

```

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                          Gamma1 := t/p^(s/n1);
39                          if Gamma1 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 if Dimension(L) le 6 then
132     Gen := GenusRepresentatives(L);
133     ZGen := [];
134     for M in Gen do
135         if Type(M) eq Lat then
136             Append(~ZGen, LLL(M));
137         else
138             Append(~ZGen, LatticeWithGram(LLLGram(Matrix(Rationals()),
139 GramMatrix(SimpleLattice(M)))));
140         end if;
141     end for;
142     PrintFileMagma(Sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
143 ZGen : Overwrite := true);
144     return ZGen;
145 end if;
146
147 M := Mass(L);
148 Gen := [L];
149 Explored := [false];
150 NumFound := [1];
151 Minima := [Minimum(L)];
152 NumShortest := [#ShortestVectors(L)];
153 SizeAuto := [#AutomorphismGroup(L)];
154 m := 1 / SizeAuto[1];

```

```

151
152     p := 2;
153
154     t0 := Realtime();
155
156     while m < M do
157         //printf "So far %o classes found. Difference to actual mass
is %o. \n", #Gen, M-m;
158         if Realtime(t0) >= 120*60 then
159             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);
160             return Gen;
161         end if;
162
163         RareFound := [];
164         MinCount := Infinity();
165
166         if &and(Explored) then
167             "All explored. Going to next prime.";
168             Explored := [false : x in Explored];
169             p := NextPrime(p);
170             if p >= 5 and Dimension(L) >= 8 then
171                 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);
172                 return Gen;
173             end if;
174             if p >= 3 and Dimension(L) >= 12 then
175                 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);
176                 return Gen;
177             end if;
178         end if;
179
180         for i in [1..#Gen] do
181             if not Explored[i] then
182                 if NumFound[i] < MinCount then
183                     RareFound := [i];
184                     MinCount := NumFound[i];
185                 elif NumFound[i] == MinCount then
186                     Append(~RareFound, i);
187                 end if;
188             end if;
189         end for;
190
191         i := RareFound[Random(1, #RareFound)];
192
193         Neigh := [CoordinateLattice(N) : N in Neighbours(Gen[i], p)];
194         Explored[i] := true;
195

```

```

196   for N in Neigh do
197
198       auto := #AutomorphismGroup(N);
199       if auto lt 1/(M-m) then continue; end if;
200
201       minimum := Minimum(N);
202       shortest := #ShortestVectors(N);
203
204       for j in [1..#Gen] do
205
206           if minimum ne Minima[j] then
207               continue j;
208           end if;
209
210           if shortest ne NumShortest[j] then
211               continue j;
212           end if;
213
214           if auto ne SizeAuto[j] then
215               continue j;
216           end if;
217
218           if IsIsometric(N, Gen[j]) then
219               NumFound[j] += 1;
220               continue N;
221           end if;
222       end for;
223
224       Append(~Gen,N);
225       Append(~Explored, false);
226       Append(~NumFound, 1);
227       Append(~Minima, minimum);
228       Append(~NumShortest, shortest);
229       Append(~SizeAuto, auto);
230       m += 1/auto;
231       if m eq M then
232           break N;
233       end if;
234   end for;
235 end while;
236
237   PrintFileMagma(Sprintf("GenusSymbols/Gen_%o", GenSymbol(L)),
Gen : Overwrite := true);
238
239   return Gen;
240
241 end function;
242
243
244 function EnumerateGenusDeterminant(det, n, even)
245 // Input: det in N; n in N; boolean even that indicates whether
only even lattices shall be enumerated
246
247 // Output: Representatives of all isometry-classes belonging to a
genus of integral lattices with determinant det, dimension n, and
square-free level

```

```

248
249 if n eq 0 then
250     return [LatticeWithGram(Matrix(Rationals(),0,0,[]))];
251 end if;
252
253 if n eq 1 then
254     L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
255     Symbol := GenSymbol(L);
256     if even and not Symbol[1] eq 2 then return []; end if;
257     if not IsSquarefree(Level(L)) then return []; end if;
258     if even and IsDivisibleBy(Determinant(L), 2) then
259         if not Symbol[3][4] eq 2 then return []; end if;
260     end if;
261     return [L];
262 end if;
263
264 if n eq 2 then
265     Results := [];
266
267     for m in [1..Floor(1.155*Sqrt(det))] do
268         for a in [-m+1..m-1] do
269
270             if not IsDivisibleBy(det + a^2, m) then continue; end if;
271             b := Integers() ! ((det + a^2) / m);
272
273             if b lt m then continue; end if;
274             if even and not IsEven(b) then continue; end if;
275
276             Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
277             if not IsPositiveDefinite(Mat) then continue; end if;
278
279             L := LatticeWithGram(Mat);
280
281             if not IsSquarefree(Level(L)) then continue; end if;
282
283             Symbol := GenSymbol(L);
284             if even and not Symbol[1] eq 2 then continue; end if;
285             if even and IsDivisibleBy(Determinant(L), 2) then
286                 if not Symbol[3][4] eq 2 then continue; end if;
287             end if;
288
289             Append(~Results, L);
290         end for;
291     end for;
292
293     return ReduceByIsometry(Results);
294 end if;
295
296
297 Rat := RationalsAsNumberField();
298 Int := Integers(Rat);
299
300 primes := PrimeBasis(det);
301 exps := [Valuation(det, p) : p in primes];
302

```

```

303 IdealList := [];
304 if not 2 in primes then
305   Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
306 end if;
307
308 for i in [1..#primes] do
309   p := primes[i];
310   e := Abs(exps[i]);
311   if n eq e then
312     Append(~IdealList, <ideal<Int|p>, [[1,e]]>);
313   elif e eq 0 then
314     Append(~IdealList, <ideal<Int|p>, [[0,n]]>);
315   else
316     Append(~IdealList, <ideal<Int|p>, [[0,n-e],[1,e]]>);
317   end if;
318 end for;
319
320 "Constructing representatives";
321 try
322   Rep := LatticesWithGivenElementaryDivisors(Rat, n, IdealList);
323 catch e
324   print "Error while trying to construct a representative.
IdealList:";
325   IdealList;
326   return [];
327 end try;
328
329 Results := [];
330
331 for L in Rep do
332
333   LZ := ToZLattice(L);
334   if IsSquarefree(Level(LZ)) then
335     Symbol := GenSymbol(LZ);
336     if even and not Symbol[1] eq 2 then continue L; end if;
337     if even and IsDivisibleBy(det, 2) then
338       if not Symbol[3][4] eq 2 then continue L; end if;
339     end if;
340
341     Gen := EnumerateGenusOfRepresentative(LZ);
342     Results cat:= Gen;
343   end if;
344 end for;
345
346 return Results;
347
348 end function;
349
350
351 function EnumerateGenusSymbol(Symbol)
352 // Input: Genus-symbol Symbol of positive definite lattices of
square-free level; t in N
353
354 // Output: Representatives of all isometry-classes belonging to
the genus

```

```

355
356     try return eval Read(Sprintf("GenusSymbols/Gen_%o", Symbol));
357 catch e; end try;
358
359     n := Symbol[2];
360
361     if n eq 0 then
362         return [LatticeWithGram(Matrix(Rationals(), 0, 0, []))];
363     end if;
364
365     if n eq 1 then
366         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
367         L := LatticeWithGram(Matrix(Rationals(), 1, 1, [det]));
368         if GenSymbol(L) eq Symbol then
369             return [L];
370         end if;
371     end if;
372
373     if n eq 2 then
374         det := &*[Symbol[i][1]^Symbol[i][2] : i in [3..#Symbol]];
375
376         for m := 2 to Floor(1.155*Sqrt(det)) by 2 do
377             for a in [-m+1..m-1] do
378
379                 if not IsDivisibleBy(det + a^2, m) then continue; end if;
380                 b := Integers() ! ((det + a^2) / m);
381
382                 if b lt m then continue; end if;
383                 if not IsEven(b) then continue; end if;
384
385                 Mat := Matrix(Rationals(), 2, 2, [m,a,a,b]);
386                 if not IsPositiveDefinite(Mat) then continue; end if;
387
388                 L := LatticeWithGram(Mat);
389
390                 if not IsSquarefree(Level(L)) then continue; end if;
391
392                 if Symbol eq GenSymbol(L) then
393                     return EnumerateGenusOfRepresentative(L);
394                 end if;
395             end for;
396         end for;
397
398         return [];
399
400     end if;
401
402     Rat := RationalsAsNumberField();
403     Int := Integers(Rat);
404
405     IdealList := [];
406     if Symbol[3][1] ne 2 then
407         Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
408     end if;

```



```

409
410   for i in [3..#Symbol] do
411       p := Symbol[i][1];
412       np := Symbol[i][2];
413
414       if n eq np then
415           Append(~IdealList, <ideal<Int|p>, [[1,np]]>);
416       elif np eq 0 then
417           Append(~IdealList, <ideal<Int|p>, [[0,n]]>);
418       else
419           Append(~IdealList, <ideal<Int|p>, [[0,n-np],[1,np]]>);
420       end if;
421   end for;
422
423   "Constructing representatives";
424   try
425       Rep := LatticesWithGivenElementaryDivisors(Rat, n, IdealList);
426   catch e
427       print "Error while trying to construct a representative.
IdealList:";
428       IdealList;
429       return [];
430   end try;
431
432   for L in Rep do
433       LZ := ToZLattice(L);
434       if GenSymbol(LZ) eq Symbol then
435           Gen := EnumerateGenusOfRepresentative(LZ);
436           return Gen;
437       end if;
438   end for;
439
440   return [];
441
442 end function;
443
444
445 function SuperLatticesMagma(L, p, s, sigma)
446 // Input: Lattice L; Prime p; s in N; Automorphism sigma of L
447
448 // Output: All even sigma-invariant superlattices of L with index
p^s using magmas method
449
450
451   LD := PartialDual(L,p:Rescale:=false);
452
453   G := MatrixGroup<NumberOfRows(sigma), Integers() | sigma >;
454   den1 := Denominator(BasisMatrix(LD));
455   den2 := Denominator(InnerProductMatrix(LD));
456
457   A := LatticeWithBasis(G, Matrix(Integers(), den1*BasisMatrix
(LD)), Matrix(Integers(), den2^2*InnerProductMatrix(LD)));
458
459   SU := [];
460   SU := Sublattices(A, p : Levels := s, Limit := 100000);

```

```

461
462     if #SU eq 100000 then "List of sublattices is probably not
complete."; end if;
463
464     Results := [];
465
466     for S in SU do
467         M := 1/den1 * 1/den2 * S;
468
469         if Determinant(M)*p^(2*s) eq Determinant(L) then
470             Append(~Results, M);
471         end if;
472     end for;
473
474
475     return [L : L in Results | IsEven(L)];
476 end function;
477
478
479 function SuperLattices(L1, Lp, p, sigma1, sigmap)
480 // Input: Lattice L1; Lattice Lp; Prime p; Automorphism sigma of L
481
482 // Output: All even superlattices of L1 + Lp invariant under diag
(sigma1, sigmap) with index p^s using isometry-method
483
484 M := OrthogonalSum(L1, Lp);
485
486 L1Quot, phil := PartialDual(L1,p : Rescale:=false) / L1;
487 LpQuot, phip := PartialDual(Lp,p : Rescale:=false) / Lp;
488
489 m := #Generators(L1Quot);
490
491 philInv := Inverse(phil);
492 phipInv := Inverse(hiph);
493
494 G1 := ZeroMatrix(GF(p),m,m);
495 Gp := ZeroMatrix(GF(p),m,m);
496 for i in [1..m] do
497     for j in [1..m] do
498         G1[i,j] := GF(p) ! (p*InnerProduct(philInv(L1Quot.i),
philInv(L1Quot.j)));
499         Gp[i,j] := GF(p) ! (-p*InnerProduct(hiphInv(LpQuot.i),
hiphInv(LpQuot.j)));
500     end for;
501 end for;
502
503 V1 := KSpace(GF(p), m, G1);
504 Vp := KSpace(GF(p), m, Gp);
505
506 O1 := IsometryGroup(V1);
507
508 sigma1Quot := ZeroMatrix(GF(p),m,m);
509 for i in [1..m]
do
510     sigma1Quot[i] := Vector(GF(p), Eltseq(phil(philInv

```

```

(L1Quot.i)*Matrix(Rationals(),sigma1))));
511   end for;
512
513   sigmapQuot := ZeroMatrix(GF(p),m,m);
514   for i in [1..m]
do
515     sigmapQuot[i] := Vector(GF(p), Eltseq(php(phpInv
(LpQuot.i)*Matrix(Rationals(),sigmap))));
516   end for;
517
518   CL1Quot := Centralizer(O1, O1 ! sigma1Quot);
519
520   CL1 := Centralizer(AutomorphismGroup(L1), sigma1);
521
522   CL1ProjGens := [];
523   for g in Generators(CL1) do
524     gProj := ZeroMatrix(GF(p),m,m);
525     for i in [1..m] do
526       gProj[i] := Vector(GF(p), Eltseq(phi1(phi1Inv
(L1Quot.i)*Matrix(Rationals(), g))));
527     end for;
528     Append(~CL1ProjGens, gProj);
529   end for;
530
531   CL1Proj := MatrixGroup<m, GF(p) | CL1ProjGens>;
532
533   _, psi := IsIsometric(V1,Vp);
534
535   psi := MatrixOfIsomorphism(psi);
536   _, u := IsConjugate(O1, O1 ! sigma1Quot, O1 !
(psi*sigmapQuot*psi^(-1)));
537
538   phi0 := u*psi;
539
540   U, mapU := CL1Quot / CL1Proj;
541
542   LphiList := [];
543   for u in U do
544     phi := Inverse(mapU)(u)*phi0;
545
546     Gens := [];
547     for i in [1..m] do
548       x := phi1Inv(L1Quot.i);
549       y := phpInv(LpQuot ! Eltseq(phi[i]));
550       Append(~Gens, Eltseq(x) cat Eltseq(y));
551     end for;
552
553     Lphi := ext<M | Gens>;
554     Append(~LphiList,LatticeWithGram(LLGram(GramMatrix(Lphi))));
555   end for;
556
557   return [L : L in LphiList | IsEven(L)];
558 end function;
559
560

```

```

561
562 function SuperLatticesJuergens(L, p, s)
563 // Input: Lattice L; Prime p; s in N; t in N
564
565 // Output: All even superlattices of L with index p^s using
566 // juergens method
567
568 if s eq 0 then
569   return [L];
570 end if;
571
572 T, mapT := PartialDual(L, p:Rescale:=false) / L;
573 mapTinv := Inverse(mapT);
574
575 m := #Generators(T);
576 G := GramMatrix(L);
577 G_F := MatrixAlgebra(GF(p), m)!0;
578
579 for i:=1 to m do
580   for j:=1 to m do
581     G_F[i,j] := GF(p)!(p*InnerProduct(mapTinv(T.i), mapTinv
582 (T.j)));
583   end for;
584 end for;
585
586 V := KSpace(GF(p), m, G_F);
587 if not s le WittIndex(V) then
588   return [];
589 end if;
590
591 M1 := MaximalTotallyIsotropicSubspace(V);
592 M := sub< M1 | Basis(M1)[1..s] >;
593
594 O := IsometryGroup(V);
595 Aut := AutomorphismGroup(L:Decomposition:=true);
596
597 Gens := [];
598 for g in Generators(Aut) do
599   g_F := MatrixAlgebra(GF(p), m)!0;
600   for i:=1 to m do
601     g_F[i] := V!Vector(Eltseq(mapT(mapTinv(T!Eltseq
602 (V.i))*Matrix(Rationals(), g))));
603   end for;
604   Append(~Gens, g_F);
605 end for;
606
607 O_L := sub< O | Gens>;
608 mapS, S, Kernel := OrbitAction(O_L, Orbit(O, M));
609 Set := [Inverse(mapS)(i[2]) : i in OrbitRepresentatives(S)];
610 SuperLat := [CoordinateLattice(ext< L | [mapTinv(T!Eltseq
611 (x)) : x in Basis(W)] >) : W in Set];
612
613 return [L : L in SuperLat | IsEven(L)];
614
615 end function;
616

```

```

613
614 function MiPoQuotient(sigma, L, p);
615 // Input : Automorphism sigma of L; Lattice L
616
617 // Output: Minimal polynomial of the operation of sigma on the
        partial dual quotient  $L^{(\#), p} / L$ 
618
619     sigma := Matrix(Rationals(), sigma);
620     L := CoordinateLattice(L);
621     LD := PartialDual(L, p : Rescale := false);
622     _, phi := LD / L;
623     MiPo := PolynomialRing(GF(p)) ! 1;
624
625     B := [];
626
627     for i in [1..Rank(LD)] do
628
629         b := LD.i;
630         if b in sub<LD|L,B> then
631             continue;
632         end if;
633         Append(~B,b);
634
635         dep := false;
636         C := [Eltseq(phi(b))];
637         while not dep do
638             b := b*sigma;
639             Append(~C, Eltseq(phi(b)));
640             Mat := Matrix(GF(p),C);
641             if Dimension(Kernel(Mat)) gt 0 then
642                 dep := true;
643                 coeff := Basis(Kernel(Mat))[1];
644                 coeff /= coeff[#C];
645                 coeff := Eltseq(coeff);
646                 MiPo := LCM(MiPo, Polynomial(GF(p), coeff));
647             else
648                 Append(~B, b);
649             end if;
650         end while;
651     end for;
652
653     return MiPo;
654
655 end function;
656
657
658 function ConstructLattices(l, n)
659 // Input: Square-free l; n in N
660
661 // 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  $\text{ggT}(p,l) = 1$  and  $\mu_{\text{sigma}} / \Phi_m \mid (x^{(m/p)} - 1)$ 
662
663     Results := [];
664
665     min := ExtremalMinimum(l,n);

```

```

665
666     AutoTypes := AutomorphismTypes(l, Integers() ! (n/2), n, min);
667
668     for phim in [Integers() ! (n/2)+1 .. n] do
669
670         n1 := n - phim;
671         np := phim;
672
673         for m in EulerPhiInverse(phim) do
674
675             printf "m = %o\n", m;
676
677             for p in PrimeDivisors(m) do
678                 //printf "Testing p = %o\n", p;
679                 if Gcd(p,l) ne 1 then continue; end if;
680                 d := Integers() ! (m/p);
681                 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])];
682
683                 //printf "Have to check %o possible automorphism-types
\n", #PossibleTypes;
684
685                 for type in PossibleTypes do
686                     s := type[4];
687
688                     detp := p^s;
689                     for i := 5 to #type by 3 do
690                         detp *:= type[i]^type[i+2];
691                     end for;
692
693                     // Enumerate ideal-lattices over K(zeta_m) with given
determinant
694                     K<z> := CyclotomicField(m);
695                     Kpos := sub<K | z + z^(-1)>;
696
697                     A := ClassesModGalois(K);
698                     M, U, FundUnits := EmbeddingMatrix(K, Kpos);
699                     LpList := IdealLattices(detp, K, Kpos, A, M, U,
FundUnits, false);
700
701                     LpList := [L : L in LpList | Minimum(L) ge min];
702                     LpList := ReduceByIsometry(LpList);
703
704                     if s eq 0 then
705                         Results cat:= [L : L in LpList | Minimum(L) ge min];
706                         continue m;
707                     end if;
708
709                     for Lp in LpList do
710                         sigmapList := [c[3] : c in ConjugacyClasses
(AutomorphismGroup(Lp)) | MiPoQuotient(c[3], Lp, p) eq Polynomial
(GF(p), CyclotomicPolynomial(d))];
711                         if #sigmapList eq 0 then

```

```

712         continue Lp;
713     end if;
714     "Enumerate candidates for L_1";
715
716     K<z> := CyclotomicField(p);
717     Kpos := sub<K|z+1/z>;
718
719     if p eq 2 then
720
721         // In this case use the sublattice U of L_1 with U^
722         {#,2} = U
723         det1U := 1;
724         for i := 5 to #type by 3 do
725             det1U *:= type[i]^type[i+1];
726         end for;
727
728         UList := EnumerateGenusDeterminant(det1U, n1,
729         false);
730
731         L1List := &cat[SuperLatticesJuergens
732         (LatticeWithGram(2*GramMatrix(U)), p, Integers() ! ((n1 -
733         s)/2)) : U in UList | Dimension(U) eq 0 or Minimum(U) ge Ceiling
734         (min/2)];
735
736         L1List := [L : L in L1List | Dimension(L) eq 0 or
737         (IsEven(L) and Minimum(L) ge min)];
738
739         elif IsPrime(l) then
740             // In this case the genus symbol of L_1 is known
741             and allows for a more efficient enumeration
742             k1 := type[6];
743             kp := type[7];
744
745             if p le 3 then
746                 f := 1;
747             else
748                 f := InertiaDegree(Factorization(ideal<Integers
749                 (Kpos) | l>)[1][1]);
750             end if;
751             deltap := (-1)^(Integers() ! (kp/f + (p-1)/2 *
752             (Binomial(Integers() ! (np / (p-1) + 1), 2) + Binomial(s, 2))));
753             delta1 := deltap * (-1)^(Integers() ! (s*(p-1)/2));
754
755             if l eq 2 then
756                 if IsDivisibleBy(np + s*(p-1), 8) then
757                     epsilonp := deltap;
758                 else
759                     epsilonp := -deltap;
760                 end if;
761
762                 if IsDivisibleBy(n, 8) then
763                     epsilon := 1;
764                 else
765                     epsilon := -1;
766                 end if;

```

```

757         else
758             epsilonp := (-1)^(Integers() ! (kp / f +
(l-1)/2*Binomial(kp,2)));
759
760             if IsDivisibleBy(n*(l+1), 16) then
761                 epsilon := 1;
762             else
763                 epsilon := -1;
764             end if;
765         end if;
766
767         epsilon1 := epsilonp*epsilon;
768
769         Sym1 := [* 2, n1 *];
770         if l eq 2 then
771             Append(~Sym1, <2, k1, epsilon1, 2, 0>);
772             Append(~Sym1, <p, s, delta1>);
773         else
774             if l lt p then
775                 Append(~Sym1, <l, k1, epsilon1>);
776                 Append(~Sym1, <p, s, delta1>);
777             else
778                 Append(~Sym1, <p, s, delta1>);
779                 Append(~Sym1, <l, k1, epsilon1>);
780             end if;
781         end if;
782
783         L1List := [L : L in EnumerateGenusSymbol(Sym1) |
Dimension(L) eq 0 or (IsEven(L) and Minimum(L) ge min)];
784
785         else
786
787             det1 := p^s;
788             for i := 5 to #type by 3 do
789                 det1 *= type[i]^type[i+1];
790             end for;
791
792             L1List := [L : L in EnumerateGenusDeterminant
(det1, n1, true) | Dimension(L) eq 0 or Minimum(L) ge min];
793
794         end if;
795
796         for L1 in L1List do
797             sigmallist := [c[3] : c in ConjugacyClasses
(AutomorphismGroup(L1)) | MiPoQuotient(c[3], L1, p) eq Polynomial
(GF(p), CyclotomicPolynomial(d)) and Degree(MinimalPolynomial(c
[3])) le EulerPhi(d)];
798             if #sigmallist eq 0 then
799                 continue L1;
800             end if;
801
802             "Constructing superlattices";
803
804             if <l,n> in [] then
805                 for signal in sigmallist do
806                     for sigmap in sigmapList do

```



```

807             LList cat:= SuperLatticesMagma
(CoordinateLattice(OrthogonalSum(L1,Lp)), p, s, DiagonalJoin
(sigm1, sigmap));
808         end for;
809     end for;
810     elif <l,n> in
[<7,18>,<7,20>,<1,24>,<2,24>,<5,24>] then
811         LList := [];
812         for sigm1 in sigm1List do
813             for sigmap in sigmapList do
814                 LList cat:= SuperLattices(CoordinateLattice
(L1), CoordinateLattice(Lp), p, sigm1, sigmap);
815             end for;
816         end for;
817     else
818         LList := SuperLatticesJuergens(CoordinateLattice
(OrthogonalSum(L1,Lp)),p,s);
819     end if;
820
821     Results cat:= [L : L in LList | Minimum(L) ge min];
822 end for;
823 end for;
824 end for;
825 end for;
826 end for;
827 end for;
828
829 return ReduceByIsometry(Results);
830
831 end function;
832
833
834 procedure MainLoop()
835     for n := 2 to 36 by 2
836     do
837         for l in [1,2,3,5,6,7,11,14,15,23] do
838             if l eq 1 and n in [2,4,6] then continue; end if;
839             if l eq 2 and n eq 2 then continue; end if;
840             if l eq 11 and n in [20,24,28,30,32,34,36] then continue;
841             end if;
842             if l eq 23 and n ge 6 then continue; end if;
843             printf "dim = %0, l = %0\n", n, l;
844             Results := ConstructLattices(l, n);
845             ModList := [L : L in Results | IsModular(L, l)];
846             StrongModList := [L : L in Results | IsStronglyModular
(L,l)];
847             PrintFileMagma(sprintf("SubidealLattices/%0-Modular/%0-
Dimensional", l, n), Results : Overwrite := true);
848             PrintFileMagma(sprintf("SubidealLattices/%0-Modular/%0-
DimensionalModular", l, n), ModList : Overwrite := true);
849             PrintFileMagma(sprintf("SubidealLattices/%0-Modular/%0-
DimensionalStronglyModular", l, n), StrongModList : Overwrite :=
true);

```

```
849         if #Results gt 0 then
850             printf "\n\n-----n = %0, l = %0: %0 lattices found! %
o of them are modular and %0 are strongly modular-----\n\n",
n, l, #Results, #ModList, #StrongModList;
851         end if;
852     end for;
853 end for;
854 end procedure;
```