```
1
2  function DivisorsWithNorm(I, n)
3  // Input: Z_K-Ideal I; norm n in Z
4
5  // Output: List of divisors of I with norm n
6
7      norm := Integers() ! Norm(I);
8
9      if n eq 1 then return [I*I^(-1)]; end if;
10     if not IsDivisibleBy(norm, n) then return []; end
   if;
11     if norm eq n then return [I]; end if;
12
13     Fact := Factorization(I);
14
15     p1 := Fact[1][1];
16     s1 := Fact[1][2];
17     np := Integers() ! Norm(p1);
18
19     Results := [];
20
21     for j in [0..s1] do
22         if IsDivisibleBy(n, np^j) then
23             B := DivisorsWithNorm(I*p1^(-s1), Integers
   () ! (n / np^j));
24
25             for J in B do
26                 Append(~Results, p1^j*J);
27             end for;
28         end if;
29     end for;
30
31     return Results;
32
33 end function;
34
35
36 function TotallyRealGenerator(I, K, Kpos)
37 // Input: Z_K-Ideal I; Field K; Field Kpos
38
39 // Output: Boolean that indicates success; totally
   real generator of I cap Kpos
40
41     ZK := Integers(K);
42     ZKpos := Integers(Kpos);
43
44     Ipos:=ideal<ZKpos|1>;
45     Split:=[];
```

```
46
47      Fact := Factorization(I);
48
49      for i in [1..#Fact] do
50          if i in Split then continue; end if;
51
52          pi:=Fact[i][1];
53          si:=Fact[i][2];
54          piConj := IdealConjugate(pi,K);
55
56          p:=MinimalInteger(pi);
57
58          pFact:=Factorization(ideal< ZKpos | p >);
59
60          for qj in [fact[1] : fact in pFact] do
61              if ideal<ZK | Generators(qj)> subset pi
    then
62                  a := qj;
63                  break;
64              end if;
65          end for;
66
67          aZK := ideal<ZK|Generators(a)>;
68
69          if aZK eq pi^2 then
70
71              if not IsDivisibleBy(si, 2) then return
    false, _; end if;
72              Ipos *:= a^(Integers() ! (si/2));
73
74          elif aZK eq pi then
75
76              Ipos *:= a^si;
77
78          elif aZK eq pi*piConj then
79
80              if Valuation(I, pi) ne Valuation(I,
    piConj) then return false, _; end if;
81              Ipos *:= a^si;
82              for j in [1..#Fact] do
83                  pj := Fact[j][1];
84                  if pj eq piConj then
85                      Append(~Split, j);
86                      break;
87                  end if;
88              end for;
89          end if;
90      end for;
```

```
91
92      return IsPrincipal(Ipos);
93
94  end function;
95
96
97  function EmbeddingMatrix(K, Kpos)
98  // Input: Field K; Field Kpos
99
100 // Output: Matrix M whose entries give the signs of
    the embeddings of the fundamental units; List U of all
    totally positive units in ZKpos modulo norms; List of
    generators of a subgroup of Z_Kpos^* of odd index
101
102     ZKpos := Integers(Kpos);
103
104     t := #Basis(ZKpos);
105
106     G, mG := pFundamentalUnits(ZKpos,2);
107     FundUnits := [mG(G.i) : i in [1..t]];
108
109     M := ZeroMatrix(GF(2), t, t);
110
111     for i in [1..t] do
112         Embeds := RealEmbeddings(FundUnits[i]);
113         for j in [1..t] do
114             if Embeds[j] lt 0 then
115                 M[i][j] := 1;
116             end if;
117         end for;
118     end for;
119
120     U := [];
121     for a in Kernel(M) do
122         e := ZKpos ! 1;
123         for i in [1..t] do
124             e *:= FundUnits[i]^(Integers() ! a[i]);
125         end for;
126         Append(~U, e);
127     end for;
128
129     ZRel := Integers(RelativeField(Kpos, K));
130
131     Units := [];
132     for u in U do
133         for w in Units do
134             if NormEquation(ZRel,ZRel ! (u/w)) then
```

```
                continue u;
            end if;
        end for;

        Append(~Units, u);
    end for;

    return M, Units, FundUnits;

end function;


function TotallyPositiveGenerators(alpha, K, Kpos, M, U, FundUnits)
// Input: alpha in ZKpos; Field K; Field Kpos; Embedding-Matrix M; List U of all totally-positive units in ZKpos modulo norms; List FundUnits of generators of a subgroup of Z_Kpos^* of odd index

// Output: Boolean that inducates success; List of all totally-positive generators of alpha*ZK modulo norms

    t := #Basis(Kpos);
    V := ZeroMatrix(GF(2), 1, t);

    Embeds := RealEmbeddings(alpha);
    for i in [1..t] do
        if Embeds[i] lt 0 then
            V[1][i] := 1;
        end if;
    end for;

    solvable, x := IsConsistent(M,V);
    if not solvable then
        return false, _;
    end if;

    g := Integers(Kpos) ! 1;
    for i in [1..t] do
        g *:= FundUnits[i]^(Integers() ! x[1][i]);
    end for;

    return true, [alpha*g*u : u in U];

end function;
```

```
177 function ClassesModGalois(K)
178 // Input : Field K
179
180 // Output : List of representatives of the class group
    of Z_K modulo the action of the Galois-group of K/Q
181
182     ZK := Integers(K);
183     Cl, mCl := ClassGroup(ZK : Proof:="GRH");
184
185     ClModGal:=[];
186     for a in Cl do
187         A:=mCl(a);
188         for f in Automorphisms(K) do
189             if Inverse(mCl)(ideal<ZK | [f(x) : x in
    Generators(A)]>) in ClModGal then
190                 continue a;
191             end if;
192         end for;
193         Append(~ClModGal,a);
194     end for;
195
196     return [mCl(g) : g in ClModGal];
197
198 end function;
199
200
201
202 function LatFromIdeal(J, alpha, K)
203 // Input: ZK-Ideal J; Totally positive element alpha
    Kpos; Field K
204
205 // Output: Z-Lattice with elements J and inner product
    (x,y) := Tr(alpha*x*Conj(y))
206
207     n := #Basis(K);
208     z := PrimitiveElement(K);
209
210     GeneratorMatrix := KMatrixSpace(Rationals(),
    #Generators(J)*n, n) ! 0;
211
212     for i in [1..#Generators(J)] do
213         g := K ! (Generators(J)[i]);
214
215         for j in [1..n] do
216             GeneratorMatrix[(i-1)*n + j] := Vector
    (Rationals(), n, Eltseq(g*z^(j-1)));
217         end for;
```

```
218     end for;
219
220
221     BaseVecs := Basis(Lattice(GeneratorMatrix));
222
223     ZBase := [];
224     for i in [1..n] do
225         b := K ! 0;
226         for j in [1..n] do
227             b +:= BaseVecs[i][j]*z^(j-1);
228         end for;
229         Append(~ZBase, b);
230     end for;
231
232     InnProd := KMatrixSpace(Rationals(), n, n) ! 0;
233     for i in [1..n] do
234         for j in [1..n] do
235             InnProd[i][j] := Trace(K ! (alpha * z^(i-
    j)));
236         end for;
237     end for;
238
239     L := LatticeWithBasis(KMatrixSpace(Rationals(), n,
    n) ! Matrix(BaseVecs), InnProd);
240     L := LatticeWithGram(LLLGram(GramMatrix(L)));
241
242     return L;
243
244 end function;
245
246
247 function IdealLattices(d, K, Kpos, A, M, U, FundUnits,
    Reduce)
248 // Input: d in N; Field K; Field Kpos; Class Group of
    K mod Galois-Group A; Embedding-Matrix M; List of
    totally-positive units U; List FundUnits of generators
    of a subgroup of Z_Kpos^* of odd index; Boolean Reduce
    that indicates, whether the list shall be reduced by
    isometry.
249
250 // Output: List of all even ideal-lattices over K of
    square-free level and determinant d
251
252     ZK := Integers(K);
253     InvDiff := Different(ZK)^(-1);
254
255     l := 1;
256     for di in PrimeDivisors(d) do
```

```
257         l *:= di;
258     end for;
259
260     B := DivisorsWithNorm(ideal<ZK|l>, d);
261
262     Results := [];
263
264     for I in A do
265         for b in B do
266             J := (I*IdealConjugate(I,K))^
    (-1)*InvDiff*b;
267
268             x, alphaPrime := TotallyRealGenerator(J,
    K, Kpos);
269
270             if x then
271                 y, TotPos := TotallyPositiveGenerators
    (alphaPrime, K, Kpos, M, U, FundUnits);
272                 if y then
273                     for alpha in TotPos do
274                         L := LatFromIdeal(I, alpha, K);
275                         if IsEven(L) then
276                             Append(~Results, L);
277                         end if;
278                     end for;
279                 end if;
280             end if;
281         end for;
282     end for;
283
284     if Reduce then Results := ReduceByIsometry
    (Results); end if;
285
286     return Results;
287
288 end function;
289
290
291 function ModIdLat(l, n , WriteFile)
292 // Input: square-free l in N; n in N; Boolean
    WriteFile that indicates whether resulting lattices
    should be saved as files
293
294 // Output: List of all l-modular lattices of dimension
    n that are ideal lattices over some cyclotomic field
    reduced by isometry
```

```
295
296     det := l^(Integers() ! (n/2));
297
298     Lattices := [];
299
300     for m in [m : m in EulerPhiInverse(n) | m mod 4 ne
    2] do
301         K<z> := CyclotomicField(m);
302         Kpos := sub<K | z + z^(-1)>;
303
304         A := ClassesModGalois(K);
305         M, U, FundUnits := EmbeddingMatrix(K, Kpos);
306         Lattices cat:= IdealLattices(det, K, Kpos, A,
    M, U, FundUnits, false);
307     end for;
308
309     Lattices := ReduceByIsometry(Lattices);
310
311     if WriteFile then
312         PrintFileMagma(Sprintf("IdealLattices/%o-
    Modular/%o-Dimensional", l, n), Lattices :
    Overwrite := true);
313     end if;
314
315     return Lattices;
316
317 end function;
```