Normaliser complements for the parabolic subgroups of finite unitary reflection groups

Don Taylor

Version: 14 December 2017
TEXed: 26 December 2017

This file provides MAGMA code to construct representatives of the conjugacy class of parabolic subgroups of a finite unitary reflection group and to find a complement to the parabolic subgroup in its normaliser.

1 Using the code

The primary purpose of the code is to supplement the results of [2] and to verify the assertions in Theorems 5.11 and 5.13 about the parabolic subgroups of G_{29} and G_{31}. These assertions can be checked using quite a small amount of code but a secondary purpose, which is the reason for the bulk of the code, is to provide a package to deal with all parabolic subgroups and to reproduce the tables in section 6 of [2].

To load the code into MAGMA, ensure that `Complements.m` is in your current directory and after starting MAGMA, type

```
> load "Complements.m";
```

The main 'top level' functions are paraData and *reflectionPart* for primitive groups and standardParabolic, standardComplement and *specialRoots* for the imprimitive groups.

In addition, *parabolicNames* is a sequence of sequences of the symbolic names of the conjugacy classes of parabolic subgroups in the primitive reflection groups.

Example 1.1. The names of the (conjugacy classes) of parabolic subgroups of G_{29} and G_{31} can be obtained by typing

```
> parabolicNames[29];
> parabolicNames[31];
```

Example 1.2. A parabolic subgroup (and related structures) of type `B2` in G_{29} can be obtained by the command

```
> P, Q, H, N, roots, ρ, J, Ξ, W := paraData(29, "B2");
```

where the objects returned are:

- P, the parabolic subgroup of type `B2`,
- Q, the pointwise stabiliser of the orthogonal complement of the space of fixed points of P,

- H, a complement to P in its normaliser,
- *roots*, the indexed set of roots of G_{29},
- ρ, the canonical map from roots to coroots,
- J, indices of the roots of the generators of P,
- Ξ, pseudo-positive roots defining H,
- W, the reflection group G_{29}.

Example 1.3. To check that H is indeed a complement to P in its normaliser N and that P is generated by the reflections with roots Ξ, use

```
> P meet H eq sub<P|>;
> N eq sub<N | P, H >;
> P eq sub<P| [REFLECTION(a, ρ(a)) : a in Ξ] >;
```

All three expressions should return true.

Example 1.4. To check that H is the *setwise* stabiliser of Ξ in W use the function stabiliser defined in §3 below. (MAGMA does not have a built-in intrinsic to compute the stabiliser of a *set* of vectors acted on by a matrix group.)

```
> H eq stabiliser(W, SET(Ξ));
```

Example 1.5. For each parabolic subgroup P the code in `Complements.m` provides an explicit complement H to P in its normaliser and a set of roots Ξ whose reflections, in most cases, generate P. Here is code to display the names of the parabolic subgroups of the primitive reflection groups for which H is *not* the setwise stabiliser of Ξ in W.

```
> for n := 23 to 37 do
>   for X in parabolicNames[n] do
>     P, Q, H, N, roots, ρ, J, Ξ, W := paraData(n, X);
>     if H ne stabiliser(W, SET(Ξ)) then n, X; end if;
>   end for;
> end for;
```

Example 1.6. To display the data for all parabolic subgroups of the primitive reflection groups use

```
> for n := 4 to 37 do display(n : headings); end for;
```

2 The primitive unitary reflection groups

The code in this section returns the parabolic subgroups, related groups and root data for the Shephard–Todd groups G_4, G_5, ..., G_{37}. Refer to [3] for an exposition of unitary reflection groups in general.

2.1 Conjugacy classes of parabolic subgroups

The enumeration of the parabolic subgroups and their standard names can be found in [4]. The labels for the conjugacy classes of irreducible parabolic subgroups uses the notation introduced by Cohen [1]. As in [4] a parabolic subgroup which is the direct product of irreducible reflection groups of types $T_1, T_2, \ldots, T_k$ is labelled $T_1+T_2+\cdots+T_k$.

For the imprimitive reflection groups which occur as parabolic subgroups we use the Coxeter notation B_n for the group $G(2,1,n)$, D_n for $G(2,2,n)$ ($n \geq 4$) and $I_2(m)$ for $G(m,m,2)$ but otherwise use $G(m,p,n)$. For the Coxeter group of type G_2 we use $I_2(6)$ to avoid confusion with the Shephard and Todd notation G_k. The cyclic reflection groups of orders 2 and 3 are denoted by `A1` and `L1`. For $n \geq 4$, the cyclic reflection group of order n is denoted by `Z`n. If there are two conjugacy classes of parabolic subgroups of type T the classes are labelled T and T'.

---

parabolicNames$[n]$ is the list of names of the (conjugacy classes) of parabolic subgroups of the primitive reflection group G_n.

```
parabolicNames := [];
parabolicNames[4] := [ "L1" ];
parabolicNames[5] := [ "L1", "L1'" ];
parabolicNames[6] := [ "A1", "L1" ];
parabolicNames[7] := [ "A1", "L1", "L1'" ];
parabolicNames[8] := [ "Z4" ];
parabolicNames[9] := [ "A1", "Z4" ];
parabolicNames[10] := [ "Z4", "L1" ];
parabolicNames[11] := [ "Z4", "L1", "A1" ];
parabolicNames[12] := [ "A1" ];
parabolicNames[13] := [ "A1", "A1'" ];
parabolicNames[14] := [ "A1", "L1" ];
parabolicNames[15] := [ "A1", "A1'", "L1" ];
parabolicNames[16] := [ "Z5" ];
parabolicNames[17] := [ "A1", "Z5" ];
parabolicNames[18] := [ "L1", "Z5" ];
parabolicNames[19] := [ "Z5", "A1", "L1" ];
parabolicNames[20] := [ "L1" ];
parabolicNames[21] := [ "A1", "L1" ];
parabolicNames[22] := [ "A1" ];
parabolicNames[23] := [ "A1", "2A1", "A2", "I2(5)" ];
parabolicNames[24] := [ "A1", "A2", "B2" ];
parabolicNames[25] := [ "L1", "2L1", "L2" ];
parabolicNames[26] := [ "A1", "L1", "A1+L1", "L2", "G(3,1,2)" ];
parabolicNames[27] := [ "A1", "A2", "A2'", "I2(5)", "B2" ];
parabolicNames[28] := [ "A1", "A1'", "2A1", "A2", "A2'", "B2",
                        "A1+A2", "(A1+A2)'", "B3", "B3'" ];
parabolicNames[29] := [ "A1", "2A1", "A2", "B2", "A1+A2", "A3", "A3'",
                        "G(4,4,3)", "B3" ];
parabolicNames[30] := [ "A1", "2A1", "A2", "I2(5)", "A1+A2", "A1+I2(5)",
                        "A3", "H3" ];
```

```
parabolicNames[31] := [ "A1", "2A1", "A2", "G(4,2,2)", "A1+A2", "A3", "G(4,2,3)" ];
parabolicNames[32] := [ "L1", "2L1", "L2", "L1+L2", "L3" ];
parabolicNames[33] := [ "A1", "2A1", "A2", "A1+A2", "A3", "3A1", "G(3,3,3)",
                        "A1+A3", "A4", "G(3,3,4)", "D4" ];
parabolicNames[34] := [ "A1", "2A1", "A2", "A1+A2", "A3", "3A1", "G(3,3,3)",
                        "A1+A3", "G(3,3,4)", "A4", "2A1+A2", "2A2",
                        "A1+G(3,3,3)", "D4", "A1+A4", "A2+A3",
                        "A1+G(3,3,4)", "A5", "A5'", "D5", "G(3,3,5)", "K5" ];
parabolicNames[35] := [ "A1", "2A1", "A2", "A1+A2", "A3", "3A1", "A1+A3", "A4",
                        "2A2", "2A1+A2", "D4", "A1+A4", "D5", "A1+2A2", "A5" ];
parabolicNames[36] := [ "A1", "A2", "2A1", "A3", "A1+A2", "3A1", "3A1'", "A4",
                        "D4", "A1+A3", "(A1+A3)'", "2A2", "2A1+A2", "4A1",
                        "A5", "A5'", "D5", "A1+A4", "A1+D4", "A2+A3",
                        "2A1+A3", "A1+2A2", "3A1+A2", "A1+A2+A3", "A1+A5",
                        "A2+A4", "A1+D5", "A6", "E6", "D6"];
parabolicNames[37] := [ "A1", "2A1", "A2", "A1+A2", "3A1", "A3", "A1+A3", "2A2",
                        "A4", "2A1+A2", "D4", "4A1", "A1+A4", "A2+A3", "A5",
                        "D5", "A1+D4", "2A1+A3", "A1+2A2", "3A1+A2",
                        "A1+A5", "A2+A4", "A6", "2A1+A4", "E6", "A1+D5",
                        "2A3", "D6", "A2+D4", "A1+A2+A3", "2A1+2A2",
                        "A1+A6", "A1+A2+A4", "A1+E6", "A2+D5", "A3+A4",
                        "A7", "D7", "E7"];
```

2.2 Setting up

The function rootDatum defined below returns the root system and associated data for the Shephard and Todd group number n. Note that in this function NUMFLD is set to true and so it is important that COMPLEXROOTDATUM only be called once in any session (otherwise the base ring of the groups and spaces returned from each call will be isomorphic but not identical, causing problems with coercion).

A horrible hack to deal with the aforementioned problem.

```
AddAttribute(GrpPerm, "rootDatum");
rtDtmFormat := recformat< roots : SetIndx, coroots : SetIndx, ρ : Map,
    W : GrpMat >;
```

A name unlikely to appear elsewhere!

```
Sxyz_3 := Sym(1);
Sxyz_3`rootDatum := [];
```

The map ρ returned by this function sends a root to the corresponding coroot and so the sequence *coroots* is never explicitly needed in the code which follows.

```
rootDatum := function(n)
  S := Sym(1);
  uggr := S`rootDatum;
  if IsDefined(uggr, n) then
```

```
        rtDtm := uggr[n];
        roots := rtDtm`roots;
        coroots := rtDtm`coroots;
        ρ := rtDtm`ρ;
        W := rtDtm`W;
    else
        roots, coroots, ρ, W, J := ComplexRootDatum(n : NumFld);
```

The roots are constructed within a vector space with the standard inner product and therefore the universe needs to be changed so that the correct inner product is used.

```
        F := BaseRing(W);
        if F eq Rationals() then
            V := VectorSpace(F, Nrows(J), J);
        else
            σ := map< F → F | x ↦ ComplexConjugate(x) >;
            V := UnitarySpace(J, σ);
        end if;
        ChangeUniverse(~roots, V);
        ChangeUniverse(~coroots, V);
        ρ := map< roots → coroots | a ↦ ρ(a) >;
        S`rootDatum[n] := rec< rtDtmFormat |
            roots := roots, coroots := coroots, ρ := ρ, W := W >;
    end if;
    return roots, coroots, ρ, W;
end function;
```

3 Miscellaneous utilities

A convenient abbreviation.

```
str := IntegerToString;
```

A sequence of reflections is *irredundant* if no reflection in the sequence is a proper power of another reflection in the sequence.

Given an irredundant sequence R of reflections, return a sequence of indecomposable components. There is no check to ensure that the sequence actually is irredundant.

```
components := function(R)
    seq := [];
    X := R;
    while not IsEmpty(X) do
        r := X[1];
        Exclude(~X, r);
        C := [r];
        ndx := 0;
        while ndx lt #C do
```

```
        ndx +:= 1;
        a := C[ndx];
        extn := [ x : x in X | a*x ne x*a ];
        C cat:= extn;
        X := [ x : x in X | x notin extn ];
      end while;
      APPEND(~seq, C);
    end while;
    return seq;
  end function;
```

Given a sequence *refsubs* of cyclic subgroups generated by reflections, return generators of the maximal elements in the set.

```
  maximalReps := function(refsubs)
    SORT(~refsubs, func< A, B | #B - #A >);
    if exists{ A : A in refsubs | not ISPRIME(#A) } then
      R := [];
      for A in refsubs do
        if forall{ B : B in R | not A subset B } then
          APPEND(~R, A);
        end if;
      end for;
    else R := refsubs;
    end if;
    return [A.1 : A in R];
  end function;
```

W is a matrix group and X is a *set* of vectors in the space on which W acts. Return W_X, the setwise stabiliser of X in W.

```
  stabiliser := function(W, X)
    f, G, _ := ORBITACTION(W, X);
    Ψ := { f(x) : x in X };
    H := STABILISER(G, Ψ);
    return H @@ f;
  end function;
```

Given a group G acting on a set T, find the orbits of G. The action could either be a matrix action or action by conjugation on a union of conjugacy classes of G.

```
  orbits := function(G, T)
    orbs := [];
    while #T gt 0 do
      S := REP(T)^G;
      APPEND(~orbs, S);
      T := { x : x in T | x notin S };
    end while;
```

```
    return orbs;
end function;
```

4 Identification and standard names

We keep track of the reflection subgroups via their names.

---

groupName is an associative array which identifies irreducible unitary reflection groups by associating the triple consisting of the group order, the number of rank 1 parabolic subgroups and the maximal order of a reflection to its name.

```
groupName := AssociativeArray(car<Integers(), Integers(), Integers()>);
groupName[<2,1,2>] := "A1";    // SymmetricGroup(2)
groupName[<3,1,3>] := "L1";    // CyclicGroup(3)
groupName[<4,1,4>] := "Z4";    // CyclicGroup(4)
groupName[<5,1,5>] := "Z5";    // CyclicGroup(5)
groupName[<6,1,6>] := "Z6";    // CyclicGroup(6)
groupName[<6,3,2>] := "A2";    // SymmetricGroup(3)
groupName[<8,1,8>] := "Z8";    // CyclicGroup(8)
groupName[<8,4,2>] := "B2";    // ShephardTodd(2,1,2)
groupName[<10,1,10>] := "Z10";    // CyclicGroup(10)
groupName[<10,5,2>] := "I2(5)";    // ShephardTodd(5,5,2)
groupName[<12,1,12>] := "Z12";    // CyclicGroup(12)
groupName[<12,6,2>] := "I2(6)";    // ShephardTodd(6,6,2)
groupName[<16,6,2>] := "G(4,2,2)";
groupName[<16,8,2>] := "I2(8)";    // ShephardTodd(8,8,2)
groupName[<18,5,3>] := "G(3,1,2)";
groupName[<20,1,20>] := "Z20";    // CyclicGroup(20)
groupName[<20,10,2>] := "I2(10)";    // ShephardTodd(10,10,2)
groupName[<24,4,3>] := "L2";    // ShephardTodd(4)
groupName[<24,1,24>] := "Z24";    // CyclicGroup(24)
groupName[<24,6,2>] := "A3";    // SymmetricGroup(4)
groupName[<24,8,2>] := "G(6,3,2)";
groupName[<30,1,30>] := "Z30";    // CyclicGroup(30)
groupName[<32,6,4>] := "G(4,1,2)";
groupName[<32,10,2>] := "G(8,4,2)";
groupName[<36,8,3>] := "G(6,2,2)";
groupName[<48,9,2>] := "B3";    // ShephardTodd(2,1,3)
groupName[<48,10,3>] := "G6";    // ShephardTodd(6)
groupName[<48,12,2>] := "G12";    // ShephardTodd(12)
groupName[<50,7,5>] := "G(5,1,2)";
groupName[<54,9,2>] := "G(3,3,3)";
groupName[<60,1,60>] := "Z60";    // CyclicGroup(60)
groupName[<64,10,4>] := "G(8,2,2)";
groupName[<72,8,3>] := "G5";    // ShephardTodd(5)
groupName[<72,8,6>] := "G(6,1,2)";
```

groupName[<96,6,4>] := "G8"; // SHEPHARDTODD(8)
groupName[<96,12,2>] := "G(4,4,3)";
groupName[<96,18,2>] := "G13"; // SHEPHARDTODD(13)
groupName[<100,12,2>] := "G(10,2,2)";
groupName[<120,10,2>] := "A4"; // SYMMETRICGROUP(5)
groupName[<120,15,2>] := "H3"; // SHEPHARDTODD(23)
groupName[<144,14,3>] := "G7"; // SHEPHARDTODD(7)
groupName[<144,20,3>] := "G14"; // SHEPHARDTODD(14)
groupName[<162,12,3>] := "G(3,1,3)";
groupName[<192,12,2>] := "D4"; // SHEPHARDTODD(2,2,4)
groupName[<192,15,2>] := "G(4,2,3)";
groupName[<192,18,4>] := "G9"; // SHEPHARDTODD(9)
groupName[<240,30,2>] := "G22"; // SHEPHARDTODD(22)
groupName[<288,14,4>] := "G10"; // SHEPHARDTODD(10)
groupName[<288,26,3>] := "G15"; // SHEPHARDTODD(15)
groupName[<336,21,2>] := "J3(4)"; // SHEPHARDTODD(24)
groupName[<360,20,3>] := "G20"; // SHEPHARDTODD(20)
groupName[<384,15,4>] := "G(4,1,3)";
groupName[<384,16,2>] := "B4"; // SHEPHARDTODD(2,1,4)
groupName[<432,21,2>] := "G(6,3,3)";
groupName[<576,26,4>] := "G11"; // SHEPHARDTODD(11)
groupName[<600,12,5>] := "G16"; // SHEPHARDTODD(16)
groupName[<648,12,3>] := "L3"; // SHEPHARDTODD(25)
groupName[<648,18,2>] := "G(3,3,4)";
groupName[<720,15,2>] := "A5"; // SYMMETRICGROUP(6)
groupName[<720,50,3>] := "G21"; // SHEPHARDTODD(21)
groupName[<1152,24,2>] := "F4"; // SHEPHARDTODD(28)
groupName[<1200,42,5>] := "G17"; // SHEPHARDTODD(17)
groupName[<1296,21,3>] := "M3"; // SHEPHARDTODD(26)
groupName[<1536,24,2>] := "G(4,4,4)";
groupName[<1800,32,5>] := "G18"; // SHEPHARDTODD(18)
groupName[<1920,20,2>] := "D5"; // SHEPHARDTODD(2,2,5)
groupName[<1944,22,3>] := "G(3,1,4)";
groupName[<2160,45,2>] := "J3(5)"; // SHEPHARDTODD(27)
groupName[<3072,28,2>] := "G(4,2,4)";
groupName[<3600,62,5>] := "G19"; // SHEPHARDTODD(19)
groupName[<3840,25,2>] := "B5"; // SHEPHARDTODD(2,1,5)
groupName[<5040,21,2>] := "A6"; // SYMMETRICGROUP(7)
groupName[<7680,40,2>] := "N4"; // SHEPHARDTODD(29)
groupName[<9720,30,2>] := "G(3,3,5)";
groupName[<14400,60,2>] := "H4"; // SHEPHARDTODD(30)
groupName[<23040,30,2>] := "D6"; // SHEPHARDTODD(2,2,6)
groupName[<40320,28,2>] := "A7"; // SYMMETRICGROUP(8)
groupName[<46080,36,2>] := "B6"; // SHEPHARDTODD(2,1,6)
groupName[<46080,60,2>] := "O4"; // SHEPHARDTODD(31)
groupName[<51840,36,2>] := "E6"; // SHEPHARDTODD(35)
groupName[<51840,45,2>] := "K5"; // SHEPHARDTODD(33)

```
groupName[<155520,40,3>] := "L4";    // ShephardTodd(32)
groupName[<174960,45,2>] := "G(3,3,6)";
groupName[<322560,42,2>] := "D7";    // ShephardTodd(2,2,7)
groupName[<362880,36,2>] := "A8";    // SymmetricGroup(9)
groupName[<2903040,63,2>] := "E7";   // ShephardTodd(36)
groupName[<5160960,56,2>] := "D8";   // ShephardTodd(2,2,8)
groupName[<39191040,126,2>] := "K6"; // ShephardTodd(34)
groupName[<696729600,120,2>] := "E8"; // ShephardTodd(37)
```

If the signature $\langle n, r, m \rangle$ is not in the list of names, return the string value of $\langle n, r, m \rangle$.

```
name := func< n, r, m |
  IsDefined(groupName, <n,r,m>) select groupName[<n,r,m>]
    else "<"*str(n)*"|"*str(r)*"|"*str(m)*">" >;
```

Given a sequence of names of the irreducible components of a reflection group, return the standard form of the name.

```
prettyName := function(names)
  if IsEmpty(names) then return "*X*"; end if;
  count := func< n | (n gt 1) select IntegerToString(n) else "" >;
  Sort(~names);
  sep := "";
  name := "";
  prev_tag := names[1];
  tag_num := 0;
  for tag in names do
    if tag eq prev_tag then tag_num +:= 1;
    else
      name *:= sep * count(tag_num) * prev_tag;
      prev_tag := tag;
      tag_num := 1;
      sep := "+";
    end if;
  end for;
  name *:= sep * count(tag_num) * prev_tag;
  return name;
end function;
```

Return the standard name of the reflection group G generated by the sequence R of reflections generating the rank 1 parabolic subgroups.

```
standardName1 := function(G, R)
  if IsEmpty(R) then return "A0"; end if;
  assert G eq sub< G | R >;
  sform := [];
  for C in components(R) do
    Append(~sform, name(Order(sub<G|C>),#C,Max({Order(r) : r in C})));
```

```
    end for;
    return prettyName(sform);
end function;
```

Return the standard name of the reflection subgroup of a group H, where *roots* is a root system of an overgroup and ρ is the function from roots to coroots.

```
standardName2 := function(H, roots, ρ)
    if #H eq 1 then return "A0"; end if;
    R_ := { sub< H | r > : a in roots | r in H where r is REFLECTION(a, ρ(a)) };
    R := maximalReps(SETSEQ(R_));
    return standardName1(H, R);
end function;
```

4.1 The main code for primitive groups

4.1.1 The groups of rank 2

```
rankTwo := function(n, X)
    roots, _, ρ, W := rootDatum(n);
    V := UNIVERSE(roots);
```

Default values, to be overwritten.

```
    H := sub<W|>;
    Ξ := [];
```

The elements of Δ are the roots of the generators of W.

```
    Δ := roots[1..NGENS(W)];
    case n:
      when 4: F<ω> := BASERING(W);
        J := case< X | "L1": 1, default: 0>;
        Ξ := [Δ[1], -Δ[1]];
      when 5: F<ω> := BASERING(W);
        J := case< X | "L1": 1, "L1′": 2, default: 0>;
        Ξ := case< X |
          "L1": [ g*Δ[1] : g in [1, -1]],
          "L1′": [ g*Δ[2] : g in [1, -1]],
           default: []>;
      when 6: F<i, ω> := BASERING(W);
        J := case< X | "A1": 1, "L1": 2, default: 0>;
        if X eq "L1" then
          Ξ := [ g*Δ[2] : g in [1, -1, i, -i]];
        elif X eq "A1" then
          H := sub< W| [-i, 0, -2*i*ω - i + 1, i]>;
        end if;
      when 7: F<i, ω> := BASERING(W);
        J := case< X | "A1": 1, "L1": 2, "L1′": 3, default: 0>;
```

```
  Ξ := case< X |
    "L1": [ g*Δ[2] : g in [1, −1, i, −i]],
    "L1′": [ g*Δ[3] : g in [1, −1, i, −i]],
     default: []> ;
  if X eq "A1" then
    H := sub< W|[−i*ω + 1, i*ω, −2*i*ω + 2, i*ω − 1]> ;
  end if;
when 8: F<i> := BaseRing(W);
  J := case< X | "Z4": 1, default: 0> ;
  Ξ := case< X | "Z4": [V| Δ[1]], default: []> ;
when 9: F<i, a> := BaseRing(W);
  J := case< X | "A1": 1, "Z4": 2, default: 0> ;
  H := case< X |
    "A1": sub< W| [(i+1)*a/2, 0, 0, (i+1)*a/2] >,
    "Z4": sub< W| [i*a/2 + a/2, 1, 0, −i*a/2 + a/2] >,
     default: sub<W|>> ;
when 10: F<i, ω> := BaseRing(W);
  J := case< X | "Z4": 1, "L1": 2, default: 0> ;
  Ξ := case< X |
    "Z4": [ g*Δ[1] : g in [1, ω, ω^2]],
    "L1": [ g*Δ[2] : g in [1, −1, i, −i]],
     default: []> ;
when 11: F<i, ω, a> := BaseRing(W);
  J := case< X | "Z4": 1, "A1": 2, "L1": 3, default: 0> ;
  Ξ := case< X |
    "L1": [ g^j*Δ[3] : j in [0..7] | true where g is (1−i)/a ],
     default: []> ;
  H := case< X |
    "Z4": sub< W| [(i−1)*ω*a/2, 0, i*ω*(a+1), −(i+1)*ω*a/2]>,
    "A1": sub< W| [(i+1)*ω*a/2, 0, 0, (i+1)*ω*a/2]>,
     default: sub<W|>> ;
when 12: F<b> := BaseRing(W);
  J := case< X | "A1": 1, default: 0> ;
  Ξ := case< X | "A1": [V| Δ[1]], default: []> ;
when 13: F<i, a> := BaseRing(W);
  J := case< X | "A1": 1, "A1′": 2, default: 0> ;
  H := case< X |
    "A1": sub< W| [ (i−1)*a/2, 0, i*(a+1), −(i+1)*a/2 ] >,
    "A1′": sub< W| [ (i+1)*a/2+1, −1, (i+1)*(a+1)+1, −(i+1)*a/2−1 ]>,
     default: sub<W|>> ;
when 14: F<b, ω> := BaseRing(W);
  J := case< X | "L1": 1, "A1": 2, default: 0> ;
  Ξ := case< X |
    "L1": [Δ[1], −Δ[1]],
    "A1": [ g*Δ[2] : g in [1, ω, ω^2]],
     default: []> ;
when 15: F<i, ω, a> := BaseRing(W);
```

J := **case**< X | "A1": 1, "L1": 2, "A1′": 3, **default**: 0>;
Ξ := **case**< X |
"L1": [$g*\Delta[2] : g$ ***in*** $[1,-1,i,-i]$],
default: []>;
H := **case**< X |
"A1": ***sub***< W | [$(i+1)*\omega*a/2, 0,\ i*\omega*(a+1), -(i-1)*\omega*a/2$]>,
"A1′": ***sub***< W | [$-i*\omega*(a+1), i*\omega,\ -2*i*\omega*(a+1), i*\omega*(a+1)$] >,
default: ***sub***<W|>>;
when 16: F<z_5> := BASERING(W);
J := **case**< X | "Z5": 1, **default**: 0>;
Ξ := **case**< X |
"Z5": [$g*\Delta[1] : g$ ***in*** $[1,-1]$],
default: []>;
when 17: F<i, z_5> := BASERING(W);
J := **case**< X | "A1": 1, "Z5": 2, **default**: 0>;
Ξ := **case**< X |
"Z5": [$i^j*\Delta[2] : j$ ***in*** $[0..3]$],
default: []>;
if X ***eq*** "A1" **then**
H := ***sub***< W | $[i*{z_5}^3, 0,\ -i*z_5 + i + {z_5}^3, -i*{z_5}^3]$ >;
end if;
when 18: F<ω, z_5> := BASERING(W);
J := **case**< X | "L1": 1, "Z5": 2, **default**: 0>;
Ξ := **case**< X |
"L1": [$(-z_5)^j*\Delta[1] : j$ ***in*** $[0..9]$],
"Z5": [$(-\omega)^j*\Delta[2] : j$ ***in*** $[0..5]$],
default: []>;
when 19: F<i, ω, z_5> := BASERING(W);
J := **case**< X | "Z5": 1, "L1": 2, "A1": 3, **default**: 0>;
Ξ := **case**< X |
"Z5": [$(-i*\omega)^j*\Delta[1] : j$ ***in*** $[0..11]$],
"L1": [$(i*z_5)^j*\Delta[2] : j$ ***in*** $[0..19]$],
default: []>;
if X ***eq*** "A1" **then**
H := ***sub***< W | $[\omega^2-z_5, z_5,\ \omega*({z_5}^2+z_5-1)-z_5-2, -\omega^2+z_5]$ >;
end if;
when 20: F<ω, τ> := BASERING(W);
J := **case**< X | "L1": 1, **default**: 0>;
Ξ := **case**< X |
"L1": [$\Delta[1],\ -\Delta[1]$],
default: []>;
when 21: F<i, ω, τ> := BASERING(W);
J := **case**< X | "A1" :1, "L1": 2, **default**: 0>;
Ξ := **case**< X |
"L1": [$i^j*\Delta[2] : j$ ***in*** $[0..3]$],
default: []>;
if X ***eq*** "A1" **then**

```
            H := sub< W | [-i*ω, 0, -i*ω^2+i-ω*τ, i*ω] >;
        end if;
      when 22: F<i,τ> := BaseRing(W);
        J := case< X | "A1": 1, default: 0>;
        if X eq "A1" then
          H := sub< W | [i, 0, i*τ-i+1, -i] >;
        end if;
      else
        error "G" * str(n), "is not rank 2";
    end case;
    error if J eq 0, "Invalid parabolic name";
    P := sub< W | W.J >;
    N := Normaliser(W, P);
    Q := Stabiliser(N, Δ[J]);
    if Ξ eq [] then Ξ := [Δ[J]]; end if;
    if #H eq 1 then H := stabiliser(W, Set(Ξ)); end if;
    return P, Q, H, N, roots, ρ, J, Ξ, W;
  end function;
```

4.1.2 The primitive groups of rank at least 3

A reflection group is said to be of Eisenstein type if the group of roots of unity in its ring of definition has order 6. In this case the ring of definition contains the Eisenstein integers $\mathbb{Z}[\omega]$, where ω is a cube root of unity. The groups in question are G_{25}, G_{26}, G_{27}, G_{32}, G_{33} and G_{34}.

Given the index n and a type X of a parabolic subgroup of G_n, the following functions return P, Q, H, N, *roots*, ρ, J, Ξ and W described in §1.

```
  eisensteinType := function(n, X)
    roots, _, ρ, W := rootDatum(n);
    V := Universe(roots);
    Δ := Basis(V);
    if n eq 27 then
      F<ω,τ> := BaseRing(W);
    else
      F<ω> := BaseRing(W);
    end if;
    thicken := func< J | J cat [ ω*v : v in J] cat [ω^2*v : v in J ] >;
    Ξ := [];
    case n:
      when 25:
        gens := [W.1, W.2, W.3];
        J := case< X |
          "L1": [1],
          "2L1": [1,3],
          "L2": [1,2],
```

default : []> ;
Ξ := **case**< X |
"L1": [V| [1,0,0], [−1,0,0]],
"2L1" : [V| [1,0,0], [−1,0,0], [0,0,1], [0,0,−1]],
default : [] > ;
when 26:
gens := [W.1, W.2, W.3] ;
J := **case**< X |
"L1": [1],
"A1": [3],
"A1+L1": [1,3],
"L2": [1,2],
"G(3,1,2)": [2,3],
default : []> ;
Ξ := **case**< X |
"L1": [V| [1,0,0], [−1,0,0]],
"A1+L1": [V| [1,0,0], [−1,0,0], [0,0,1],[0,0,ω],[0,0,ω^2]],
"G(3,1,2)": [V| [0,1,0], [0,−1,0], [0,0,1], [0,0,−1]],
default : [] > ;
when 27:
Δ ***cat***:= [V| [1,−1,0]] ;
gens := [W.1, W.2, W.3, W.1∗W.2∗W.1] ;
J := **case**< X |
"A1": [1],
"A2": [1,2],
"A2′": [3,4],
"I2(5)": [1,3],
"B2": [2,3],
default : []> ;
Ξ := **case**< X |
"A2" : [V|
[1, 0, 0], [ω, 0, 0], [ω^2, 0, 0],
[0, −1, 0], [0, −ω, 0], [0, −ω^2, 0]],
"A2′" : [V|
[0, 0, 1], [0, 0, ω], [0, 0, ω^2],
[−1, 1, 0], [−ω, ω, 0], [−ω^2, ω^2, 0]],
default : [] > ;
when 32:
gens := [W.1, W.2, W.3, W.4] ;
J := **case**< X |
"L1": [1],
"L2": [1,2],
"2L1": [1,3],
"L3":[1,2,3],
"L1+L2":[1,2,4],
default : []> ;
Ξ := **case**< X |

```
    "L1": [V| [1,0,0,0], [−1,0,0,0]],
    "2L1" : [V| [1,0,0,0], [−1,0,0,0], [0,0,1,0], [0,0,−1,0]],
    "L3": [V|
      [1, 0, 0, 0], [−1, 0, 0, 0],
      [0, ω², 0, 0], [0,−ω², 0, 0],
      [0, 0, ω, 0], [0, 0,−ω, 0]],
    "L1+L2": [V|
      [1, 0, 0, 0], [ω, 0, 0, 0], [ω², 0, 0, 0],
      [0,−1, 0, 0], [0,−ω, 0, 0], [0,−ω², 0, 0],
      [0, 0, 0, 1], [0, 0, 0,−1]],
     default : [] >;
when 33:
  Δ cat:= [V ![−1,−1,−1,0,0]];
  ss := W.1∗W.2∗W.1;
  gens := [ W.i : i in [1..5]];
  APPEND(∼gens, ss∗W.3∗ss);
  J := case< X |
    "A1": [1],
    "2A1": [1,3],
    "A2": [1,2],
    "A1+A2": [1,2,5],
    "A3": [1,2,3],
    "3A1": [1,3,5],
    "G(3,3,3)": [2,3,4],
    "A1+A3": [1,2,3,5],
    "A4": [1,2,4,5],
    "G(3,3,4)": [1,2,3,4],
    "D4": [2,4,5,6],
     default : [] >;
  Ξ := case< X |
    "G(3,3,3)": [V|
      [0, 1, 0, 0, 0],
      [0, 0, 1, 0, 0],
      [0, 0, 0, ω², 0],
      [0, 1, 1, −ω², 0],
      [0, 1, −ω², ω, 0],
      [0, −ω, 1, 1, 0],
      [0, 0, −ω, −ω, 0],
      [0, −ω², 0, −1, 0]],
    "G(3,3,4)" : [V|
      [1, 0, 0, 0, 0],
      [0, 1, 0, 0, 0],
      [0, 0, 1, 0, 0],
      [0, 0,−1, 0, 0],
      [0, 0, 0, 1, 0],
      [0, 0, 0,−1, 0],
      [−1, −1, 0, 0, 0]],
```

```
        default : [] >;
when 34:
    Δ cat:= [V ! [−1, −1, −1, 0, 0, 0]];
    ss := W.1 ∗ W.2 ∗ W.1;
    gens := [ W.i : i in [1..6]];
    APPEND(∼gens, ss ∗ W.3 ∗ ss);
    J := case< X |
      "A1": [1],
      "2A1": [1, 3],
      "A2": [1, 2],
      "A1+A2": [1, 2, 5],
      "A3": [1, 2, 3],
      "3A1": [1, 3, 5],
      "G(3,3,3)": [2, 3, 4],     // *
      "A1+A3": [1, 2, 3, 5],
      "G(3,3,4)": [1, 2, 3, 4],
      "A4": [1, 2, 4, 5],
      "2A1+A2": [1, 3, 5, 6],
      "2A2": [1, 2, 5, 6],
      "A1+G(3,3,3)": [2, 3, 4, 6],     // *
      "D4": [2, 4, 5, 7],
      "A1+A4": [1, 3, 4, 5, 6],
      "K5": [1, 2, 3, 4, 5],
      "A5": [1, 2, 4, 5, 6],
      "A5′": [1, 4, 5, 6, 7],
      "A1+G(3,3,4)": [1, 2, 3, 4, 6],     // *
      "D5": [2, 4, 5, 6, 7],
      "A2+A3": [1, 2, 3, 5, 6],
      "G(3,3,5)": [2, 3, 4, 5, 6],     // *
       default : [] >;
    Ξ := case< X |
      "G(3,3,3)": [V |
        [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, ω^2, 0, 0],
        [0, 1, 1, −ω^2, 0, 0], [0, 1, −ω^2, ω, 0, 0],
        [0, −ω, 1, 1, 0, 0], [0, 0, −ω, −ω, 0, 0], [0, −ω^2, 0, −1, 0, 0]],
      "G(3,3,4)" : [V |
        [0, 1, 0, 0, 0, 0], [−1, −1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, −1, 0, 0],
        [0, 0, ω^2, 0, 0, 0], [0, 0, −ω^2, 0, 0, 0], [0, 0, ω, ω, 0, 0],
        [0, 0, −ω, −ω, 0, 0]],
      "G(3,3,5)": [V |
        [0, ω^2, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, −ω, −ω, 0, 0, 0],
        [0, −ω, 0, −ω^2, 0, 0], [0, 1, 1, −ω^2, 0, 0], [0, 0, −ω^2, −ω^2, 0, 0],
        [0, 0, 0, −ω, −ω, −ω]],
      "A1+G(3,3,3)": [V |
        [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, ω^2, 0, 0],
        [0, 1, 1, −ω^2, 0, 0], [0, 1, −ω^2, ω, 0, 0],
        [0, −ω, 1, 1, 0, 0], [0, 0, −ω, −ω, 0, 0], [0, −ω^2, 0, −1, 0, 0],
```

```
            [0,0,0,0,0,1], [0,0,0,0,0,ω], [0,0,0,0,0,ω^2]],
          "A1+G(3,3,4)": [V|
            [0,1,0,0,0,0], [−1,−1,0,0,0,0], [0,0,0,1,0,0], [0,0,0,−1,0,0],
            [0,0,ω^2,0,0,0], [0,0,−ω^2,0,0,0], [0,0,ω,ω,0,0],
            [0,0,−ω,−ω,0,0],
            [0,0,0,0,0,1], [0,0,0,0,0,ω], [0,0,0,0,0,ω^2]],
           default : [] >;
      else
        error "G" * str(n), "is not Eisenstein";
    end case;
    error if IsEmpty(J), "Invalid parabolic name";
    P := sub< W | [gens[i] : i in J ]>;
    if IsEmpty(Ξ) then Ξ := thicken([Δ[j] : j in J]); end if;
    N := Normaliser(W, P);
    Q := Stabiliser(N, [Δ[j] : j in J]);
    H := stabiliser(W, Set(Ξ));
    return P, Q, H, N, roots, ρ, J, Ξ, W;
  end function;
```

A reflection group is said to be of Euclidean type if the group of roots of unity in its ring of definition is $\{\pm 1\}$. In this case the ring of definition is real. The groups in question are G_{23}, G_{24}, G_{28}, G_{30}, G_{35}, G_{36} and G_{37}.

```
  euclidType := function(n, X)
      roots, _, ρ, W := rootDatum(n);
      V := Universe(roots);
      Δ := Basis(V);
      Ξ := [];
      case n:
        when 23:
          J := case< X |
            "A1" : [1],
            "2A1": [1,3],
            "A2": [2,3],
            "I2(5)": [1,2],
              default : [] >;
        when 24:
          J := case< X |
            "A1": [1],
            "A2": [1,2],
            "B2": [1,3],
              default : []>;
          Ξ := case< X |
            "A2" : [V| [1,0,0], [0,−1,0]],
              default : []>;
        when 28:
          J := case< X |
```

```
      “A1”: [1],
      “A1′”: [3],
      “A2”: [1,2],
      “2A1”: [1,3],
      “A2′”: [3,4],
      “B2”: [2,3],
      “B3”: [2,3,4],
      “B3′”: [1,2,3],
      “A1+A2”: [1,3,4],
      “(A1+A2)′”: [1,2,4],
       default : [] >;
when 30:
  J := case< X |
      “A1”: [1],
      “2A1”: [1,3],
      “I2(5)”: [1,2],
      “A2”: [3,4],
      “A1+I2(5)”: [1,2,4],
      “A3”: [2,3,4],
      “H3”: [1,2,3],
      “A1+A2”: [1,3,4],
       default : [] >;
when 35:
  J := case< X |
      “A1”: [1],
      “2A1”: [1,2],
      “A2”: [1,3],
      “A1+A2”: [1,2,3],
      “A3”: [1,3,4],
      “3A1”: [1,2,6],
      “A1+A3”: [1,3,4,6],
      “A4”: [1,2,3,4],
      “2A2”: [1,3,5,6],
      “2A1+A2”: [1,2,5,6],
      “D4”: [2,3,4,5],
      “A5”: [1,3,4,5,6],
      “A1+A4”: [1,2,3,4,6],
      “D5”: [1,2,3,4,5],
      “A1+2A2”: [1,2,3,5,6],
       default : [] >;
when 36:
  J := case< X |
      “A1”: [1],
      “2A1”: [1,2],
      “A2”: [1,3],
      “A1+A2”: [1,2,3],
      “A3”: [1,3,4],
```

"3A1": [1, 2, 6],
"3A1′": [2, 5, 7],
"A1+A3": [1, 3, 4, 6],
"(A1+A3)′": [2, 5, 6, 7],
"A4": [1, 2, 3, 4],
"2A2": [1, 3, 5, 6],
"2A1+A2": [1, 2, 5, 6],
"D4": [2, 3, 4, 5],
"4A1": [1, 2, 5, 7],
"A5": [1, 3, 4, 5, 6],
"A5′": [2, 4, 5, 6, 7],
"A1+A4": [1, 2, 3, 4, 6],
"D5": [1, 2, 3, 4, 5],
"2A1+A2": [1, 2, 3, 5, 6],
"A2+A3": [1, 3, 5, 6, 7],
"2A1+A3": [1, 2, 5, 6, 7],
"A1+2A2": [1, 2, 3, 6, 7],
"A1+D4": [2, 3, 4, 5, 7],
"3A1+A2": [1, 2, 3, 5, 7],
"A1+D5": [1, 2, 3, 4, 5, 7],
"A1+A5": [1, 2, 4, 5, 6, 7],
"A2+A4": [1, 2, 3, 4, 6, 7],
"A1+A2+A3": [1, 2, 3, 5, 6, 7],
"A6": [1, 3, 4, 5, 6, 7],
"D6": [2, 3, 4, 5, 6, 7],
"E6": [1, 2, 3, 4, 5, 6],
default : [] > ;

when 37:

J := **case**< X |
"A1": [1],
"2A1": [1, 2],
"A2": [1, 3],
"A1+A2": [1, 2, 3],
"3A1": [1, 2, 6],
"A3": [1, 3, 4],
"A1+A3": [1, 3, 4, 6],
"2A2": [1, 3, 5, 6],
"A4": [1, 2, 3, 4],
"2A1+A2": [1, 2, 5, 6],
"D4": [2, 3, 4, 5],
"4A1": [1, 2, 5, 7],
"A1+A4": [1, 2, 3, 4, 6],
"A2+A3": [1, 3, 5, 6, 7],
"A5": [1, 3, 4, 5, 6],
"D5": [1, 2, 3, 4, 5],
"A1+D4": [2, 3, 4, 5, 7],
"2A1+A3": [1, 2, 5, 6, 7],

```
            "A1+2A2": [1,2,3,5,6],
            "3A1+A2": [1,2,3,5,7],
            "A1+A5": [1,2,4,5,6,7],
            "A2+A4": [1,2,3,4,6,7],
            "A6": [1,3,4,5,6,7],
            "2A1+A4": [1,2,5,6,7,8],
            "E6": [1,2,3,4,5,6],
            "A1+D5": [1,2,3,4,5,7],
            "2A3": [1,3,4,6,7,8],
            "D6": [2,3,4,5,6,7],
            "A2+D4": [2,3,4,5,7,8],
            "A1+A2+A3": [1,2,3,5,6,7],
            "2A1+2A2": [1,2,3,5,7,8],
            "A1+A6": [1,2,4,5,6,7,8],
            "A1+A2+A4": [1,2,3,5,6,7,8],
            "A1+E6": [1,2,3,4,5,6,8],
            "A2+D5": [1,2,3,4,5,7,8],
            "A3+A4": [1,2,3,4,6,7,8],
            "A7": [1,3,4,5,6,7,8],
            "D7": [2,3,4,5,6,7,8],
            "E7": [1,2,3,4,5,6,7],
             default : [] >;
        else
            error "G" * str(n), "is not Euclidean";
    end case;
    error if IsEmpty(J), "Invalid parabolic name", X;
    P := sub< W | [W.i : i in J ]>;
    if IsEmpty(Ξ) then Ξ := [Δ[j] : j in J]; end if;
    N := Normaliser(W, P);
    Q := Stabiliser(N, Ξ);
    H := stabiliser(W, Set(Ξ));
    return P, Q, H, N, roots, ρ, J, Ξ, W;
end function;
```

A reflection group is said to be of Gaussian type if the group of roots of unity in its ring of definition is $\{\pm 1, \pm i\}$. In this case the ring of definition is the Gaussian integers $\mathbb{Z}[i]$. The groups in question are G_{29} and G_{31}.

```
gaussType := function(n, X)
    roots, _, ρ, W := rootDatum(n);
    V := Universe(roots);
    Δ := Basis(V);
    F<i> := BaseRing(W);
    Ξ := [];
    case n:
      when 29:
        gens := [W.1, W.2, W.3, W.4, W.1*W.2*W.1];
```

```
  Δ cat:= [V ! [1,1,0,0]];
  J := case< X |
    "A1" : [1],     // Q not in H, so H is not a stabiliser
    "A2" : [1,2],
    "B2" : [2,3],
    "2A1" : [1,4],     // Q not in H, so H is not a stabiliser
    "A1+A2" : [1,2,4],     // Q = 1 but H is not a stabiliser
    "A3" : [1,3,4],
    "A3′" : [3,4,5],
    "B3" : [2,3,4],     // Q not in H, so H is not a stabiliser
    "G(4,4,3)" : [1,2,3],
    default : []>;
  Ξ := case< X |
    "A2" : [V|
      [ 1, 0, 0, 0], [−1, 0, 0, 0], [0, i, 0, 0], [0, −i, 0, 0]],
    "B2" : [V|
      [0, i, i − 1, 0], [0, 0, 1, 0], [0, 1, 0, 0], [0, −i − 1, −i, 0]],
    "A3" : [V|
      [ i, 0, 0, 0], [−i, 0, 0, 0], [ 0, 0, 1, 0], [ 0, 0, −1, 0],
      [ 0, 0, i, 0], [ 0, 0, −i, 0], [ 0, 0, 0, i], [ 0, 0, 0, −i]],
    "A3′" : [V|
      [ 0, 0, 0, i], [ 0, 0, 0, −i], [ 0, 0, 1, 0], [ 0, 0, −1, 0],
      [ 0, 0, i, 0], [ 0, 0, −i, 0], [ 1, 1, 0, 0], [−1, −1, 0, 0]],
   "G(4,4,3)" : [V|
      [ 0, 0, i, 0], [ 0, 0, −i, 0], [ i, 0, 1, 0], [−i, 0, −1, 0],
      [ 0, i, i − 1, 0], [ 0, −i, −i+1, 0], [ i, 1, i+1, 0], [−i, −1, −i−1, 0]],
    default : []>;
  H := case< X |
    "A1": sub< W |
      [1,0,0,0, i,2,i+2,i+1, −i,−1,−i−1,−i−1, −1, i,2∗i−1,i],
      [1,0,0,0, −i,−i−1,−2∗i−1,−i, 0,0,1,0, i,i+2,2∗i+1,i+1],
      [−1,0,0,0, 0,i,i−1,0, 0,−i−1,−i,0, i+1,2,2,1],
      [i,0,0,0, 0,i,0,0, 0,0,i,0, 0,0,0,i]>,
    "2A1" : sub< W |
      [0,0,0,i, i,1,i+2,1, 0,0,−1,0, −i,0,0,0],
      [0,0,0,−1, −i,−i−2,−2∗i−1,−i, i+1,2,i+2,i+1, −1,0,0, 0],
      [−1,0,0,0, −1,2∗i−1,2∗i−2,i−1, 1,−2∗i,−2∗i+1,−i, 0,0,0,1]>,
    "A1+A2" : sub< W | [i,0,0,0, 0,i,0,0, 0,0,i,0, 0,0,0,i]>,
    "B3" : sub< W | [i,0,0,0, 0,i,0,0, 0,0,i,0, 0,0,0,i]>,
     default : sub<W|> >;
when 31:
  gens := [W.1, W.2, W.3, W.4, W.5];
  Δ cat:= [V ! [1,0,−i,0]];
  J := case< X |
    "A1" : [1],     // Q not in H
    "2A1" : [1,4],     // Q not in H
    "A2" : [1,2],
```

```
            "G(4,2,2)" : [1,3,5],    // Q not in H
            "A3" : [2,3,4],
            "A1+A2" : [1,2,4],    // H is not a stabiliser
            "G(4,2,3)" : [1,2,3,5],    // Q not in H
            default : []>;
        Ξ := case< X |
            "A2" : [V|
              [1, 0, 0, 0], [−1, 0, 0, 0], [0, i, 0, 0], [0, −i, 0, 0]],
            "A3" : [V|
              [0, 1, 0, 0], [0,−1, 0, 0],
              [0, 0, 1, 0], [0, 0,−1, 0], [0, 0, i, 0], [0, 0,−i, 0],
              [0, 0, 0, 1], [0, 0, 0,−1]],
            default : []>;
        H := case< X |
            "A1" : sub< W |
              [1,0,0,0, 0,1,0,0, 0,0,1,1, 0,0,0,−1],
              [1,0,0,0, i−1,−1,−i+1,0, 0,0,1,0, −1,−i−1,1,1],
              [i,0,0,0, −2∗i−1,−i,2∗i,i, 1,0,−1,0, 0,0,0,−1]>,
            "2A1" : sub< W |
              [0,0,0,i, −2,−i−1,i+2,1, −1,−i,i+1,i+1, −i,0,0,0],
              [1,0,0,0, −i+1,1,−2,−1, 1,0,−i,−i, 0,0,0,i]>,
            "G(4,2,2)" : sub< W |
              [−1,0,0,0, 0,−i,1,1, i,0,−i,0, −1,0,i+1,1],
              [−i,0,−1,0, i+1,0,−i,−i, 0,0,−1,0, i−1,−1,−i+2,−i+1]>,
            "A1+A2" : sub< W | [i,0,0,0, 0,i,0,0, 0,0,i,0, 0,0,0,i]>,
            "G(4,2,3)" : sub< W | [i,0,0,0, 0,i,0,0, 0,0,i,0, 0,0,0,i]>,
            default : sub<W|> >;
    else
        error "G" ∗ str(n), "is not Gaussian";
    end case;
    error if IsEmpty(J), "Invalid parabolic name";
    P := sub< W | [gens[i] : i in J ]>;
    if IsEmpty(Ξ) then
        Ξ := [V| Δ[j] : j in J];
    end if;
    N := Normaliser(W, P);
    Q := Stabiliser(N, Ξ);
    if #H eq 1 then H := stabiliser(W, Set(Ξ)); end if;
    return P, Q, H, N, roots, ρ, J, Ξ, W;
end function;
```

The dispatching function for the primitive groups.

```
paraData := function(n, X)
    if n in [4..22] then
        return rankTwo(n, X);
    elif n in [29,31] then
```

```
    return gaussType(n, X);
  elif n in [23, 24, 28, 30, 35, 36, 37] then
    return euclidType(n, X);
  elif n in [25, 26, 27, 32, 33, 34] then
    return eisensteinType(n, X);
  else error "Shephard-Todd index", n, "is out of range";
  end if;
end function;
```

4.2 Restriction to the space of fixed points

Given a parabolic subgroup P the code in the previous section returns a complement H to P in its normaliser. The following functions can be used to obtain the subgroup H° of H generated by the elements that act as reflections on the space of fixed points of P.

Given a matrix M, return the restriction of M to an invariant subspace, where S is either the basis matrix of the space or the sequence of basis vectors.

```
restriction := func< M, S | SOLUTION(T, T*M) where T is MATRIX(S) >;
```

Apply the restriction function to a group G.

```
restrictGroup := function(G, S)
  T := MATRIX(S);
  n := NROWS(T);
  F := BASERING(T);
  return sub<GL(n, F) | [ restriction(G.i, S) : i in [1..NGENS(G)]] >;
end function;
```

Given groups P and K acting on V the function returns the reflection subgroup K° of the restriction of K to the space E of fixed points of P, and generators of the rank 1 parabolic subgroups of K°.

```
reflectionPart := function(P, K, V)
  E := sub< V | BASIS(&meet[ EIGENSPACE(r, 1) : r in GENERATORS(P)]) >;
  L := restrictGroup(K, BASIS(E));
  R_ := SETSEQ({ sub<L|r> : r in L | ISREFLECTION(r) });
  R := maximalReps(R_);
  return sub< L | R >, E, R;
end function;
```

5 Display and test procedures

Display the data for the parabolic subgroups of the primitive reflection group G_n in the format of the tables in [2].

```
display := procedure(n : headings := false)
  if headings then
    printf "\n%3o %8o %10o %10o %12o %10o %6o\n",
      "G"*IntegerToString(n), "P", "Q", "|H0|", "H0", "H=H0", "a";
    print "                ", &*["-" : i in [1..56]];
  end if;
  for X in parabolicNames[n] do
    P, Q, H, N, roots, ρ, J, Ξ, W := paraData(n, X);
    V := Universe(roots);
    H0, E, R := reflectionPart(P, N, V);
    a := Index(H, Centraliser(H, P));
    printf "%12o %10o %10o %12o %10o %6o\n",
      X, standardName2(Q, roots, ρ), #H0, standardName1(H0, R), #H eq #H0, a;
  end for;
end procedure;
```

In Magma version 2.23 and earlier the first NGens(W) roots returned by ComplexRootDatum are the roots of the generators $W.i$ of W. The code for the groups of rank two relies on this and therefore we provide a test to ensure that it holds in the current version.

```
checkRoots := procedure()
  for n := 4 to 37 do
    roots, coroots, ρ, W, J := ComplexRootDatum(n : NumFld);
    V := Universe(roots);
    for i := 1 to NGens(W) do
      a := roots[i];
      b := a*W.i;
      assert sub<V|a> eq sub<V|b> and a ne b;
    end for;
  end for;
  "Test passed!";
end procedure;
```

Check that the parabolic subgroup returned by paraData agrees with the standard name.

```
nameCheck := procedure()
  for n := 4 to 37 do
    passed := true;
    for X in parabolicNames[n] do
      P, Q, H, N, roots, ρ := paraData(n, X);
      N := standardName2(P, roots, ρ);
```

```
        if X ne N then
          print "Possible name error:", X, N, "in group", n;
          passed := false;
        end if;
      end for;
      if passed then "Group", n, "passed"; end if;
    end for;
  end procedure;
```

Test that the subgroup H returned by paraData is in fact a complement to the parabolic subgroup P in its normaliser N.

```
  checkComplement := procedure(n, X)
    P, Q, H, N, roots, ρ, J, Ξ, W := paraData(n, X);
    "P meet H = 1     :", P meet H eq sub<P|>;
    "N eq <P,H>       :", N eq sub<N | P, H >;
    "generation of P  :", P eq sub<P| [REFLECTION(a, ρ(a)) : a in Ξ] >;
  end procedure;
```

6 The imprimitive unitary reflection groups

Given integers m, p, n, n_0 and a partition $\lambda = (n_1, n_2, \ldots, n_d)$ of $n - n_0$, the standard parabolic subgroup $P_{(n_0,\lambda)}$ of the reflection group $G(m,p,n)$ is

$$\prod_{i=1}^{d} G(1,1,n_i) \times G(m,p,n_0).$$

In contrast to the description in §3 of [2] we place the factor $G(m,p,n_0)$ at the right and we ensure that the n_i (for $i \geq 1$) are in increasing order.

The standard parabolic subgroup $P_{(n_0,\lambda)}$ of $G(m,p,n)$.

```
  standardParabolic := function(m, p, n0, λ)
    n := n0 + &+λ;
    W := SHEPHARDTODD(m, p, n);
    SORT(~λ);
```

If $\lambda = (n_1, n_2, \ldots, n_d)$, then $K = [0, n_1, n_1 + n_2, \ldots, n_1 + \cdots + n_d]$.

```
    K := [ i eq 1 select 0 else SELF(i-1) + λ[i-1] : i in [1..#λ+1] ];
    gens := [ W.i : i in &cat [[K[j-1]+1 .. K[j]-1] : j in [2..#K]] ];
    if n0 eq 1 then
      APPEND(~gens, W.NGENS(W) );
    elif n0 gt 1 then
      gens cat:= [ W.i : i in [K[#K] + 1 .. NGENS(W)] ];
    end if;
    return sub<W | gens>;
  end function;
```

Convert a partition lambda to index form; i.e., a sequence of terms $\langle i, m \rangle$, where m is the number of times i occurs in λ.

```
indexForm := func<λ |
  [ < i, m > : i in [1..&+λ] | m ne 0 where m is #[x : x in λ | x eq i]] >;
```

Generators of the subgroup of a complement of the standard parabolic subgroup P in its normaliser that induce reflections on $\mathrm{Fix}(P)$. If $p = 1$ or $n_0 \neq 0$, the kth member of the sequence returned generates a group isomorphic to $G(m, 1, b_k)$, otherwise it generates a group isomorphic to $G(m, p_k, b_k)$ where $p_k = p/\gcd(p, k)$ and where $\langle k, b_k \rangle$ is the kth term of the index form of λ.

```
stdRefCompGens := function(m, p, n0, λ)
  n := n0 + &+λ;
  ndx := indexForm(λ);
  W := ShephardTodd(m, p, n);
  if m le 2 then F := Rationals(); θ := -1;
  else F<θ> := CyclotomicField(m); end if;
  t := IdentityMatrix(F, n);
  if n0 ne 0 then
    t[n, n] := θ^-1;
  end if;
  gseq := [];
  ptr := 0;
  for pair in ndx do
    k, b_k := Explode(pair);
    p_k := (p ne 1 and n0 eq 0) select p div GCD(p, k) else 1;
    G := ChangeRing(ShephardTodd(m, p_k, b_k), F);
    id1 := IdentityMatrix(F, ptr);
    id_k := IdentityMatrix(F, k);
    ptr +:= k*b_k;
    id2 := IdentityMatrix(F, n-ptr);
    if n0 ne 0 then
      gens := [W | DiagonalJoin(<id1, TensorProduct(G.i, id_k), id2>) :
        i in [1..Ngens(G)-1]];
      g := DiagonalJoin(<id1, TensorProduct(G.b_k, id_k), id2>) * t^k;
      Append(~gens, W ! g);
    else
      gens := [W | DiagonalJoin(<id1, TensorProduct(G.i, id_k), id2>) :
        i in [1..Ngens(G)]];
    end if;
    Append(~gseq, gens);
  end for;
  return gseq;
end function;
```

The Shephard–Todd group $G(m, p, n)$ is the semidirect product of a group $A(m, p, n)$

by the symmetric group $\mathrm{Sym}(n)$, where $A(m,p,n)$ is a group of index p in the direct product of n copies of the group of mth roots of unity.

The following function returns the group $A(m,p,n)$.

```
baseGroup := function(m, p, n)
  if m le 2 then F := Rationals(); θ := −1;
  else F<θ> := CyclotomicField(m); end if;
  gens := [];
  if p eq 1 then
    for j := 1 to n do
      M := IdentityMatrix(F, n);
      M[j, j] := θ;
      Append(~gens, M);
    end for;
  else
    for j := 1 to n−1 do
      M := IdentityMatrix(F, n);
      M[j, j] := θ;
      M[j+1, j+1] := θ^−1;
      Append(~gens, M);
    end for;
    if p ne m then
      M := IdentityMatrix(F, n);
      M[1, 1] := θ^p;
      Append(~gens, M);
    end if;
  end if;
  return sub<GL(n, F) | gens>;
end function;
```

This function applies only to parabolic subgroups of type $(0,\lambda)$. It is the intersetion of the normaliser with the base group in $G(m,1,n)$.

```
diagPart := function(m, λ)
  n := &+λ;
  ndx := indexForm(λ);
  if m le 2 then F := Rationals(); θ := −1;
  else F<θ> := CyclotomicField(m); end if;
  gseq := [];
  ptr := 0;
  for pair in ndx do
    k, b_k := Explode(pair);
    A := ChangeRing(baseGroup(m, 1, b_k), F);
    I_1 := IdentityMatrix(F, ptr);
    I_K := IdentityMatrix(F, k);
    ptr +:= k*b_k;
    I_2 := IdentityMatrix(F, n−ptr);
```

```
      gens := [DiagonalJoin(<I_1, TensorProduct(A.i, I_k), I_2>) : i in [1..Ngens(A)]];
      Append(~gseq, gens);
    end for;
    return gseq;
  end function;
```

This is a complement of the standard parabolic in its normaliser.

```
  standardComplement := function(m, p, n_0, λ)
    gseq := stdRefCompGens(m, p, n_0, λ);
    n := n_0 + &+λ;
    H := sub< ShephardTodd(m, p, n) | &cat gseq >;
    if n_0 ne 0 or p eq 1 then return H; end if;
    ndxfrm := indexForm(λ);
    e := GCD(p, GCD(λ));
    e *:= &*[ p div GCD(p, k[1]) : k in ndxfrm ];
    e := e div p;
    if e eq 1 then return H; end if;
```

At this point we know that n_0 is 0, p **ne** 1 and that the index form of lambda has at least two terms.

```
    if m eq 2 then
      F := Rationals();
      z := -1;
    else
      F<z> := CyclotomicField(m);
    end if;
    ptr := 0;
    gseq := [];
    for pair in ndxfrm do
      k, b_k := Explode(pair);
      if b_k eq 1 then
        ptr +:= 1;
      else
        G := ChangeRing(ShephardTodd(1, 1, b_k), F);
        id_1 := IdentityMatrix(F, ptr);
        id_k := IdentityMatrix(F, k);
        ptr +:= k*b_k;
        id_2 := IdentityMatrix(F, n-ptr);
        Append(~gseq, [DiagonalJoin(<id_1, TensorProduct(G.i, id_k), id_2>) :
          i in [1..Ngens(G)]]);
      end if;
    end for;
    S := sub<GL(n, F) | &cat gseq>;
    dseq := diagPart(m, λ);
    D := sub<GL(n, F) | &cat dseq > meet baseGroup(m, p, n);
    return sub< GL(n, F) | S, D >;
  end function;
```

In Theorem 4.5 of [2] we proved that if the order of the group of roots of unity of the ring of definition of $G(m,p,n)$ is twice an odd integer and if m/p and p/e are coprime, where e is $\gcd(n_1,n_2,\ldots,n_d)$, then there is a set J of roots of P whose setwise stabiliser is a complement to P in its normaliser.

```
specialRoots := function(m, p, n0, λ)
  n := n0 + &+λ;
  if m le 2 then F := RationalField(); ζ := −1;
  else F<ζ> := CyclotomicField(m); end if;
  γ := IsOdd(m) select ζ else ζ^2;
  h := IsOdd(m) select m else m div 2;
  assert IsOdd(h);
  V := VectorSpace(F, n);
  K := [ i eq 1 select 0 else Self(i−1) + λ[i−1] : i in [1..#λ+1] ];
  rts := &join[ {@ V.(i−1)−V.i : i in [K[j−1]+2..K[j]] @} : j in [2..#K]];
  if n0 eq 0 then δ0 := {@ @};
  else
    e := GCD(p, GCD(λ));
    j := p div e;
    k := m div j;
    θ := ζ^k;
    d := K[#K]+1;
    δ0 := {@ V.(n−1) − θ^i*V.n : i in [0..j−1] @} join
                {@ ζ*V.(n−1) − θ^i*V.n : i in [0..j−1] @} join
                {@ V.(i−1)−V.i : i in [d+1..n−1] @};
    if p ne m then
      δ0 join:= {@ θ^i*V.n : i in [0..j−1] @}
              join {@ V.i : i in [d..n−1] @};
    end if;
  end if;
  return δ0 join &join[ {@ γ^i*v : v in rts @} : i in [0..h−1]];
end function;
```

References

[1] A. M. Cohen. Finite complex reflection groups. *Ann. Sci. École Norm. Sup. (4)*, 9:379–436, 1976.

[2] M. Khrishnasamy and D. E. Taylor. Normalisers of parabolic subgroups in unitary reflection groups. Technical report, University of Sydney, July 2017.

[3] G. I. Lehrer and D. E. Taylor. *Unitary reflection groups*, volume 20 of *Australian Mathematical Society Lecture Series*. Cambridge University Press, Cambridge, 2009.

[4] D. E. Taylor. Reflection subgroups of finite complex reflection groups. *J. Algebra*, 366:218–234, 2012.