

```

1
2 function DivisorsWithNorm(I, n)
3 // Input:  $\mathbb{Z}_K$ -Ideal I; norm n in  $\mathbb{Z}$ 
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:  $\mathbb{Z}_K$ -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 ! &*[FundUnits[i]^(Integers() ! a
[i]) : i in [1..t]];
123         Append(~U, e);
124     end for;
125
126     ZRel := Integers(RelativeField(Kpos, K));
127
128     Units := [];
129     for u in U do
130         for w in Units do
131             if NormEquation(ZRel, ZRel ! (u/w)) then
132                 continue u;
133             end if;

```

```

134         end for;
135
136         Append(~Units, u);
137     end for;
138
139     return M, Units, FundUnits;
140
141 end function;
142
143
144 function TotallyPositiveGenerators(alpha, K, Kpos, M,
U, FundUnits)
145 // 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
146
147 // Output: Boolean that indicates success; List of all
totally-positive generators of  $\alpha \cdot ZK$  modulo norms
148
149     t := #Basis(Kpos);
150     V := ZeroMatrix(GF(2), 1, t);
151
152     Embeds := RealEmbeddings(alpha);
153     for i in [1..t] do
154         if Embeds[i] < 0 then
155             V[1][i] := 1;
156         end if;
157     end for;
158
159     solvable, x := IsConsistent(M,V);
160     if not solvable then
161         return false, _;
162     end if;
163
164     g := Integers(Kpos) ! &*[FundUnits[i]^(Integers
()) ! x[1][i]) : i in [1..t]];
165
166     return true, [alpha*g*u : u in U];
167
168 end function;
169
170
171 function ClassesModGalois(K)
172 // Input : Field K
173
174 // Output : List of representatives of the class

```

group of Z_K modulo the action of the Galois-group of K/Q

```

175
176     ZK := Integers(K);
177     Cl, mCl := ClassGroup(ZK : Proof:="GRH");
178
179     ClModGal:=[];
180     for a in Cl do
181         A:=mCl(a);
182         for f in Automorphisms(K) do
183             if Inverse(mCl)(ideal<ZK | [f(x) : x in
Generators(A)]>) in ClModGal then
184                 continue a;
185             end if;
186         end for;
187         Append(~ClModGal,a);
188     end for;
189
190     return [mCl(g) : g in ClModGal];
191
192 end function;
193
194
195
196 function LatFromIdeal(J, alpha, K)
197 // Input: ZK-Ideal J; Totally positive element alpha
Kpos; Field K
198
199 // Output: Z-Lattice with elements J and inner product
(x,y) := Tr(alpha*x*Conj(y))
200
201     n := #Basis(K);
202     z := PrimitiveElement(K);
203
204     GeneratorMatrix := KMatrixSpace(Rationals(),
#Generators(J)*n, n) ! 0;
205
206     for i in [1..#Generators(J)] do
207         g := K ! (Generators(J)[i]);
208
209         for j in [1..n] do
210             GeneratorMatrix[(i-1)*n + j] := Vector
(Rationals(), n, Eltseq(g*z^(j-1)));
211         end for;
212     end for;
213
214
215     BaseVecs := Basis(Lattice(GeneratorMatrix));

```

```

216
217     ZBase := [];
218     for i in [1..n] do
219         b := K ! 0;
220         for j in [1..n] do
221             b += BaseVecs[i][j]*z^(j-1);
222         end for;
223         Append(~ZBase, b);
224     end for;
225
226     InnProd := KMatrixSpace(Rationals(), n, n) ! 0;
227     for i in [1..n] do
228         for j in [1..n] do
229             InnProd[i][j] := Trace(K ! (alpha * z^(i-
230 j)));
231         end for;
232     end for;
233
234     L := LatticeWithBasis(KMatrixSpace(Rationals(), n,
235 n) ! Matrix(BaseVecs), InnProd);
236     L := LatticeWithGram(LLGram(GramMatrix(L)));
237
238     return L;
239
240 end function;
241
242 function IdealLattices(d, K, Kpos, A, M, U, FundUnits,
243 Reduce)
244 // Input: d in N; Field K; Field Kpos; Class Group of
245 K mod Galois-Group A; Embedding-Matrix M; List of
246 totally-positive units U; List FundUnits of generators
247 of a subgroup of  $\mathbb{Z}_{Kpos}^*$  of odd index; Boolean Reduce
248 that indicates, whether the list shall be reduced by
249 isometry.
250
251 // Output: List of all even ideal-lattices over K of
252 square-free level and determinant d
253
254     ZK := Integers(K);
255     InvDiff := Different(ZK)^(-1);
256
257     l := &*(PrimeDivisors(d))
258
259     B := DivisorsWithNorm(ideal<ZK|l>, d);
260
261     Results := [];
262
263

```

```

255     for I in A do
256         for b in B do
257             J := (I*IdealConjugate(I,K))^
(-1)*InvDiff*b;
258
259             x, alphaPrime := TotallyRealGenerator(J,
K, Kpos);
260
261             if x then
262                 y, TotPos := TotallyPositiveGenerators
(alphaPrime, K, Kpos, M, U, FundUnits);
263                 if y then
264                     for alpha in TotPos do
265                         L := LatFromIdeal(I, alpha, K);
266                         if IsEven(L) then
267                             Append(~Results, L);
268                         end if;
269                     end for;
270                 end if;
271             end if;
272         end for;
273     end for;
274
275     if Reduce then Results := ReduceByIsometry
(Results); end if;
276
277     return Results;
278
279 end function;
280
281
282 function ModIdLat(l, n , PrintFile)
283 // Input: square-free l in N; n in N; Boolean
PrintFile that indicates whether resulting lattices
should be saved as files
284
285 // Output: List of all l-modular lattices of dimension
n that are ideal lattices over some cyclotomic field
reduced by isometry
286
287     det := l^(Integers() ! (n/2));
288
289     Lattices := [];
290
291     for m in [m : m in EulerPhiInverse(n) | m mod 4 ne
2] do

```

```

292         K<z> := CyclotomicField(m);
293         Kpos := sub<K | z + z^(-1)>;
294
295         A := ClassesModGalois(K);
296         M, U, FundUnits := EmbeddingMatrix(K, Kpos);
297         Lattices cat:= IdealLattices(det, K, Kpos, A,
M, U, FundUnits, false);
298     end for;
299
300     Lattices := ReduceByIsometry(Lattices);
301
302     if PrintFile then
303         PrintFileMagma(Sprintf("IdealLattices/%o-
Modular/%o-Dimensional", l, n), Lattices :
Overwrite := true);
304     end if;
305
306     return Lattices;
307
308 end function;

```