

```

1
2 function AutomorphismTypes(l, k, n, t)
3 // Input: Square-free l in N, k in N, n in N, t in N
4
5 // Output: List of all possible types of automorphisms
  of prime order for even lattices of level l with
  determinant l^k, dimension n and minimum greater or
  equal to t
6   Results := [];
7
8   lFactors := PrimeDivisors(l);
9
10  for p in PrimesUpTo(n+1) do
11    if p in lFactors then continue; end if;
12
13    K<z> := CyclotomicField(p);
14    Kpos := sub<K|z+1/z>;
15
16    f := [];
17
18    for q in lFactors do
19      if p le 3 then
20        Append(~f, 1);
21      else
22        Append(~f, InertiaDegree(Factorization
  (ideal<Integers(Kpos) | q>)[1][1]));
23      end if;
24    end for;
25
26    for np in [i*(p-1) : i in [1..Floor(n/
  (p-1))]] do
27
28      n1 := n - np;
29      if l eq 1 then
30        for s in [0..Min(n1, Integers() ! (np/
  (p-1)))] do
31          if not IsDivisibleBy(s - Integers
  () ! (np / (p-1)), 2) then continue s; end if;
32          if n1 gt 0 then
33            Gamma1 := t/p^(s/n1);
34            if Gamma1 gt HermiteBounds
  [n1] + 0.1 then continue s; end if;
35          end if;
36
37          if np gt 0 then
38            Gammap := t/p^(s/np);
39            if Gammap gt HermiteBounds
  [np] + 0.1 then continue s; end if;

```

```

40         end if;
41         type := <p, n1, np, s>;
42
43         Append(~Results, type);
44     end for;
45 else
46     for kp in CartesianProduct([[2*f
[i]*j : j in [0..Floor(Min(np,k)/(2*f[i]))]] : i in
[1..#f]]) do
47
48         k1 := [k - kp[i] : i in [1..#kp]];
49
50         for i in [1..#kp] do
51             if k1[i] gt Min(n1,k) then
continue kp; end if;
52             if not IsDivisibleBy(k1[i] -
k, 2) then continue kp; end if;
53             if not IsDivisibleBy(kp[i],
2) then continue kp; end if;
54         end for;
55
56         for s in [0..Min(n1, Integers() !
(np / (p-1)))] do
57             if not IsDivisibleBy(s -
Integers() ! ((p-2) * np / (p-1)), 2) then continue s;
end if;
58
59             if n1 gt 0 then
60                 Gamma1 := p^s;
61                 for i in [1..#lFactors] do
62                     Gamma1 := lFactors
[i]^k1[i];
63                 end for;
64                 Gamma1 := t / Gamma1^(1/
n1);
65
66                 if Gamma1 gt HermiteBounds
[n1] + 0.1 then continue s; end if;
67             end if;
68
69             if np gt 0 then
70                 Gammap := p^s;
71                 for i in [1..#lFactors] do
72                     Gammap := lFactors
[i]^kp[i];

```

```

73         end for;
74         Gammap := t / Gammap^(1/
np);
75
76         if Gammap gt HermiteBounds
[ np ] + 0.1 then continue s; end if;
77         end if;
78
79         if p eq 2 then
80             if n1 gt 0 then
81                 Gamma1 := 1;
82                 for i in
[ 1..#lFactors ] do
83                     Gamma1 *:=
lFactors[i]^k1[i];
84                 end for;
85                 Gamma1 := t/2 /
Gamma1^(1/n1);
86
87                 if Gamma1 gt
HermiteBounds[n1] + 0.1 then continue s; end if;
88                 end if;
89
90                 if np gt 0 then
91                     Gammap := 1;
92                     for i in
[ 1..#lFactors ] do
93                         Gammap *:=
lFactors[i]^kp[i];
94                     end for;
95                     Gammap := t/2 /
Gammap^(1/np);
96
97                     if Gammap gt
HermiteBounds[np] + 0.1 then continue s; end if;
98                     end if;
99                     end if;
100
101         type := <p, n1, np, s>;
102         for i in [ 1..#lFactors ] do
103             Append(~type, lFactors
[i]);
104             Append(~type, k1[i]);
105             Append(~type, kp[i]);
106         end for;
107
108         Append(~Results, type);
109     end for;

```

```

110         end for;
111     end if;
112 end for;
113 end for;
114
115     return Results;
116
117 end function;
118
119
120 function EnumerateGenusOfRepresentative(L, t)
121 // Input: Lattice L, t in N
122
123 // Output: List of all representatives of isometry-
124           classes in the genus of L with Minimum at least t
125     if Dimension(L) le 8 then
126         Gen := GenusRepresentatives(L);
127         ZGen := [];
128         for M in Gen do
129             if Type(M) eq Lat then
130                 Append(~ZGen, LLL(M));
131             else
132                 Append(~ZGen, LatticeWithGram(LLLGram
133 (Matrix(Rationals()), GramMatrix(SimpleLattice(M)))));
134             end if;
135         end for;
136         return [M : M in ZGen | Minimum(M) ge t];
137     end if;
138
139     M := Mass(L);
140     m := 1 / #AutomorphismGroup(L);
141     Gen := [L];
142     Explored := [false];
143     NumFound := [1];
144     Minima := [* *];
145     NumShortest := AssociativeArray();
146     SizeAuto := AssociativeArray();
147
148     if Minimum(L) ge t then
149         GenMin := [L];
150     else
151         GenMin := [];
152     end if;
153
154     if m lt M then
155         "Entering Kneser neighboring-method";
156     end if;

```

```

155
156     while m < M do
157         printf "Difference to actual mass is %o\n", M-
m;
158         RareFound := [];
159         MinCount := Infinity();
160
161         for i in [1..#Gen] do
162             if not Explored[i] then
163                 if NumFound[i] < MinCount then
164                     RareFound := [i];
165                     MinCount := NumFound[i];
166                 elif NumFound[i] eq MinCount then
167                     Append(~RareFound, i);
168                 end if;
169             end if;
170         end for;
171
172         i := RareFound[Random(1, #RareFound)];
173
174         Neigh := Neighbours(Gen[i], 2);
175         Explored[i] := true;
176         for N in Neigh do
177             MinAuto := 1 / (M - m);
178
179             // Efficient isometry test follows
180             min_computed := false;
181             minimum := 0;
182
183             shortest_computed := false;
184             shortest := 0;
185
186             auto_computed := false;
187             auto := 0;
188
189             for j in [1..#Gen] do
190                 K := Gen[j];
191
192                 if not min_computed then
193                     min_computed := true;
194                     minimum := Min(N);
195                 end if;
196
197                 if not IsDefined(Minima, j) then
198                     Minima[j] := Min(K);
199                 end if;
200
201                 if minimum ne Minima[j] then

```

```

202         continue;
203     end if;
204
205
206     if not shortest_computed then
207         shortest_computed := true;
208         shortest := #ShortestVectors(N);
209     end if;
210
211     if not IsDefined(NumShortest, j) then
212         NumShortest[j] := #ShortestVectors
(K);
213     end if;
214
215     if shortest ne NumShortest[j] then
216         continue;
217     end if;
218
219
220     if not auto_computed then
221         auto_computed := true;
222         auto := #AutomorphismGroup(N);
223
224         if auto lt MinAuto then continue
N; end if;
225
226     end if;
227
228     if not IsDefined(SizeAuto, j) then
229         SizeAuto[j] := #AutomorphismGroup
(K);
230     end if;
231
232     if auto ne SizeAuto[j] then
233         continue;
234     end if;
235
236
237     if IsIsometric(N, K) then
238         NumFound[j] += 1;
239         continue N;
240     end if;
241 end for;
242
243 Append(~Gen, N);
244 Append(~Explored, false);
245 Append(~NumFound, 1);
246

```

```

247         NewIndex := #Gen;
248         if min_computed then
249             Minima[NewIndex] := minimum;
250         end if;
251
252         if shortest_computed then
253             NumShortest[NewIndex] := shortest;
254         end if;
255
256         if not auto_computed then
257             auto := #AutomorphismGroup(N);
258         end if;
259         SizeAuto[NewIndex] := auto;
260         m += auto;
261
262         if Minimum(N) lt t then
263             Append(~GenMin, LLL(N));
264         end if;
265
266         if m eq M then
267             break N;
268         end if;
269     end for;
270 end while;
271
272 return GenMin;
273
274 end function;
275
276 function EnumerateGenusDeterminant(det, n, t)
277 // Input: det in N, n in N, t
278
279 // Output: Representatives of all isometry-classes
280 with minimum >= t belonging to a genus with even
281 lattices with determinant det, dimension n, and square-
282 free level
283     if n eq 2 then
284         Results := [];
285
286         for m:= t to Floor(1.155*Sqrt(det)) by 2 do
287             for a in [-m+1..m-1] do
288                 if not IsDivisibleBy(det + a^2, m)
289 then continue; end if;
290                 b := Integers() ! ((det + a^2) / m);

```

```

291         if b lt m then continue; end if;
292         if not IsEven(b) then continue; end
if;
293
294         Mat := Matrix(Rationals(), 2, 2,
[m,a,a,b]);
295         if not IsPositiveDefinite(Mat) then
continue; end if;
296
297         L := LatticeWithGram(Mat);
298
299         if not IsSquarefree(Level(L)) then
continue; end if;
300
301         Symbol := GenSymbol(L);
302         if not Symbol[1] eq 2 then continue;
end if;
303         if IsDivisibleBy(Determinant(L), 2)
then
304             if not Symbol[3][4] eq 2 then
continue; end if;
305             end if;
306
307         Append(~Results, L);
308     end for;
309 end for;
310
311     return Results;
312 end if;
313
314
315     Rat := RationalsAsNumberField();
316     Int := Integers(Rat);
317
318     primes := PrimeBasis(det);
319     exps := [Valuation(det, p) : p in primes];
320
321     IdealList := [];
322     if not 2 in primes then
323         Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
324     end if;
325
326     for i in [1..#primes] do
327         p := primes[i];
328         e := Abs(exps[i]);
329         if n eq e then
330             Append(~IdealList, <ideal<Int|p>,
[[1,e]]>);

```



```

331         else
332             Append(~IdealList, <ideal<Int|p>, [[0,n-
e],[1,e]]>);
333         end if;
334     end for;
335
336     try
337         Rep := LatticesWithGivenElementaryDivisors
(Rat, n, IdealList);
338     catch e
339         print "Error while trying to construct a
representative. IdealList:";
340         IdealList;
341         return [];
342     end try;
343
344     PossibleRep := [];
345     for L in Rep do
346
347         LZ := ToZLattice(L);
348         if IsSquarefree(Level(LZ)) then
349             Symbol := GenSymbol(LZ);
350             if not Symbol[1] eq 2 then continue L;
end if;
351             if IsDivisibleBy(det, 2) then
352                 if not Symbol[3][4] eq 2 then continue
L; end if;
353             end if;
354
355             Append(~PossibleRep, LZ);
356         end if;
357     end for;
358
359     return &cat([EnumerateGenusOfRepresentative(L,
t) : L in PossibleRep]);
360
361 end function;
362
363
364 function EnumerateGenusSymbol(Symbol, t)
365 // Input: Genus-symbol Symbol of positive definite
lattices of square-free level; t in N
366
367 // Output: Representatives of all isometry-classes
with minimum >= t belonging to the genus
368
369 if Symbol[2] eq 2 then
370     det := &*[Symbol[i][1]^Symbol[i][2] : i in

```

```

[3..#Symbol]];
371
372     for m:= t to Floor(1.155*Sqrt(det)) by 2 do
373         for a in [-m+1..m-1] do
374
375             if not IsDivisibleBy(det + a^2, m)
376 then continue; end if;
376             b := Integers() ! ((det + a^2) / m);
377
378             if b lt m then continue; end if;
379             if not IsEven(b) then continue; end
380 if;
380
381             Mat := Matrix(Rationals(), 2, 2,
382 [m,a,a,b]);
382             if not IsPositiveDefinite(Mat) then
383 continue; end if;
383
384             L := LatticeWithGram(Mat);
385
386             if not IsSquarefree(Level(L)) then
387 continue; end if;
387
388             if Symbol eq GenSymbol(L) then
389                 return
EnumerateGenusOfRepresentative(L, t);
390             end if;
391         end for;
392     end for;
393
394     return [];
395
396 end if;
397
398 Rat := RationalsAsNumberField();
399 Int := Integers(Rat);
400
401 n := Symbol[2];
402
403 IdealList := [];
404 if Symbol[3][1] ne 2 then
405     Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
406 end if;
407
408 for i in [3..#Symbol] do
409     p := Symbol[i][1];
410     np := Symbol[i][2];
411

```

```

412         if n eq np then
413             Append(~IdealList, <ideal<Int|p>,
[[1,np]]>);
414         else
415             Append(~IdealList, <ideal<Int|p>, [[0,n-
np],[1,np]]>);
416         end if;
417     end for;
418
419     try
420         Rep := LatticesWithGivenElementaryDivisors
(Rat, n, IdealList);
421     catch e
422         print "Error while trying to construct a
representative. IdealList:";
423         IdealList;
424         return [];
425     end try;
426
427     for L in Rep do
428         LZ := ToZLattice(L);
429         if GenSymbol(LZ) eq Symbol then
430             return EnumerateGenusOfRepresentative(LZ,
t);
431         end if;
432     end for;
433
434     return [];
435
436 end function;
437
438
439 function SuperLattices(L, p, s)
440 // Input: Lattice L; Prime p; s in N; t in N
441
442 // Output: All integral superlattices of L with index
p^s and minimum at least t
443     T,_mapT:=DualQuotient(L);
444     mapTinv := Inverse(mapT);
445     Tp:=pPrimaryComponent(T,p);
446
447     m:=#Generators(Tp);
448     G:=GramMatrix(L);
449     G_F:=MatrixAlgebra(GF(p),m)!0;
450
451     for i:=1 to m do
452         for j:=1 to m do
453             G_F[i,j]:=GF(p)!(p*InnerProduct(mapTinv

```

```

(Tp.i),mapTinv(Tp.j))));
454         end for;
455     end for;
456
457     V:=KSpace(GF(p),m,G_F);
458     if not s le WittIndex(V) then
459         return [];
460     end if;
461
462     M1:=MaximalTotallyIsotropicSubspace(V);
463     M:=sub< M1 | Basis(M1)[1..s] >;
464
465     O:=IsometryGroup(V);
466     Aut:=AutomorphismGroup(L:Decomposition:=true);
467
468     Gens:=[];
469     for g in Generators(Aut) do
470         g_F:=MatrixAlgebra(GF(p),m)!0;
471         for i:=1 to m do
472             g_F[i]:=V!Vector(Eltseq(Tp!mapT(mapTinv(T!
Tp!Eltseq(V.i))*MatrixAlgebra(Rationals(),Dimension
(L))!g)));
473         end for;
474         Append(~Gens,g_F);
475     end for;
476
477     O_L:=sub< O | Gens>;
478
479     mapS,S,Kernel:=OrbitAction(O_L,Orbit(O,M));
480     Set:=[Inverse(mapS)(i[2]) : i in
OrbitRepresentatives(S)];
481     SuperLat := [CoordinateLattice(ext< L | [mapTinv
(Tp!Eltseq(x)) : x in Basis(W)] >) : W in Set];
482
483     return SuperLat;
484
485 end function;
486
487
488 function ConstructLattices(l, n)
489 // Input: Square-free l; n in N
490
491 // Output: List of all extremal l-modular lattices
that have a large automorphism sigma of order m with
n/2 < phi(m) < n, such that there is a prime divisor p
of m with ggT(p,l) = 1 and mu_sigma / Phi_m | (x^(m/p)
- 1)
492     Results := [];

```

```

493
494     min := ExtremalMinimum(l,n);
495
496     AutoTypes := AutomorphismTypes(l, Integers() !
(n/2), n, min);
497
498     for phim in [Integers() ! (n/2)+1 .. n] do
499
500         n1 := n - phim;
501         np := phim;
502
503         for m in [m : m in EulerPhiInverse(phim) |
IsDivisibleBy(m,4)] do
504
505             printf "m = %o\n", m;
506
507             for p in PrimeDivisors(m) do
508                 //printf "Testing p = %o\n", p;
509                 if Gcd(p,l) ne 1 then continue; end
if;
510                 d := Integers() ! (m/p);
511                 PossibleTypes := [type : type in
AutoTypes | type[1] eq p and type[2] eq n1 and type[3]
eq np and EulerPhi(d) le type[4]];
512
513                 //printf "Have to check %o possible
automorphism-types\n", #PossibleTypes;
514
515                 for type in PossibleTypes do
516                     s := type[4];
517
518                     detp := p^s;
519                     for i := 5 to #type by 3 do
520                         detp *= type[i]^type[i+2];
521                     end for;
522
523                     // Enumerate ideal-lattices over K
(zeta_m) with given determinant
524                     K<z> := CyclotomicField(m);
525                     Kpos := sub<K | z + z^(-1)>;
526
527                     A := ClassesModGalois(K);
528                     M, U, FundUnits := EmbeddingMatrix
(K, Kpos);
529                     LpList := IdealLattices(detp, K,
Kpos, A, M, U, FundUnits, false);
530
531                     LpList := [L : L in LpList |

```

```

Minimum(L) ge min];
532
533         LpList := ReduceByIsometry
(LpList);
534
535         for Lp in LpList do
536             //CAP := ConjugacyClasses
(AutomorphismGroup(Lp));
537             //sigmaplist := [c[3] : c in CAP
| MiPoQuotient(c[3], Lp, p) eq Polynomial(GF
(p),CyclotomicPolynomial(d))] do
538
539             // Enumerate genus
540
541             if IsPrime(l) and p gt 2 then
542                 // In this case the genus
symbol of L_1 is known and allows for a more efficient
enumeration
543                 k1 := type[6];
544                 kp := type[7];
545
546                 f := InertiaDegree
(Factorization(ideal<Integers(Kpos) | l>)[1][1]);
547                 deltap := (-1)^(Integers
() ! (kp/f + (p-1)/2 * (Binomial(Integers() ! (np /
(p-1) + 1), 2) + Binomial(s, 2)))));
548                 delta1 := deltap * (-1)^
(Integers() ! (s*(p-1)/2));
549
550                 if l eq 2 then
551                     if IsDivisibleBy(np +
s*(p-1), 8) then
552                         epsilonp :=
deltap;
553                     else
554                         epsilonp := -
deltap;
555                     end if;
556
557                     if IsDivisibleBy(n,
8) then
558                         epsilon := 1;
559                     else
560                         epsilon := -1;
561                     end if;
562                 else
563                     epsilonp := (-1)^
(Integers() ! (kp / f + (l-1)/2*Binomial(kp,2)));

```



```

600             M := CoordinateLattice
        (OrthogonalSum(L1, Lp));
601             LList := SuperLattices(M, p,
        s);
602             Results cat:= [L : L in LList
        | Minimum(L) ge min];
603
604             end for;
605         end for;
606     end for;
607 end for;
608 end for;
609 end for;
610
611     return ReduceByIsometry(Results);
612
613 end function;
614
615
616 for n := 2 to 36 by 2
do
617     for l in [1,2,3,5,6,7,11,14,15,23] do
618         if l eq 1 and n in [2,4,6] then continue; end
        if;
619         if l eq 2 and n eq 2 then continue; end if;
620         if l eq 11 and n in [20,24,28,30,32,34,36]
        then continue; end if;
621         if l eq 23 and n ge 6 then continue; end if;
622         printf "dim = %o, l = %o\n", n, l;
623         List := ConstructLattices(l, n);
624         ModList := [L : L in List | IsModular(L, l)];
625         StrongModList := [L : L in List |
        IsStronglyModular(L,l)];
626         if #List gt 0 then
627             printf "\nn = %o, l = %o: Found %o
        lattices! %o of them are modular and %o are strongly
        modular.\n\n", n, l, #List, #ModList, #StrongModList;
628         end if;
629         PrintFileMagma(Sprintf("SubidealLattices/%o-
        Modular/%o-Dimensional", l, n), List : Overwrite :=
        true);
630         PrintFileMagma(Sprintf("SubidealLattices/%o-
        Modular/%o-DimensionalModular", l, n), ModList :
        Overwrite := true);
631         PrintFileMagma(Sprintf("SubidealLattices/%o-
        Modular/%o-DimensionalStronglyModular", l, n),
        StrongModList : Overwrite := true);
632     end for;

```



```
633  end for;
```