfRows(Grams2
155 [1])])) then
156     Append(~Symbol, 2);
157 else
158     Append(~Symbol, 1);
159 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
180 [1..sym[2]]) then
181                 Append(~sym, 2);
182             else
183                 Append(~sym, 1);
184             end if;
185
186             if sym[4] eq 2 then
187                 Append(~sym, 0);
188             else
189                 Append(~sym, Integers() ! (Trace(G)* Denominator(Trace
190 (G))^2) mod 8);
191             end if;
192         else
193             Append(~sym, LegendreSymbol(det, p));
194         end if;
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 );
209   Form:=IntegralMatrix(Form);
210
211   LZ := LatticeWithGram(LLGram(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
221 // partial dual quotient L^(#, p) / L
222
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
260 function IsModular(L, l)

```

```

261 // Input: Lattice L; l in N
262
263 // Output: true iff L is a l-modular lattice
264
265     return IsIsometric(L, LatticeWithGram(l*GramMatrix(Dual
(L:Rescale:=false))));
266
267 end function;
268
269 function IsStronglyModular(L,l)
270 // Input: Lattice L; l in N
271
272 // Output: true iff L is a strongly l-modular lattice
273
274     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]];
275
276 end function;

```