

```

1  load "Utility.m";
2  load "IdeallLattices.m";
3
4
5  function AutomorphismTypes(l, k, n, t)
6  // Input: Square-free l in N, k in N, n in N, t in N
7
8  // 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
9      Results := [];
10
11      lFactors := PrimeDivisors(l);
12
13      for p in PrimesUpTo(n+1) do
14          if p in lFactors then continue; end if;
15
16          K<z> := CyclotomicField(p);
17          Kpos := sub<K|z+1/z>;
18
19          f := [];
20
21          for q in lFactors do
22              if p le 3 then
23                  Append(~f, 1);
24              else
25                  Append(~f, InertiaDegree(Factorization(ideal<Integers
(Kpos) | q>)[1][1]));
26              end if;
27          end for;
28
29          for np in [i*(p-1) : i in [1..Floor(n/(p-1))]] do
30
31              n1 := n - np;
32              for s in [0..Min(n1, Integers() ! (np/(p-1)))] do
33                  if not IsDivisibleBy(s - Integers() ! (np / (p-1)), 2)
then continue s; end if;
34                  if p eq 2 and not IsDivisibleBy(s, 2) then continue s;
end if;
35
36                  if l eq 1 then
37                      if n1 gt 0 then
38                          Gammap := t/p^(s/n1);
39                          if Gammap gt HermiteBounds[n1] + 0.1 then continue;
end if;
40                      end if;
41
42                      if np gt 0 then
43                          Gammap := t/p^(s/np);
44                          if Gammap gt HermiteBounds[np] + 0.1 then continue;
end if;
45                      end if;
46                      type := <p, n1, np, s>;
47
48                      Append(~Results, type);
49                  else

```

```

50         for kp in CartesianProduct([[2*f[i]*j : j in [0..Floor
(Min(np,k)/(2*f[i]))]] : i in [1..#f]]) do
51
52             k1 := [k - kp[i] : i in [1..#kp]];
53
54             for i in [1..#kp] do
55                 if k1[i] gt Min(n1,k) then continue kp; end if;
56                 if not IsDivisibleBy(k1[i] - k, 2) then continue
kp; end if;
57                 if not IsDivisibleBy(kp[i], 2) then continue kp;
end if;
58             end for;
59
60             if n1 gt 0 then
61                 Gamma1 := p^s;
62                 for i in [1..#lFactors] do
63                     Gamma1 *= lFactors[i]^k1[i];
64                 end for;
65                 Gamma1 := t / Gamma1^(1/n1);
66
67                 if Gamma1 gt HermiteBounds[n1] + 0.1 then
continue; end if;
68             end if;
69
70             if np gt 0 then
71                 Gammap := p^s;
72                 for i in [1..#lFactors] do
73                     Gammap *= lFactors[i]^kp[i];
74                 end for;
75                 Gammap := t / Gammap^(1/np);
76
77                 if Gammap gt HermiteBounds[np] + 0.1 then
continue; end if;
78             end if;
79
80             if p eq 2 then
81                 if n1 gt 0 then
82                     Gamma1 := 1;
83                     for i in [1..#lFactors] do
84                         Gamma1 *= lFactors[i]^k1[i];
85                     end for;
86                     Gamma1 := t/2 / Gamma1^(1/n1);
87
88                     if Gamma1 gt HermiteBounds[n1] + 0.1 then
continue; end if;
89                 end if;
90
91                 if np gt 0 then
92                     Gammap := 1;
93                     for i in [1..#lFactors] do
94                         Gammap *= lFactors[i]^kp[i];
95                     end for;
96                     Gammap := t/2 / Gammap^(1/np);
97
98                     if Gammap gt HermiteBounds[np] + 0.1 then
continue; end if;

```

```

99         end if;
100     end if;
101
102     type := <p, n1, np, s>;
103     for i in [1..#lFactors] do
104         Append(~type, lFactors[i]);
105         Append(~type, k1[i]);
106         Append(~type, kp[i]);
107     end for;
108
109     Append(~Results, type);
110 end for;
111 end if;
112 end for;
113 end for;
114 end for;
115
116 return Results;
117
118 end function;
119
120
121 function EnumerateGenusOfRepresentative(L)
122 // Input: Lattice L, t in N
123
124 // Output: List of all representatives of isometry-classes in the
125 // genus of L
126
127 "Enumerate genus of representative";
128 try return eval Read(Sprintf("GenusSymbols/Gen_%o", GenSymbol
129 (L))); catch e; end try;
130
131 try
132     Gen := eval Read(Sprintf("GenusSymbols/Gen_%o_partial",
133 GenSymbol(L)));
134     printf "Only using partial genus for %o!\n", GenSymbol(L);
135     return Gen;
136 catch e; end try;
137
138 if Dimension(L) le 6 then
139     Gen := GenusRepresentatives(L);
140     ZGen := [];
141     for M in Gen do
142         if Type(M) eq Lat then
143             Append(~ZGen, LLL(M));
144         else
145             Append(~ZGen, LatticeWithGram(LLLGram(Matrix(Rationals()),
146 GramMatrix(SimpleLattice(M)))));
147         end if;
148     end for;
149     PrintFileMagma(Sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
150 ZGen : Overwrite := true);
151     return ZGen;
152 end if;
153
154 M := Mass(L);
155 Gen := [L];

```

```

150     Explored := [false];
151     NumFound := [1];
152     Minima := [Minimum(L)];
153     NumShortest := [#ShortestVectors(L)];
154     SizeAuto := [#AutomorphismGroup(L)];
155     m := 1 / SizeAuto[1];
156
157     p := 2;
158
159     t0 := Realtime();
160
161     while m lt M do
162         //printf "So far %o classes found. Difference to actual mass
is %o. \n", #Gen, M-m;
163         if Realtime(t0) ge 120*60 then
164             printf "2 hours have elapsed and not the whole genus was
explored. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
165             PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
166             return Gen;
167         end if;
168
169         RareFound := [];
170         MinCount := Infinity();
171
172         if &and(Explored) then
173             "All explored. Going to next prime.";
174             Explored := [false : x in Explored];
175             p := NextPrime(p);
176             if p ge 5 and Dimension(L) ge 8 then
177                 printf "Prime too large, cannot continue constructing
neighbours. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
178                 PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
179                 return Gen;
180             end if;
181             if p ge 3 and Dimension(L) ge 12 then
182                 printf "Prime too large, cannot continue constructing
neighbours. Remaining difference to actual mass is %o. %o classes
were found so far. The symbol is %o.\n", M-m, #Gen, GenSymbol(L);
183                 PrintFileMagma(Sprintf("GenusSymbols/Gen_%o_partial",
GenSymbol(L)), Gen : Overwrite := true);
184                 return Gen;
185             end if;
186         end if;
187
188         for i in [1..#Gen] do
189             if not Explored[i] then
190                 if NumFound[i] lt MinCount then
191                     RareFound := [i];
192                     MinCount := NumFound[i];

```

```

193         elif NumFound[i] eq MinCount then
194             Append(~RareFound, i);
195         end if;
196     end if;
197 end for;
198
199 i := RareFound[Random(1, #RareFound)];
200
201 Neigh := [CoordinateLattice(N) : N in Neighbours(Gen[i], p)];
202 Explored[i] := true;
203
204 for N in Neigh do
205
206     auto := #AutomorphismGroup(N);
207     if auto lt 1/(M-m) then continue; end if;
208
209     minimum := Minimum(N);
210     shortest := #ShortestVectors(N);
211
212     for j in [1..#Gen] do
213
214         if minimum ne Minima[j] then
215             continue j;
216         end if;
217
218         if shortest ne NumShortest[j] then
219             continue j;
220         end if;
221
222         if auto ne SizeAuto[j] then
223             continue j;
224         end if;
225
226         if IsIsometric(N, Gen[j]) then
227             NumFound[j] += 1;
228             continue N;
229         end if;
230     end for;
231
232     Append(~Gen, N);
233     Append(~Explored, false);
234     Append(~NumFound, 1);
235     Append(~Minima, minimum);
236     Append(~NumShortest, shortest);
237     Append(~SizeAuto, auto);
238     m += 1/auto;
239     if m eq M then
240         break N;
241     end if;
242 end for;
243 end while;
244
245 PrintFileMagma(sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
Gen : Overwrite := true);
246

```

```

247     return Gen;
248
249 end function;
250
251
252 function EnumerateGenusDeterminant(det, n, even)
253 // Input: det in N; n in N; boolean even that indicates whether
254 //         only even lattices shall be enumerated
255 // Output: Representatives of all isometry-classes belonging to a
256 //         genus of integral lattices with determinant det, dimension n, and
257 //         square-free level
258
259 if n eq 0 then
260     return [LatticeWithGram(Matrix(Rationals(), 0, 0, []))];
261 end if;
262
263 if n eq 1 then
264     L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
265     Symbol := GenSymbol(L);
266     if even and not Symbol[1] eq 2 then return []; end if;
267     if not IsSquarefree(Level(L)) then return []; end if;
268     if even and IsDivisibleBy(Determinant(L), 2) then
269         if not Symbol[3][4] eq 2 then return []; end if;
270     end if;
271     return [L];
272 end if;
273
274 if n eq 2 then
275     Results := [];
276
277     for m in [1..Floor(1.155*Sqrt(det))] do
278         for a in [-m+1..m-1] do
279
280             if not IsDivisibleBy(det + a^2, m) then continue; end if;
281             b := Integers() ! ((det + a^2) / m);
282
283             if b lt m then continue; end if;
284             if even and not IsEven(b) then continue; end if;
285
286             Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
287             if not IsPositiveDefinite(Mat) then continue; end if;
288
289             L := LatticeWithGram(Mat);
290
291             if not IsSquarefree(Level(L)) then continue; end if;
292
293             Symbol := GenSymbol(L);
294             if even and not Symbol[1] eq 2 then continue; end if;
295             if even and IsDivisibleBy(Determinant(L), 2) then
296                 if not Symbol[3][4] eq 2 then continue; end if;
297             end if;
298
299             Append(~Results, L);
300         end for;
301     end for;

```

```

299     end for;
300
301     return ReduceByIsometry(Results);
302 end if;
303
304
305 Rat := RationalsAsNumberField();
306 Int := Integers(Rat);
307
308 primes := PrimeBasis(det);
309 exps := [Valuation(det, p) : p in primes];
310
311 IdealList := [];
312 if not 2 in primes then
313     Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
314 end if;
315
316 for i in [1..#primes] do
317     p := primes[i];
318     e := Abs(exps[i]);
319     if n eq e then
320         Append(~IdealList, <ideal<Int|p>, [[1,e]]>);
321     elif e eq 0 then
322         Append(~IdealList, <ideal<Int|p>, [[0,n]]>);
323     else
324         Append(~IdealList, <ideal<Int|p>, [[0,n-e],[1,e]]>);
325     end if;
326 end for;
327
328 "Constructing representatives";
329 try
330     Rep := LatticesWithGivenElementaryDivisors(Rat, n, IdealList);
331 catch e
332     print "Error while trying to construct a representative.
IdealList:";
333     IdealList;
334     return [];
335 end try;
336
337 Results := [];
338
339 for L in Rep do
340
341     LZ := ToZLattice(L);
342     if IsSquarefree(Level(LZ)) then
343         Symbol := GenSymbol(LZ);
344         if even and not Symbol[1] eq 2 then continue L; end if;
345         if even and IsDivisibleBy(det, 2) then
346             if not Symbol[3][4] eq 2 then continue L; end if;
347         end if;
348
349         Gen := EnumerateGenusOfRepresentative(LZ);
350         Results cat:= Gen;
351     end if;
352 end for;

```

```

353
354     return Results;
355
356 end function;
357
358
359 function EnumerateGenusSymbol(Symbol)
360 // Input: Genus-symbol Symbol of positive definite lattices of
361 // square-free level; t in N
362
363 // Output: Representatives of all isometry-classes belonging to
364 // the genus
365
366     try return eval Read(Sprintf("GenusSymbols/Gen_%o", Symbol));
367 catch e; end try;
368     try
369         Gen := eval Read(Sprintf("GenusSymbols/Gen_%o_partial",
370 Symbol));
371         printf "Only using partial genus for %o!\n", Symbol;
372         return Gen;
373     catch e; end try;
374
375     n := Symbol[2];
376
377     if n eq 0 then
378         return [LatticeWithGram(Matrix(Rationals(),0,0,[]))];
379     end if;
380
381     if n eq 1 then
382         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
383         L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
384         if GenSymbol(L) eq Symbol then
385             return [L];
386         end if;
387         return [];
388     end if;
389
390     if n eq 2 then
391         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
392
393         for m := 2 to Floor(1.155*Sqrt(det)) by 2 do
394             for a in [-m+1..m-1] do
395
396                 if not IsDivisibleBy(det + a^2, m) then continue; end if;
397                 b := Integers() ! ((det + a^2) / m);
398
399                 if b lt m then continue; end if;
400                 if not IsEven(b) then continue; end if;
401
402                 Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
403                 if not IsPositiveDefinite(Mat) then continue; end if;
404
405                 L := LatticeWithGram(Mat);
406
407                 if not IsSquarefree(Level(L)) then continue; end if;

```



```

404
405         if Symbol eq GenSymbol(L) then
406             return EnumerateGenusOfRepresentative(L);
407         end if;
408     end for;
409 end for;
410
411     return [];
412
413 end if;
414
415 Rat := RationalsAsNumberField();
416 Int := Integers(Rat);
417
418 IdealList := [];
419 if Symbol[3][1] ne 2 then
420     Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
421 end if;
422
423 for i in [3..#Symbol] do
424     p := Symbol[i][1];
425     np := Symbol[i][2];
426
427     if n eq np then
428         Append(~IdealList, <ideal<Int|p>, [[1,np]]>);
429     elif np eq 0 then
430         Append(~IdealList, <ideal<Int|p>, [[0,n]]>);
431     else
432         Append(~IdealList, <ideal<Int|p>, [[0,n-np],[1,np]]>);
433     end if;
434 end for;
435
436 "Constructing representatives";
437 try
438     Rep := LatticesWithGivenElementaryDivisors(Rat, n, IdealList);
439 catch e
440     print "Error while trying to construct a representative.
IdealList:";
441     IdealList;
442     return [];
443 end try;
444
445 for L in Rep do
446     LZ := ToZLattice(L);
447     if GenSymbol(LZ) eq Symbol then
448         Gen := EnumerateGenusOfRepresentative(LZ);
449         return Gen;
450     end if;
451 end for;
452
453 return [];
454
455 end function;
456
457

```

```

458 function SuperLatticesMagma(L, p, s, sigma)
459 // Input: Lattice L; Prime p; s in N; Automorphism sigma of L
460
461 // Output: All even sigma-invariant superlattices of L with index
         p^s using magmas method
462
463
464     LD := PartialDual(L,p:Rescale:=false);
465
466     G := MatrixGroup<NumberOfRows(sigma), Integers() | sigma >;
467     den1 := Denominator(BasisMatrix(LD));
468     den2 := Denominator(InnerProductMatrix(LD));
469
470     A := LatticeWithBasis(G, Matrix(Integers(), den1*BasisMatrix
         (LD)), Matrix(Integers(), den2^2*InnerProductMatrix(LD)));
471
472     SU := [];
473     SU := Sublattices(A, p : Levels := s, Limit := 100000);
474
475     if #SU eq 100000 then "List of sublattices is probably not
         complete."; end if;
476
477     Results := [];
478
479     for S in SU do
480         M := 1/den1 * 1/den2 * S;
481
482         if Determinant(M)*p^(2*s) eq Determinant(L) then
483             Append(~Results, M);
484         end if;
485     end for;
486
487     return [L : L in Results | IsEven(L)];
488 end function;
489
490
491
492 function SuperLattices(L1, Lp, p, sigma1, sigma2)
493 // Input: Lattice L1; Lattice Lp; Prime p; Automorphism sigma of L
494
495 // Output: All even superlattices of L1 + Lp invariant under diag
         (sigma1, sigma2) with index p^s using isometry-method
496
497     M := OrthogonalSum(L1, Lp);
498
499     L1Quot, phi1 := PartialDual(L1,p : Rescale:=false) / L1;
500     LpQuot, phi2 := PartialDual(Lp,p : Rescale:=false) / Lp;
501
502     m := #Generators(L1Quot);
503
504     phi1Inv := Inverse(phi1);
505     phi2Inv := Inverse(phi2);
506
507     G1 := ZeroMatrix(GF(p),m,m);
508     G2 := ZeroMatrix(GF(p),m,m);

```

```

509     for i in [1..m] do
510         for j in [1..m] do
511             G1[i,j] := GF(p) ! (p*InnerProduct(philInv(L1Quot.i),
philInv(L1Quot.j)));
512             Gp[i,j] := GF(p) ! (-p*InnerProduct(philInv(LpQuot.i),
philInv(LpQuot.j)));
513         end for;
514     end for;
515
516     V1 := KSpace(GF(p), m, G1);
517     Vp := KSpace(GF(p), m, Gp);
518
519     O1 := IsometryGroup(V1);
520
521     sigma1Quot := ZeroMatrix(GF(p),m,m);
522     for i in [1..m]
do
523         sigma1Quot[i] := Vector(GF(p), Eltseq(phil(philInv
(L1Quot.i)*Matrix(Rationals(),sigma1))));
524     end for;
525
526     sigma2Quot := ZeroMatrix(GF(p),m,m);
527     for i in [1..m]
do
528         sigma2Quot[i] := Vector(GF(p), Eltseq(phil(philInv
(LpQuot.i)*Matrix(Rationals(),sigma2))));
529     end for;
530
531     CL1Quot := Centralizer(O1, O1 ! sigma1Quot);
532
533     CL1 := Centralizer(AutomorphismGroup(L1), sigma1);
534
535     CL1ProjGens := [];
536     for g in Generators(CL1) do
537         gProj := ZeroMatrix(GF(p),m,m);
538         for i in [1..m] do
539             gProj[i] := Vector(GF(p), Eltseq(phil(philInv
(L1Quot.i)*Matrix(Rationals(), g))));
540         end for;
541         Append(~CL1ProjGens, gProj);
542     end for;
543
544     CL1Proj := MatrixGroup<m, GF(p) | CL1ProjGens>;
545
546     _, psi := IsIsometric(V1,Vp);
547
548     psi := MatrixOfIsomorphism(psi);
549     _, u := IsConjugate(O1, O1 ! sigma1Quot, O1 !
(psi*sigma2Quot*psi^(-1)));
550
551     phi0 := u*psi;
552
553     U, mapU := CL1Quot / CL1Proj;
554
555     LphiList := [];
556     for u in U do

```

```

557     phi := Inverse(mapU)(u)*phi0;
558
559     Gens := [];
560     for i in [1..m] do
561         x := phiInv(L1Quot.i);
562         y := phiInv(LpQuot ! Eltseq(phi[i]));
563         Append(~Gens, Eltseq(x) cat Eltseq(y));
564     end for;
565
566     Lphi := ext<M | Gens>;
567     Append(~LphiList, LatticeWithGram(LLLGram(GramMatrix(Lphi))));
568 end for;
569
570 return [L : L in LphiList | IsEven(L)];
571 end function;
572
573
574
575 function SuperLatticesJuergens(L, p, s)
576 // Input: Lattice L; Prime p; s in N; t in N
577
578 // Output: All even superlattices of L with index p^s using
579 // juergens method
580
581 if s eq 0 then
582     return [L];
583 end if;
584
585 T, mapT := PartialDual(L, p:Rescale:=false) / L;
586 mapTinv := Inverse(mapT);
587
588 m := #Generators(T);
589 G := GramMatrix(L);
590 G_F := MatrixAlgebra(GF(p), m)!0;
591
592 for i:=1 to m do
593     for j:=1 to m do
594         G_F[i, j] := GF(p)!(p*InnerProduct(mapTinv(T.i), mapTinv
595 (T.j)));
596     end for;
597 end for;
598
599 V := KSpace(GF(p), m, G_F);
600 if not s le WittIndex(V) then
601     return [];
602 end if;
603
604 M1 := MaximalTotallyIsotropicSubspace(V);
605 M := sub< M1 | Basis(M1)[1..s] >;
606
607 O := IsometryGroup(V);
608 Aut := AutomorphismGroup(L:Decomposition:=true);
609
610 Gens := [];
611 for g in Generators(Aut) do

```

```

610     g_F:=MatrixAlgebra(GF(p),m)!0;
611     for i:=1 to m do
612         g_F[i]:=V!Vector(Eltseq(mapT(mapTinv(T!Eltseq
(V.i))*Matrix(Rationals(),g))));
613     end for;
614     Append(~Gens,g_F);
615 end for;
616
617 O_L:=sub< 0 | Gens>;
618 mapS,S,Kernel:=OrbitAction(O_L,Orbit(0,M));
619 Set:=[Inverse(mapS)(i[2]) : i in OrbitRepresentatives(S)];
620 SuperLat := [CoordinateLattice(ext< L | [mapTinv(T!Eltseq
(x)) : x in Basis(W)] >) : W in Set];
621
622 return [L : L in SuperLat | IsEven(L)];
623
624 end function;
625
626
627 function MiPoQuotient(sigma, L, p);
628 // Input : Automorphism sigma of L; Lattice L
629
630 // Output: Minimal polynomial of the operation of sigma on the
partial dual quotient L^(#, p) / L
631
632 sigma := Matrix(Rationals(), sigma);
633 L := CoordinateLattice(L);
634 LD := PartialDual(L, p : Rescale := false);
635 _,phi := LD / L;
636 MiPo := PolynomialRing(GF(p)) ! 1;
637
638 B := [];
639
640 for i in [1..Rank(LD)] do
641
642     b := LD.i;
643     if b in sub<LD|L,B> then
644         continue;
645     end if;
646     Append(~B,b);
647
648     dep := false;
649     C := [Eltseq(phi(b))];
650     while not dep do
651         b := b*sigma;
652         Append(~C, Eltseq(phi(b)));
653         Mat := Matrix(GF(p),C);
654         if Dimension(Kernel(Mat)) gt 0 then
655             dep := true;
656             coeff := Basis(Kernel(Mat))[1];
657             coeff /= coeff[#C];
658             coeff := Eltseq(coeff);
659             MiPo := LCM(MiPo, Polynomial(GF(p), coeff));
660         else
661             Append(~B, b);
662         end if;

```

```

663         end while;
664     end for;
665
666     return MiPo;
667
668 end function;
669
670
671 function ConstructLattices(l, n)
672 // Input: Square-free l; n in N
673
674 // Output: List of all extremal l-modular lattices that have a
        large automorphism sigma of order m, such that there is a prime
        divisor p of m with ggT(p,l) = 1 and mu_sigma / Phi_m | (x^(m/p)
        - 1)
675     Results := [];
676
677     min := ExtremalMinimum(l,n);
678
679     AutoTypes := AutomorphismTypes(l, Integers() ! (n/2), n, min);
680
681     for phim in [Integers() ! (n/2)+1 .. n] do
682
683         n1 := n - phim;
684         np := phim;
685
686         for m in EulerPhiInverse(phim) do
687
688             printf "m = %o\n", m;
689
690             for p in PrimeDivisors(m) do
691                 if Gcd(p,l) ne 1 then continue; end if;
692                 d := Integers() ! (m/p);
693                 PossibleTypes := [type : type in AutoTypes | type[1] eq p
and type[2] eq n1 and type[3] eq np and (type[4] eq 0 or EulerPhi
(d) le type[4])];
694
695
696                 for type in PossibleTypes do
697                     s := type[4];
698
699                     detp := p^s;
700                     for i := 5 to #type by 3 do
701                         detp *= type[i]^type[i+2];
702                     end for;
703
704                     // Enumerate ideal-lattices over K(zeta_m) with given
determinant
705                     K<z> := CyclotomicField(m);
706                     Kpos := sub<K | z + z^(-1)>;
707
708                     A := ClassesModGalois(K);
709                     M, U, FundUnits := EmbeddingMatrix(K, Kpos);
710                     LpList := IdealLattices(detp, K, Kpos, A, M, U,
FundUnits, false);

```

```

711         LpList := [L : L in LpList | Minimum(L) ge min];
712         LpList := ReduceByIsometry(LpList);
713
714     if n1 eq 0 then
715         Results cat:= LpList;
716         continue type;
717     end if;
718
719     for Lp in LpList do
720         if s ne 0 then
721             sigmapList := [c[3] : c in ConjugacyClasses
722 (AutomorphismGroup(Lp)) | MiPoQuotient(c[3], Lp, p) eq Polynomial
723 (GF(p), CyclotomicPolynomial(d))];
724             if #sigmapList eq 0 then
725                 continue Lp;
726             end if;
727         end if;
728
729         "Enumerate candidates for L_1";
730
731         K<z> := CyclotomicField(p);
732         Kpos := sub<K|z+1/z>;
733
734         if p eq 2 then
735             // In this case use the sublattice U of L_1 with U^
736             {#,2} = U
737             det1U := 1;
738             for i := 5 to #type by 3 do
739                 det1U *:= type[i]^type[i+1];
740             end for;
741
742             "Enumerate sublattices U";
743             UList := EnumerateGenusDeterminant(det1U, n1,
744 false);
745
746             "Construct L1 as superlattice for U";
747             L1List := &cat[SuperLatticesJuergens
748 (LatticeWithGram(2*GramMatrix(U)), p, Integers() ! ((n1 -
749 s)/2)) : U in UList | Minimum(U) ge Ceiling(min/2)];
750             L1List := [L : L in L1List | IsEven(L) and Minimum
751 (L) ge min];
752
753             elif IsPrime(l) then
754                 // In this case the genus symbol of L_1 is known
755                 and allows for a more efficient enumeration
756                 k1 := type[6];
757                 kp := type[7];
758
759                 if p le 3 then
760                     f := 1;
761                 else
762                     f := InertiaDegree(Factorization(ideal<Integers
763 (Kpos) | l>)[1][1]);
764                 end if;

```

```

758      deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 *
(Binomial(Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
759      delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));
760
761      if l eq 2 then
762          if IsDivisibleBy(np + s*(p-1), 8) then
763              epsilonp := deltap;
764          else
765              epsilonp := -deltap;
766          end if;
767
768          if IsDivisibleBy(n, 8) then
769              epsilon := 1;
770          else
771              epsilon := -1;
772          end if;
773      else
774          epsilonp := (-1)^(Integers() ! (kp / f +
(l-1)/2*Binomial(kp,2)));
775
776          if IsDivisibleBy(n*(l+1), 16) then
777              epsilon := 1;
778          else
779              epsilon := -1;
780          end if;
781      end if;
782
783      epsilon1 := epsilonp*epsilon;
784
785      Sym1 := [* 2, n1 *];
786      if l eq 2 then
787          Append(~Sym1, <2, k1, epsilon1, 2, 0>);
788          Append(~Sym1, <p, s, delta1>);
789      else
790          if l lt p then
791              Append(~Sym1, <l, k1, epsilon1>);
792              Append(~Sym1, <p, s, delta1>);
793          else
794              Append(~Sym1, <p, s, delta1>);
795              Append(~Sym1, <l, k1, epsilon1>);
796          end if;
797      end if;
798
799      "Enumerate genus symbol";
800
801      L1List := [L : L in EnumerateGenusSymbol(Sym1) |
IsEven(L) and Minimum(L) ge min];
802
803      else
804
805          det1 := p^s;
806          for i := 5 to #type by 3 do
807              det1 := type[i]^type[i+1];
808          end for;

```



```

809
810         "Enumerate genus by determinant";
811
812         L1List := [L : L in EnumerateGenusDeterminant
813 (det1, n1, true) | IsEven(L) and Minimum(L) ge min];
814
815         end if;
816
817         for L1 in L1List do
818             M := CoordinateLattice(OrthogonalSum(L1,Lp));
819
820             if s eq 0 then
821                 if Minimum(M) ge min then
822                     Append(~Results, M);
823                 end if;
824                 continue L1;
825             end if;
826
827             sigmalList := [c[3] : c in ConjugacyClasses
828 (AutomorphismGroup(L1)) | MiPoQuotient(c[3], L1, p) eq Polynomial
829 (GF(p), CyclotomicPolynomial(d)) and Degree(MinimalPolynomial(c
830 [3])) le EulerPhi(d)];
831
832             if #sigmalList eq 0 then
833                 continue L1;
834             end if;
835
836             "Constructing superlattices";
837
838             if <l,n> in [] then
839                 for sigmal in sigmalList do
840                     for sigmap in sigmapList do
841                         LList cat:= SuperLatticesMagma(M, p, s,
842 DiagonalJoin(sigmal, sigmap));
843                     end for;
844                 end for;
845             elif <l,n> in
846 [<7,18>,<7,20>,<1,24>,<2,24>,<5,24>,<1,32>] then
847                 LList := [];
848                 for sigmal in sigmalList do
849                     for sigmap in sigmapList do
850                         LList cat:= SuperLattices(CoordinateLattice
851 (L1), CoordinateLattice(Lp), p, sigmal, sigmap);
852                     end for;
853                 end for;
854             else
855                 LList := SuperLatticesJuergens(M,p,s);
856             end if;
857
858             Results cat:= [L : L in LList | Minimum(L) ge min];
859
860         end for;
861     end for;
862 end for;

```

```

856         end for;
857     end for;
858 end for;
859
860     return ReduceByIsometry(Results);
861
862 end function;
863
864
865 procedure MainLoop(nMin, nMax : lList :=
[1,2,3,5,6,7,11,14,15,23])
866     for n := nMin to nMax by 2
867 do
868     for l in lList do
869         if l eq 1 and not IsDivisibleBy(n,8) then continue; end if;
870         if l eq 2 and n eq 2 then continue; end if;
871         if l eq 11 and n in [20,24,28,30,32,34,36] then continue;
872 end if;
873         if l eq 23 and n ge 6 then continue; end if;
874
875         printf "dim = %o, l = %o\n", n, l;
876         Results := ConstructLattices(l, n);
877         ModList := [L : L in Results | IsModular(L, l)];
878         StrongModList := [L : L in Results | IsStronglyModular
(L,l)];
879         PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-
Dimensional", l, n), Results : Overwrite := true);
880         PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-
DimensionalModular", l, n), ModList : Overwrite := true);
881         PrintFileMagma(Sprintf("SubidealLattices/%o-Modular/%o-
DimensionalStronglyModular", l, n), StrongModList : Overwrite :=
true);
882
883         if #Results gt 0 then
884             printf "\n\n-----n = %o, l = %o: %o lattices found! %
o of them are modular and %o are strongly modular-----\n\n",
n, l, #Results, #ModList, #StrongModList;
885         end if;
886     end for;
887 end for;
888 end procedure;

```