

```

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      for s in [0..Min(n1, Integers() ! (np/
  (p-1)))] do
30        if not IsDivisibleBy(s - Integers() !
  (np / (p-1)), 2) then continue s; end if;
31        if p eq 2 and not IsDivisibleBy(s, 2)
  then continue s; end if;
32
33        if l eq 1 then
34          if n1 gt 0 then
35            Gamma1 := t/p^(s/n1);
36            if Gamma1 gt HermiteBounds
  [n1] + 0.1 then continue; end if;
37          end if;
38
39          if np gt 0 then

```

```

40      Gammap := t/p^(s/np);
41      if Gammap gt HermiteBounds
42  [np] + 0.1 then continue; end if;
43      end if;
44      type := <p, n1, np, s>;
45      Append(~Results, type);
46      else
47      for kp in CartesianProduct([[2*f
48  [i]*j : j in [0..Floor(Min(np,k)/(2*f[i]))] : i in
49  [1..#f]]) do
50
51      k1 := [k - kp[i] : i in
52  [1..#kp]];
53
54      for i in [1..#kp] do
55      if k1[i] gt Min(n1,k)
56  then continue kp; end if;
57      if not IsDivisibleBy(k1
58  [i] - k, 2) then continue kp; end if;
59      if not IsDivisibleBy(kp
60  [i], 2) then continue kp; end if;
61      end for;
62
63      if n1 gt 0 then
64      Gammal := p^s;
65      for i in [1..#lFactors] do
66      Gammal *:= lFactors
67  [i]^k1[i];
68
69      end for;
70      Gammal := t / Gammal^(1/
71  n1);
72
73      if Gammal gt HermiteBounds
74  [n1] + 0.1 then continue; end if;
75      end if;
76
77      if np gt 0 then
78      Gammap := p^s;
79      for i in [1..#lFactors] do
80      Gammap *:= lFactors
81  [i]^kp[i];
82
83      end for;
84      Gammap := t / Gammap^(1/
85  np);
86  end if;
87  end for;
88  end if;
89  end for;
90  end if;
91  end for;
92  end if;
93  end for;
94  end if;
95  end for;
96  end if;
97  end for;
98  end if;
99  end for;
100 end if;
101 end for;
102 end if;
103 end for;
104 end if;
105 end for;
106 end if;
107 end for;
108 end if;
109 end for;
110 end if;
111 end for;
112 end if;
113 end for;
114 end if;
115 end for;
116 end if;
117 end for;
118 end if;
119 end for;
120 end if;
121 end for;
122 end if;
123 end for;
124 end if;
125 end for;
126 end if;
127 end for;
128 end if;
129 end for;
130 end if;
131 end for;
132 end if;
133 end for;
134 end if;
135 end for;
136 end if;
137 end for;
138 end if;
139 end for;
140 end if;
141 end for;
142 end if;
143 end for;
144 end if;
145 end for;
146 end if;
147 end for;
148 end if;
149 end for;
150 end if;
151 end for;
152 end if;
153 end for;
154 end if;
155 end for;
156 end if;
157 end for;
158 end if;
159 end for;
160 end if;
161 end for;
162 end if;
163 end for;
164 end if;
165 end for;
166 end if;
167 end for;
168 end if;
169 end for;
170 end if;
171 end for;
172 end if;
173 end for;
174 end if;
175 end for;
176 end if;
177 end for;
178 end if;
179 end for;
180 end if;
181 end for;
182 end if;
183 end for;
184 end if;
185 end for;
186 end if;
187 end for;
188 end if;
189 end for;
190 end if;
191 end for;
192 end if;
193 end for;
194 end if;
195 end for;
196 end if;
197 end for;
198 end if;
199 end for;
200 end if;
201 end for;
202 end if;
203 end for;
204 end if;
205 end for;
206 end if;
207 end for;
208 end if;
209 end for;
210 end if;
211 end for;
212 end if;
213 end for;
214 end if;
215 end for;
216 end if;
217 end for;
218 end if;
219 end for;
220 end if;
221 end for;
222 end if;
223 end for;
224 end if;
225 end for;
226 end if;
227 end for;
228 end if;
229 end for;
230 end if;
231 end for;
232 end if;
233 end for;
234 end if;
235 end for;
236 end if;
237 end for;
238 end if;
239 end for;
240 end if;
241 end for;
242 end if;
243 end for;
244 end if;
245 end for;
246 end if;
247 end for;
248 end if;
249 end for;
250 end if;
251 end for;
252 end if;
253 end for;
254 end if;
255 end for;
256 end if;
257 end for;
258 end if;
259 end for;
260 end if;
261 end for;
262 end if;
263 end for;
264 end if;
265 end for;
266 end if;
267 end for;
268 end if;
269 end for;
270 end if;
271 end for;
272 end if;
273 end for;
274 end if;
275 end for;
276 end if;
277 end for;
278 end if;
279 end for;
280 end if;
281 end for;
282 end if;
283 end for;
284 end if;
285 end for;
286 end if;
287 end for;
288 end if;
289 end for;
290 end if;
291 end for;
292 end if;
293 end for;
294 end if;
295 end for;
296 end if;
297 end for;
298 end if;
299 end for;
300 end if;
301 end for;
302 end if;
303 end for;
304 end if;
305 end for;
306 end if;
307 end for;
308 end if;
309 end for;
310 end if;
311 end for;
312 end if;
313 end for;
314 end if;
315 end for;
316 end if;
317 end for;
318 end if;
319 end for;
320 end if;
321 end for;
322 end if;
323 end for;
324 end if;
325 end for;
326 end if;
327 end for;
328 end if;
329 end for;
330 end if;
331 end for;
332 end if;
333 end for;
334 end if;
335 end for;
336 end if;
337 end for;
338 end if;
339 end for;
340 end if;
341 end for;
342 end if;
343 end for;
344 end if;
345 end for;
346 end if;
347 end for;
348 end if;
349 end for;
350 end if;
351 end for;
352 end if;
353 end for;
354 end if;
355 end for;
356 end if;
357 end for;
358 end if;
359 end for;
360 end if;
361 end for;
362 end if;
363 end for;
364 end if;
365 end for;
366 end if;
367 end for;
368 end if;
369 end for;
370 end if;
371 end for;
372 end if;
373 end for;
374 end if;
375 end for;
376 end if;
377 end for;
378 end if;
379 end for;
380 end if;
381 end for;
382 end if;
383 end for;
384 end if;
385 end for;
386 end if;
387 end for;
388 end if;
389 end for;
390 end if;
391 end for;
392 end if;
393 end for;
394 end if;
395 end for;
396 end if;
397 end for;
398 end if;
399 end for;
400 end if;
401 end for;
402 end if;
403 end for;
404 end if;
405 end for;
406 end if;
407 end for;
408 end if;
409 end for;
410 end if;
411 end for;
412 end if;
413 end for;
414 end if;
415 end for;
416 end if;
417 end for;
418 end if;
419 end for;
420 end if;
421 end for;
422 end if;
423 end for;
424 end if;
425 end for;
426 end if;
427 end for;
428 end if;
429 end for;
430 end if;
431 end for;
432 end if;
433 end for;
434 end if;
435 end for;
436 end if;
437 end for;
438 end if;
439 end for;
440 end if;
441 end for;
442 end if;
443 end for;
444 end if;
445 end for;
446 end if;
447 end for;
448 end if;
449 end for;
450 end if;
451 end for;
452 end if;
453 end for;
454 end if;
455 end for;
456 end if;
457 end for;
458 end if;
459 end for;
460 end if;
461 end for;
462 end if;
463 end for;
464 end if;
465 end for;
466 end if;
467 end for;
468 end if;
469 end for;
470 end if;
471 end for;
472 end if;
473 end for;
474 end if;
475 end for;
476 end if;
477 end for;
478 end if;
479 end for;
480 end if;
481 end for;
482 end if;
483 end for;
484 end if;
485 end for;
486 end if;
487 end for;
488 end if;
489 end for;
490 end if;
491 end for;
492 end if;
493 end for;
494 end if;
495 end for;
496 end if;
497 end for;
498 end if;
499 end for;
500 end if;
501 end for;
502 end if;
503 end for;
504 end if;
505 end for;
506 end if;
507 end for;
508 end if;
509 end for;
510 end if;
511 end for;
512 end if;
513 end for;
514 end if;
515 end for;
516 end if;
517 end for;
518 end if;
519 end for;
520 end if;
521 end for;
522 end if;
523 end for;
524 end if;
525 end for;
526 end if;
527 end for;
528 end if;
529 end for;
530 end if;
531 end for;
532 end if;
533 end for;
534 end if;
535 end for;
536 end if;
537 end for;
538 end if;
539 end for;
540 end if;
541 end for;
542 end if;
543 end for;
544 end if;
545 end for;
546 end if;
547 end for;
548 end if;
549 end for;
550 end if;
551 end for;
552 end if;
553 end for;
554 end if;
555 end for;
556 end if;
557 end for;
558 end if;
559 end for;
560 end if;
561 end for;
562 end if;
563 end for;
564 end if;
565 end for;
566 end if;
567 end for;
568 end if;
569 end for;
570 end if;
571 end for;
572 end if;
573 end for;
574 end if;
575 end for;
576 end if;
577 end for;
578 end if;
579 end for;
580 end if;
581 end for;
582 end if;
583 end for;
584 end if;
585 end for;
586 end if;
587 end for;
588 end if;
589 end for;
590 end if;
591 end for;
592 end if;
593 end for;
594 end if;
595 end for;
596 end if;
597 end for;
598 end if;
599 end for;

```

```

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

```

```

112
113     return Results;
114
115 end function;
116
117
118 function EnumerateGenusOfRepresentative(L)
119 // Input: Lattice L, t in N
120
121 // Output: List of all representatives of isometry-
122 // classes in the genus of L
123
124     "Enumerate genus of representative";
125     try return eval Read(Sprintf("GenusSymbols/Gen_%
126 o", GenSymbol(L))); catch e; end try;
127
128     if Dimension(L) le 4 then
129         Gen := GenusRepresentatives(L);
130         ZGen := [];
131         for M in Gen do
132             if Type(M) eq Lat then
133                 Append(~ZGen, LLL(M));
134             else
135                 Append(~ZGen, LatticeWithGram(LLLGram
136 (Matrix(Rationals(), GramMatrix(SimpleLattice(M))))));
137             end if;
138         end for;
139         PrintFileMagma(Sprintf("GenusSymbols/Gen_%
140 o", GenSymbol(L)), ZGen : Overwrite := true);
141         return ZGen;
142     end if;
143
144     M := Mass(L);
145     Gen := [L];
146     Explored := [false];
147     NumFound := [1];
148     Minima := [Minimum(L)];
149     NumShortest := [#ShortestVectors(L)];
150     SizeAuto := [#AutomorphismGroup(L)];
151     m := 1 / SizeAuto[1];
152
153     p := 2;
154
155     t0 := Realtime();
156
157     while m lt M do
158         //printf "So far %o classes found.

```

```

Difference to actual mass is %. \n", #Gen, M-m;
155     if Realtime(t0) ge 120*60 then
156         printf "2 hours have elapsed and not the
whole genus was explored. Remaining difference to
actual mass is %. % classes were found so far.\n", M-
m, #Gen;
157         return Gen;
158     end if;
159
160     RareFound := [];
161     MinCount := Infinity();
162
163     if &and(Explored) then
164         //"All explored. Going to next prime.";
165         Explored := [false : x in Explored];
166         p := NextPrime(p);
167     end if;
168
169     for i in [1..#Gen] do
170         if not Explored[i] then
171             if NumFound[i] lt MinCount then
172                 RareFound := [i];
173                 MinCount := NumFound[i];
174             elif NumFound[i] eq MinCount then
175                 Append(~RareFound, i);
176             end if;
177         end if;
178     end for;
179
180     i := RareFound[Random(1, #RareFound)];
181
182     Neigh := [CoordinateLattice(N) : N in
Neighbours(Gen[i], p)];
183     Explored[i] := true;
184
185     for N in Neigh do
186
187         auto := #AutomorphismGroup(N);
188         if auto lt 1/(M-m) then continue; end if;
189
190         minimum := Minimum(N);
191         shortest := #ShortestVectors(N);
192
193         for j in [1..#Gen] do
194
195             if minimum ne Minima[j] then
196                 continue j;
197             end if;

```

```

198
199         if shortest ne NumShortest[j] then
200             continue j;
201         end if;
202
203         if auto ne SizeAuto[j] then
204             continue j;
205         end if;
206
207         if IsIsometric(N, Gen[j]) then
208             NumFound[j] += 1;
209             continue N;
210         end if;
211     end for;
212
213     Append(~Gen,N);
214     Append(~Explored, false);
215     Append(~NumFound, 1);
216     Append(~Minima, minimum);
217     Append(~NumShortest, shortest);
218     Append(~SizeAuto, auto);
219     m += 1/auto;
220     if m eq M then
221         break N;
222     end if;
223 end for;
224 end while;
225
226     PrintFileMagma(Sprintf("GenusSymbols/Gen_%
o",GenSymbol(L)), Gen : Overwrite := true);
227
228     return Gen;
229
230 end function;
231
232
233 function EnumerateGenusDeterminant(det, n, even)
234 // Input: det in N; n in N; boolean even that
indicates whether only even lattices shall be
enumerated
235
236 // Output: Representatives of all isometry-classes
belonging to a genus of integral lattices with
determinant det, dimension n, and square-free level
237
238     if n eq 0 then
239         return [LatticeWithGram(Matrix(Rationals
(),0,0,[]))];

```

```

240     end if;
241
242     if n eq 1 then
243         L := LatticeWithGram(Matrix(Rationals(), 1,
244 1, [det]));
245         Symbol := GenSymbol(L);
246         if even and not Symbol[1] eq 2 then return
247 []; end if;
248         if not IsSquarefree(Level(L)) then return [];
249 end if;
250         if even and IsDivisibleBy(Determinant(L), 2)
251 then
252             if not Symbol[3][4] eq 2 then return [];
253 end if;
254         end if;
255         return [L];
256     end if;
257
258     if n eq 2 then
259         Results := [];
260
261         for m in [1..Floor(1.155*Sqrt(det))] do
262             for a in [-m+1..m-1] do
263
264                 if not IsDivisibleBy(det + a^2, m)
265 then continue; end if;
266                 b := Integers() ! ((det + a^2) / m);
267
268                 if b lt m then continue; end if;
269                 if even and not IsEven(b) then
270 continue; end if;
271
272                 Mat := Matrix(Rationals(), 2, 2,
273 [m,a,a,b]);
274                 if not IsPositiveDefinite(Mat) then
275 continue; end if;
276
277                 L := LatticeWithGram(Mat);
278
279                 if not IsSquarefree(Level(L)) then
280 continue; end if;
281
282                 Symbol := GenSymbol(L);
283                 if even and not Symbol[1] eq 2 then
284 continue; end if;
285                 if even and IsDivisibleBy(Determinant
286 (L), 2) then
287                     if not Symbol[3][4] eq 2 then

```

```

        continue; end if;
276         end if;
277
278         Append(~Results, L);
279     end for;
280 end for;
281
282     return ReduceByIsometry(Results);
283 end if;
284
285
286 Rat := RationalsAsNumberField();
287 Int := Integers(Rat);
288
289 primes := PrimeBasis(det);
290 exps := [Valuation(det, p) : p in primes];
291
292 IdealList := [];
293 if not 2 in primes then
294     Append(~IdealList, <ideal<Int|2>, [[0,n]]>);
295 end if;
296
297 for i in [1..#primes] do
298     p := primes[i];
299     e := Abs(exps[i]);
300     if n eq e then
301         Append(~IdealList, <ideal<Int|p>,
302             [[1,e]]>);
303     elif e eq 0 then
304         Append(~IdealList, <ideal<Int|p>,
305             [[0,n]]>);
306     else
307         Append(~IdealList, <ideal<Int|p>, [[0,n-
308 e],[1,e]]>);
309     end if;
310 end for;
311
312 "Constructing representatives";
313 try
314     Rep := LatticesWithGivenElementaryDivisors
315     (Rat, n, IdealList);
316 catch e
317     print "Error while trying to construct a
318     representative. IdealList:";
319     IdealList;
320     return [];
321 end try;

```



```

318     Results := [];
319
320     for L in Rep do
321
322         LZ := ToZLattice(L);
323         if IsSquarefree(Level(LZ)) then
324             Symbol := GenSymbol(LZ);
325             if even and not Symbol[1] eq 2 then
326                 continue L; end if;
327             if even and IsDivisibleBy(det, 2) then
328                 if not Symbol[3][4] eq 2 then continue
329                     L; end if;
330                 end if;
331             Gen := EnumerateGenusOfRepresentative(LZ);
332             Results cat:= Gen;
333         end if;
334     end for;
335
336     return Results;
337 end function;
338
339
340 function EnumerateGenusSymbol(Symbol)
341 // Input: Genus-symbol Symbol of positive definite
342 //         lattices of square-free level; t in N
343
344 // Output: Representatives of all isometry-classes
345 //         belonging to the genus
346
347     try return eval Read(Sprintf("GenusSymbols/Gen_%
348 o", Symbol)); catch e; end try;
349
350     n := Symbol[2];
351
352     if n eq 0 then
353         return [LatticeWithGram(Matrix(Rationals
354             (),0,0,[]))];
355     end if;
356
357     if n eq 1 then
358         det := &*[Symbol[i][1]^Symbol[i][2] : i in
359             [3..#Symbol]];
360         L := LatticeWithGram(Matrix(Rationals(), 1,
361             1, [det]));
362         if GenSymbol(L) eq Symbol then

```

```

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

```

```

398
399     for i in [3..#Symbol] do
400         p := Symbol[i][1];
401         np := Symbol[i][2];
402
403         if n eq np then
404             Append(~IdealList, <ideal<Int|p>,
[[1,np]]>);
405         elif np eq 0 then
406             Append(~IdealList, <ideal<Int|p>,
[[0,n]]>);
407         else
408             Append(~IdealList, <ideal<Int|p>, [[0,n-
np],[1,np]]>);
409         end if;
410     end for;
411
412     "Constructing representatives";
413     try
414         Rep := LatticesWithGivenElementaryDivisors
(Rat, n, IdealList);
415     catch e
416         print "Error while trying to construct a
representative. IdealList:";
417         IdealList;
418         return [];
419     end try;
420
421     for L in Rep do
422         LZ := ToZLattice(L);
423         if GenSymbol(LZ) eq Symbol then
424             Gen := EnumerateGenusOfRepresentative(LZ);
425             return Gen;
426         end if;
427     end for;
428
429     return [];
430
431 end function;
432
433
434 function SuperLatticesMagma(L, p, s, sigma)
435 // Input: Lattice L; Prime p; s in N; Automorphism
sigma of L
436
437 // Output: All even sigma-invariant superlattices of L
with index p^s using magmas method

```

```

438
439
440     LD := PartialDual(L,p:Rescale:=false);
441
442     G := MatrixGroup<NumberOfRows(sigma), Integers()
| sigma >;
443     den1 := Denominator(BasisMatrix(LD));
444     den2 := Denominator(InnerProductMatrix(LD));
445
446     A := LatticeWithBasis(G, Matrix(Integers(),
den1*BasisMatrix(LD)), Matrix(Integers(),
den2^2*InnerProductMatrix(LD)));
447
448     SU := [];
449     SU := Sublattices(A, p : Levels := s, Limit :=
100000);
450
451     if #SU eq 100000 then "List of sublattices is
probably not complete."; end if;
452
453     Results := [];
454
455     for S in SU do
456
457         M := 1/den1 * 1/den2 * S;
458
459         if Determinant(M)*p^(2*s) eq Determinant(L)
then
460             Append(~Results, M);
461         end if;
462     end for;
463
464     return [L : L in Results | IsEven(L)];
465 end function;
466
467
468 function SuperLattices(L1, Lp, p, sigma1, sigmap)
469 // Input: Lattice L1; Lattice Lp; Prime p;
Automorphism sigma of L
470
471 // Output: All even sigma-invariant superlattices of L
with index p^s and minimum at least t using magmas
method
472
473     M := OrthogonalSum(L1, Lp);
474
475     L1Quot, phil := PartialDual(L1,p :
Rescale:=false) / L1;

```

```

476     LpQuot, phip := PartialDual(Lp,p :
Rescale:=false) / Lp;
477
478     m := #Generators(L1Quot);
479
480     philInv := Inverse(phil);
481     phipInv := Inverse(hiph);
482
483     G1 := ZeroMatrix(GF(p),m,m);
484     Gp := ZeroMatrix(GF(p),m,m);
485     for i in [1..m] do
486         for j in [1..m] do
487             G1[i,j] := GF(p) ! (p*InnerProduct(philInv
(L1Quot.i), philInv(L1Quot.j)));
488             Gp[i,j] := GF(p) ! (-p*InnerProduct
(hiphInv(LpQuot.i), phipInv(LpQuot.j)));
489         end for;
490     end for;
491
492     V1 := KSpace(GF(p), m, G1);
493     Vp := KSpace(GF(p), m, Gp);
494
495     O1 := IsometryGroup(V1);
496
497     sigma1Quot := ZeroMatrix(GF(p),m,m);
498     for i in [1..m]
do
499         sigma1Quot[i] := Vector(GF(p), Eltseq(phil
(philInv(L1Quot.i)*Matrix(Rationals(),sigma1))));
500     end for;
501
502     sigmaMapQuot := ZeroMatrix(GF(p),m,m);
503     for i in [1..m]
do
504         sigmaMapQuot[i] := Vector(GF(p), Eltseq(hiph
(hiphInv(LpQuot.i)*Matrix(Rationals(),sigmaMap))));
505     end for;
506
507     CL1Quot := Centralizer(O1, O1 ! sigma1Quot);
508
509     CL1 := Centralizer(AutomorphismGroup(L1), sigma1);
510
511     CL1ProjGens := [];
512     for g in Generators(CL1) do
513         gProj := ZeroMatrix(GF(p),m,m);
514         for i in [1..m] do
515             gProj[i] := Vector(GF(p), Eltseq(phil

```

```

        (phiInv(L1Quot.i)*Matrix(Rationals(), g))));
516         end for;
517         Append(~CL1ProjGens, gProj);
518     end for;
519
520     CL1Proj := MatrixGroup<m, GF(p) | CL1ProjGens>;
521
522     _, psi := IsIsometric(V1,Vp);
523
524     psi := MatrixOfIsomorphism(psi);
525     _, u := IsConjugate(01, 01 ! sigma1Quot, 01 !
    (psi*sigma1Quot*psi^(-1)));
526
527     phi0 := u*psi;
528
529     U, mapU := CL1Quot / CL1Proj;
530
531     LphiList := [];
532     for u in U do
533         phi := Inverse(mapU)(u)*phi0;
534
535         Gens := [];
536         for i in [1..m] do
537             x := phiInv(L1Quot.i);
538             y := phiInv(LpQuot ! Eltseq(phi[i]));
539             Append(~Gens, Eltseq(x) cat Eltseq(y));
540         end for;
541
542         Lphi := ext<M | Gens>;
543         Append(~LphiList, LatticeWithGram(LLLGram
    (GramMatrix(Lphi))));
544     end for;
545
546     return [L : L in LphiList | IsEven(L)];
547 end function;
548
549
550
551 function SuperLatticesJuergens(L, p, s)
552 // Input: Lattice L; Prime p; s in N; t in N
553
554 // Output: All even superlattices of L with index p^s
    using juergens method
555
556     if s eq 0 then
557         return [L];
558     end if;
559

```

```

560     T,mapT:=PartialDual(L,p:Rescale:=false) / L;
561     mapTinv := Inverse(mapT);
562
563     m:=#Generators(T);
564     G:=GramMatrix(L);
565     G_F:=MatrixAlgebra(GF(p),m)!0;
566
567     for i:=1 to m do
568         for j:=1 to m do
569             G_F[i,j]:=GF(p)!(p*InnerProduct(mapTinv
(T.i),mapTinv(T.j)));
570         end for;
571     end for;
572
573     V:=KSpace(GF(p),m,G_F);
574     if not s le WittIndex(V) then
575         return [];
576     end if;
577
578     M1:=MaximalTotallyIsotropicSubspace(V);
579     M:=sub< M1 | Basis(M1)[1..s] >;
580
581     O:=IsometryGroup(V);
582     Aut:=AutomorphismGroup(L:Decomposition:=true);
583
584     Gens:=[];
585     for g in Generators(Aut) do
586         g_F:=MatrixAlgebra(GF(p),m)!0;
587         for i:=1 to m do
588             g_F[i]:=V!Vector(Eltseq(mapT(mapTinv(T!
Eltseq(V.i))*Matrix(Rationals(),g))));
589         end for;
590         Append(~Gens,g_F);
591     end for;
592
593     O_L:=sub< O | Gens>;
594     mapS,S,Kernel:=OrbitAction(O_L,Orbit(O,M));
595     Set:=[Inverse(mapS)(i[2]) : i in
OrbitRepresentatives(S)];
596     SuperLat := [CoordinateLattice(ext< L | [mapTinv(T!
Eltseq(x)) : x in Basis(W)] >) : W in Set];
597
598     return [L : L in SuperLat | IsEven(L)];
599
600 end function;
601
602
603 function ConstructLattices(l, n)

```



```

642
643         A := ClassesModGalois(K);
644         M, U, FundUnits := EmbeddingMatrix
(K, Kpos);
645         LpList := IdealLattices(detp, K,
Kpos, A, M, U, FundUnits, false);
646
647         LpList := [L : L in LpList |
Minimum(L) ge min];
648         LpList := ReduceByIsometry
(LpList);
649
650         for Lp in LpList do
651             sigmapList := [c[3] : c in
ConjugacyClasses(AutomorphismGroup(Lp)) | MiPoQuotient
(c[3], Lp, p) eq Polynomial(GF(p), CyclotomicPolynomial
(d))];
652             if #sigmapList eq 0 then
653                 continue Lp;
654             end if;
655             "Enumerate candidates for L_1";
656
657             if p eq 2 then
658
659                 // In this case use the
sublattice U of L_1 with U^{#,2} = U
660                 det1U := 1;
661                 for i := 5 to #type by 3
do
662                     det1U *:= type[i]^type
[i+1];
663                 end for;
664
665                 UList :=
EnumerateGenusDeterminant(det1U, n1, false);
666
667                 L1List := &cat
[SuperLatticesJuergens(LatticeWithGram(2*GramMatrix
(U)), p, Integers() ! ((n1 - s)/2)) : U in UList |
Dimension(U) eq 0 or Minimum(U) ge Ceiling(min/2)];
668                 L1List := [L : L in
L1List | Dimension(L) eq 0 or (IsEven(L) and Minimum
(L) ge min)];
669
670                 elif IsPrime(l) then
671                     // In this case the genus
symbol of L_1 is known and allows for a more

```

efficient enumeration

```

672         k1 := type[6];
673         kp := type[7];
674
675         f := InertiaDegree
676         (Factorization(ideal<Integers(Kpos) | l>)[1][1]);
677         deltap := (-1)^(Integers
678         () ! (kp/f + (p-1)/2 * (Binomial(Integers() ! (np /
679         (p-1) + 1), 2) + Binomial(s, 2)));
680         delta1 := deltap * (-1)^
681         (Integers() ! (s*(p-1)/2));
682
683         if l eq 2 then
684             if IsDivisibleBy(np +
685             s*(p-1), 8) then
686                 epsilonp :=
687                 deltap;
688             else
689                 epsilonp := -
690                 deltap;
691             end if;
692             if IsDivisibleBy(n,
693             8) then
694                 epsilon := 1;
695             else
696                 epsilon := -1;
697             end if;
698         else
699             epsilonp := (-1)^
700             (Integers() ! (kp / f + (l-1)/2*Binomial(kp,2)));
701             if IsDivisibleBy(n*(l
702             +1), 16) then
703                 epsilon := 1;
704             else
705                 epsilon := -1;
706             end if;
707         end if;
708         epsilon1 :=
709         epsilonp*epsilon;
710
711         Sym1 := [* 2, n1 *];
712         if l eq 2 then
713             Append(~Sym1, <2, k1,
714             epsilon1, 2, 0>);
715             Append(~Sym1, <p, s,

```

```

    delta1>);
707         else
708             if l lt p then
709                 Append(~Sym1,
710                     <l, k1, epsilon1>);
711                 Append(~Sym1,
712                     <p, s, delta1>);
713                 Append(~Sym1,
714                     <l, k1, epsilon1>);
715             end if;
716         end if;
717         LlList := [L : L in
EnumerateGenusSymbol(Sym1) | Dimension(L) eq 0 or
(IsEven(L) and Minimum(L) ge min)];
718
719         else
720
721             det1 := p^s;
722             for i := 5 to #type by 3
do
723                 det1 := type[i]^type
724                 [i+1];
725             end for;
726             LlList := [L : L in
EnumerateGenusDeterminant(det1, n1, true) | Dimension
(L) eq 0 or Minimum(L) ge min];
727
728         end if;
729
730         for L1 in LlList do
731             sigmaList := [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)];
732             if #sigmaList eq 0 then
733                 continue L1;
734             end if;
735
736             "Constructing superlattices";
737
738             if <l,n> in [] then

```

```

739         for signal in signalList
740             do
741                 for sigmap in
742                     sigmapList do
743                         LList cat:=
744                         SuperLatticesMagma(CoordinateLattice(OrthogonalSum
745                         (L1,Lp)), p, s, DiagonalJoin(signal, sigmap));
746                         end for;
747                         end for;
748                         elif <l,n> in [<1,24>]
749                         then
750                             LList := [];
751                             for signal in signalList
752                                 do
753                                     for sigmap in
754                                         sigmapList do
755                                             LList cat:=
756                                             SuperLattices(CoordinateLattice(L1), CoordinateLattice
757                                             (Lp), p, signal, sigmap);
758                                             end for;
759                                             end for;
760                                             else
761                                             LList :=
762                                             SuperLatticesJuergens(CoordinateLattice(OrthogonalSum
763                                             (L1,Lp)),p,s);
764                                             end if;
765                                             Results cat:= [L : L in
766                                             LList | Minimum(L) ge min];
767                                             end for;
768                                             end for;
769                                             end for;
770                                             end for;
771                                             end for;
772                                             end for;
773                                             return ReduceByIsometry(Results);
774                                         end function;
775                                     for n := 2 to 36 by 2
776                                         do
777                                             for l in [1,2,3,5,6,7,11,14,15,23] do
778                                                 if l eq 1 and n in [2,4,6] then continue; end
779                                                 if;
780                                                 if l eq 2 and n eq 2 then continue; end if;
781                                                 if l eq 11 and n in [20,24,28,30,32,34,36]

```

```

    then continue; end if;
773     if l eq 23 and n ge 6 then continue; end if;
774     printf "dim = %0, l = %0\n", n, l;
775     Results := ConstructLattices(l, n);
776     ModList := [L : L in Results | IsModular(L,
l)];
777     StrongModList := [L : L in Results |
IsStronglyModular(L,l)];
778     PrintFileMagma(Sprintf("SubidealLattices/%0-
Modular/%0-Dimensional", l, n), Results : Overwrite :=
true);
779     PrintFileMagma(Sprintf("SubidealLattices/%0-
Modular/%0-DimensionalModular", l, n), ModList :
Overwrite := true);
780     PrintFileMagma(Sprintf("SubidealLattices/%0-
Modular/%0-DimensionalStronglyModular", l, n),
StrongModList : Overwrite := true);
781
782     if #Results gt 0 then
783         printf "\n\n-----n = %0, l = %0: %0
lattices found! %0 of them are modular and %0 are
strongly modular-----\n\n", n, l, #Results,
#ModList, #StrongModList;
784     end if;
785     end for;
786 end for;

```