Reflection subgroups of complex reflection groups

Don Taylor

Version: 30 January 2025

Definition 1. If H is a reflection subgroup of G, a *simple extension* of H is a subgroup K such that $K = \langle H, r \rangle$ for some reflection $r \notin H$. The simple extension K is a *minimal extension* of H if for all reflection subgroups L such that $H \subsetneq L \subseteq K$ we have $L = K$.

The purpose of the MAGMA code in this file is to find the simple extensions of all the reflection subgroups (up to conjugacy) of an irreducible complex reflection group. The conjugacy classes of subgroups will be identified using the naming convention of Shephard and Todd and the abbreviations `Bn = G(2,1,n)`, `Bn(3) = G(3,1,n)`, `Bn(2p) = G(2p,p,n)`, `Dn = G(2,2,n)` and `Dn(p) = G(p,p,n)`.

To use the code, first use the function `setup` to obtain a record — in this case *ctx* — whose field *ctx`group* is a complex reflection group. (The function `setup` accepts either an integer or a sequence such as [4,2,3].)

Next use the function `rankOne` to find the reflection subgroups of rank 1, up to conjugacy. Finally, use `simpleExtensions` to recursively find all conjugacy classes of reflection subgroups and all of their simple extensions. For example:

```
ctx := setup(25); // the Shephard--Todd group G25.
names, refgroup := rankOne(ctx);
extension := AssociativeArray(Parent(""));
simpleExtensions(~names,~extension,~refgroup,ctx);
```

In this example, *names* is the sequence of string names of the conjugacy classes of reflection subgroups of the Shephard and Todd group G_{25} and *refgroup* is an associative array associating a name with a representative subgroup.

The associative array *extension* associates the name of a conjugacy class with the sequence of names of its simple extensions. For example:

```
> names;
[ Z3, Z3Z3, G4, G25, Z3Z3Z3 ]
> extension["L1L1"];
[ G25, Z3Z3Z3 ]
```

For convenience, there is a function `printTable` which carries out these actions and prints a table of simple extensions. In the table, `P` indicates a parabolic subgroup. If SORTED is specified, the subgroups will be sorted in order of rank and group order and the table will display the rank of each subgroup, the number of reflection generators and the number of conjugates.

```
> printTable(26 : Sorted);

P 1 1 9  | A1      | [ A1Z3, B2(3), A2 ]
P 1 1 12 | Z3      | [ G4, Z3Z3, B2(3), A1Z3 ]
P 2 2 36 | A1Z3    | [ A1G4, G26, B3(3), B2(3)Z3, A2Z3 ]
  2 2 12 | A2      | [ B3(3), A2Z3, D3(3), B2(3) ]
  2 2 12 | Z3Z3    | [ G25, B2(3)Z3, B2(3), Z3Z3Z3 ]
P 2 2 12 | B2(3)   | [ G26, B3(3), B2(3)Z3 ]
P 2 2 9  | G4      | [ G25, G26, A1G4 ]
  3 3 12 | A2Z3    | [ G26, B3(3), B2(3)Z3 ]
  3 3 4  | Z3Z3Z3  | [ G25, B2(3)Z3 ]
  3 3 9  | A1G4    | [ G26 ]
  3 3 1  | D3(3)   | [ B3(3) ]
  3 3 12 | B2(3)Z3 | [ G26, B3(3) ]
  3 3 4  | B3(3)   | [ G26 ]
  3 3 1  | G25     | [ G26 ]
P 3 3 1  | G26     | []
```

1 Identification

Tools to identify irreducible unitary reflection groups. The index is $<m,r,z>$ where m is the order, r is the number of reflections of order 2 and z is the number of classes of reflections.

```
toStr := IntegerToString;
Z := Integers();
groupName := AssociativeArray(CartesianProduct(<Z, Z, Z>));
groupName[<2,1,1>] := "A1";    // ShephardTodd(2,1,1)
groupName[<3,0,2>] := "Z3";    // ShephardTodd(3,1,1)
groupName[<4,1,3>] := "Z4";    // ShephardTodd(4,1,1)
groupName[<5,0,4>] := "Z5";    // ShephardTodd(5,1,1)
groupName[<6,3,1>] := "A2";    // ShephardTodd(1,1,3)
groupName[<8,4,2>] := "B2";    // ShephardTodd(2,1,2)
groupName[<10,5,1>] := "D2(5)";    // ShephardTodd(5,5,2)
groupName[<12,6,2>] := "D2(6)";    // ShephardTodd(6,6,2)
groupName[<16,6,3>] := "B2(4)";    // ShephardTodd(4,2,2)
groupName[<16,8,2>] := "D2(8)";    // ShephardTodd(8,8,2)
groupName[<18,3,3>] := "B2(3)";    // ShephardTodd(3,1,2)
groupName[<20,10,2>] := "D2(10)";    // ShephardTodd(10,10,2)
groupName[<24,0,2>] := "G4";    // ShephardTodd(4) = SL(2,3)
groupName[<24,6,1>] := "A3";    // ShephardTodd(1,1,4)
groupName[<24,8,2>] := "B2(6)";    // ShephardTodd(6,3,2)
groupName[<32,6,4>] := "G(4,1,2)";    // ShephardTodd(4,1,2)
groupName[<32,10,3>] := "B2(8)";    // ShephardTodd(8,4,2)
groupName[<36,6,4>] := "G(6,2,2)";    // ShephardTodd(6,2,2)
groupName[<48,9,2>] := "B3";    // ShephardTodd(2,1,3)
groupName[<48,6,3>] := "G6";    // ShephardTodd(6)
```

```
groupName[<48,12,1>] := "G12";    // ShephardTodd(12)
groupName[<50,5,5>] := "G(5,1,2)";    // ShephardTodd(5,1,2)
groupName[<54,9,1>] := "D3(3)";    // ShephardTodd(3,3,3)
groupName[<64,10,5>] := "G(8,2,2)";    // ShephardTodd(8,2,2)
groupName[<72,0,4>] := "G5";    // ShephardTodd(5)
groupName[<96,6,3>] := "G8";    // ShephardTodd(8)
groupName[<96,12,1>] := "D3(4)";    // ShephardTodd(4,4,3)
groupName[<96,18,2>] := "G13";    // ShephardTodd(13)
groupName[<100,10,6>] := "G(10,2,2)";    // ShephardTodd(10,2,2)
groupName[<120,10,1>] := "A4";    // ShephardTodd(1,1,5)
groupName[<120,15,1>] := "G23";    // ShephardTodd(23)
groupName[<144,6,5>] := "G7";    // ShephardTodd(7)
groupName[<144,12,3>] := "G14";    // ShephardTodd(14)
groupName[<150,15,1>] := "D3(5)";    // ShephardTodd(5,5,3)
groupName[<162,9,3>] := "B3(3)";    // ShephardTodd(3,1,3)
groupName[<192,12,1>] := "D4";    // ShephardTodd(2,2,4)
groupName[<192,15,2>] := "B3(4)";    // ShephardTodd(4,2,3)
groupName[<192,18,4>] := "G9";    // ShephardTodd(9)
groupName[<240,30,1>] := "G22";    // ShephardTodd(22)
groupName[<288,6,5>] := "G10";    // ShephardTodd(10)
groupName[<288,18,4>] := "G15";    // ShephardTodd(15)
groupName[<336,21,1>] := "G24";    // ShephardTodd(24)
groupName[<360,0,2>] := "G20";    // ShephardTodd(20)
groupName[<384,16,2>] := "B4";    // ShephardTodd(2,1,4)
groupName[<576,18,6>] := "G11";    // ShephardTodd(11)
groupName[<600,0,4>] := "G16";    // ShephardTodd(16)
groupName[<648,0,2>] := "G25";    // ShephardTodd(25)
groupName[<648,18,1>] := "D4(3)";    // ShephardTodd(3,3,4)
groupName[<720,15,1>] := "A5";    // ShephardTodd(1,1,6)
groupName[<720,30,3>] := "G21";    // ShephardTodd(21)
groupName[<1152,24,2>] := "G28";    // ShephardTodd(28)
groupName[<1200,30,5>] := "G17";    // ShephardTodd(17)
groupName[<1296,9,3>] := "G26";    // ShephardTodd(26)
groupName[<1536,24,1>] := "D4(4)";    // ShephardTodd(4,4,4)
groupName[<1800,0,6>] := "G18";    // ShephardTodd(18)
groupName[<1920,20,1>] := "D5";    // ShephardTodd(2,2,5)
groupName[<2160,45,1>] := "G27";    // ShephardTodd(27)
groupName[<3072,28,2>] := "B4(4)";    // ShephardTodd(4,2,4)
groupName[<3600,30,7>] := "G19";    // ShephardTodd(19)
groupName[<5040,21,1>] := "A6";    // ShephardTodd(1,1,7)
groupName[<7680,40,1>] := "G29";    // ShephardTodd(29)
groupName[<9720,30,1>] := "D5(3)";    // ShephardTodd(3,3,5)
groupName[<14400,60,1>] := "G30";    // ShephardTodd(30)
groupName[<23040,30,1>] := "D6";    // ShephardTodd(2,2,6)
groupName[<40320,28,1>] := "A7";    // ShephardTodd(1,1,8)
groupName[<46080,60,1>] := "G31";    // ShephardTodd(31)
groupName[<51840,36,1>] := "G35";    // ShephardTodd(35)
```

```
groupName[<51840,45,1>] := "G33";    // ShephardTodd(33)
groupName[<155520,0,2>] := "G32";    // ShephardTodd(32)
groupName[<174960,45,1>] := "D6(3)";    // ShephardTodd(3,3,6)
groupName[<322560,42,1>] := "D7";    // ShephardTodd(2,2,7)
groupName[<362880,36,1>] := "A8";    // ShephardTodd(1,1,9)
groupName[<2903040,63,1>] := "G36";    // ShephardTodd(36)
groupName[<5160960,56,1>] := "D8";    // ShephardTodd(2,2,8)
groupName[<39191040,126,1>] := "G34";    // ShephardTodd(34)
groupName[<696729600,120,1>] := "G37";    // ShephardTodd(37)

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

Components

Given a group G acting on a union T of its conjugacy classes, find representatives for the conjugacy classes.

```
class_reps := function(G, T)
    reps := [];
    while #T gt 0 do
        t := Rep(T);
        Append(~reps, t);
        S := Class(G, t);
        T := { x : x in T | x notin S };
    end while;
    return reps;
end function;
```

A complex reflection group W is the direct product of uniquely determined irreducible reflection subgroups W_1, W_2, ..., W_n; these are the *components* of W. In these notes it is assumed that W acts on the right on a (left) complex vector space V and that W preserves a non-degenerate hermitian form on V. Then $V = V_1 \perp V_2 \perp \cdots \perp V_n$ and W_s is generated by the reflections whose roots belong to W_s.

Given the global context and a reflection subgroup H, return a sequence of indecomposable components. There is no check that H is generated by its reflections.

```
components := function(ctx, H)
    refs := [ r : r in &join ctx`refs | r in H ];
    rtGrps := {@ Stabiliser(H, Basis(Eigenspace(r, 1))) : r in refs @};
    rtRefs := [ [ r : r in refs | r in H ] : H in rtGrps ];
    seq := [];
    X := [ 1..#rtRefs ];
    while not IsEmpty(X) do
        n := X[1];
        Exclude(~X, n);
```

```
        C := rtRefs[n];
        while exists(ndx){ x : c in C, r in rtRefs[x], x in X | c*r ne r*c } do
            C cat:= rtRefs[ndx];
            EXCLUDE(~X, ndx);
        end while;
        APPEND(~seq, C);
    end while;
    return seq;
end function;
```

Standard names

We keep track of the reflection subgroups via their names. For example, if H is the first reflection subgroup of W found to have type A_3 it will have name `A3`. If K is another subgroup of type A_3, not conjugate in W to H, it will have name `A3-2`, and so on. In this case `A3` is the *base name* of both H and K.

Return the concatenation of the sorted list of base names of the components of H.

```
getTag := function(ctx, H)
    W := ctx`group;
    if forall{ r : r in GENERATORS(W) | r in H } then
        refno2 := &+[ Z | c[2] : c in ctx`refrep | c[1] eq 2 ];
        sform := [name(#W, refno2, #ctx`refrep)];
    else
        sform := [];
        for C in components(ctx, H) do
            refno2 := #{ r : r in C | ORDER(r) eq 2 };
            H := sub<W|C>;
            refno := #class_reps(H, C);
            APPEND(~sform, name(#H, refno2, refno));
        end for;
    end if;
    return &cat SORT(sform);
end function;
```

A stand-alone version of getTag. Assumes indecomposable.

```
cTag := function(H)
    refreps := [ c : c in CLASSES(H) | RANK(MATRIX(c[3]) - H.0) eq 1 ];
    refno2 := &+[ Z | c[2] : c in refreps | c[1] eq 2 ];
    return "<"*toStr(#H)*"|"*toStr(refno2)*"|"*toStr(#refreps)*">";
end function;
```

The parameter *refgroup* in the functions which follow is an associative array which associates the name of a conjugacy class of reflection subgroups with a representative subgroup. It is created in rankOne.

If the base name of H is already in use, find the next available index, otherwise set the index to 1.

```
baseName := function(ctx, refgroup, H)
    base := getTag(ctx, H);
    tag := base;
    ndx := 1;
    while tag in KEYS(refgroup) do
        ndx +:= 1;
        tag := base*"-"*toStr(ndx);
    end while;
    return base, ndx;
end function;
```

2 Setting up

```
context := recformat< group : GRPMAT, refrep : SEQENUM, refs : SEQENUM >;
```

group is the complex reflection group;
refrep is a sequence of representatives for the conjugacy classes of reflections;
refs is the sequence of conjugacy classes of reflections.

Given a complex reflection group W, return a *context* record.

```
prepare := function(W)
    cc := CLASSES(W);
    refreps := [ c : c in cc | RANK(MATRIX(c[3]) - W.0) eq 1 ];
    R := [ CLASS(W, c[3]) : c in refreps ];
    return rec< context | group := W, refrep := refreps, refs := R >;
end function;
```

If *ndx* is an integer, the function setup returns the *context* record for the primitive complex reflection group whose Shephard and Todd index is *ndx*. If *ndx* is a sequence $[m, p, n]$ it returns the *context* record for the imprimitive group $G(m, p, n)$.

```
setup := function(ndx)
    if TYPE(ndx) eq RNGINTELT then
        W := SHEPHARDTODD( ndx : NUMFLD);
    else
        m, p, n := EXPLODE(ndx);
        W := SHEPHARDTODD(m, p, n);
    end if;
    return prepare(W);
end function;
```

The function extendGrp will be applied to a sequence of subgroups already constructed to produce a larger list. Some subgroups that we produce may be conjugate to groups already indexed by *refroup*. The following function checks for conjugacy.

```
ccCheck := function(W, baseTag, limit, refgroup, H)
    tag := baseTag;
    if tag in KEYS(refgroup) then
        ndx := 1;
        while tag in KEYS(refgroup) and not ISCONJUGATE(W, refgroup[tag], H) do
            ndx +:= 1;
            tag := baseTag * "-" * toStr(ndx);
        end while;
        if ndx lt limit then return true, tag; end if;
    end if;
    return false, _;
end function;
```

Reflection subgroups of rank 1

The function rankOne begins the process of finding all reflection subgroups of W and their simple extensions.

```
rankOne := function(ctx)
    refgroup := ASSOCIATIVEARRAY(PARENT(""));
    names := [];
    for H in [ sub< ctx`group | c[3] > : c in ctx`refrep ] do
        baseTag, n := baseName(ctx, refgroup, H);
        if n eq 1 then
            tag := baseTag;
        else
            found, theTag := ccCheck(ctx`group, baseTag, n, refgroup, H);
            if found then
                continue;
            else
                tag := baseTag * "-" * toStr(n);
            end if;
        end if;
        APPEND(~names, tag);
        refgroup[tag] := H;
    end for;
    return names, refgroup;
end function;
```

Extending subgroups

The list of simple extensions of a representative of a conjugacy class of reflection subgroups of a given type is held in the following associative array.

```
extension := AssociativeArray(Parent(""));
```

The function extendGrp finds all extensions, up to conjugacy, of the subgroup H.

```
extendGrp := function(ctx, H)
    W := ctx`group;
    N := Normaliser(W, H);
    X := [ r : r in &join ctx`refs | r notin H ];
    ccreps := class_reps(N, X);
```

For each orbit of N on the reflections not in H we produce a simple extension by adjoining a representative of the orbit. The extension is added to the list only if it is not conjugate in W to an earlier extension.

```
    return [ G : r in ccreps |
                not exists{ E : E in Self() | IsConjugate(W, E, G) }
                                where G is sub< W | H, r > ];
end function;
```

---

simpleExtensions begins with the list of names of conjugacy classes of reflection subgroups in *names* and recursively finds representatives (in *refgroup*) of all simple extensions (in *extension*). If VERBOSE is true, diagnostic information is printed.

```
simpleExtensions := procedure(~names, ~extension, ~refgroup, ctx :
    VERBOSE := false)
    ndx := 0;
    while ndx lt #names do
        enames := {@ @};
        ndx +:= 1;
        tag := names[ndx];
        if VERBOSE then print "Extending:", tag; end if;
        H0 := refgroup[tag];
        extn := extendGrp(ctx, H0);
        for H in extn do
            baseTag, n := baseName(ctx, refgroup, H);
            if baseTag in names then
                found, theTag := ccCheck(ctx`group, baseTag, n, refgroup, H);
                if not found then
                    etag := n eq 1 select baseTag else baseTag * "-" * toStr(n);
                    refgroup[etag] := H;
                    Include(~enames, etag);
                    Append(~names, etag);
                    if VERBOSE then print "  Adding", etag; end if;
                else
                    Include(~enames, theTag);
```

```
                if VERBOSE then print "   Skipping", theTag; end if;
            end if;
        else
            refgroup[baseTag] := H;
            INCLUDE(~enames, baseTag);
            APPEND(~names, baseTag);
            if VERBOSE then print "   Adding", baseTag; end if;
        end if;
    end for;
    extension[tag] := SETSEQ(enames);
    if VERBOSE then print extension[tag]; end if;
  end while;
end procedure;
```

Parabolic closure

If H is a subgroup of W, its space of fixed points is written V^H. If U is a subset of V, its pointwise stabiliser in W is written $W(U)$.

If $H \subseteq W$ is parabolic, then $H = W(U)$ for some subspace U of V. Thus $H \subseteq W(V^H) \subseteq W(U) = H$ and so $H = W(V^H)$. Therefore, if H is a reflection subgroup of W, the subgroup $W(V^H)$ is the smallest parabolic subgroup containing H; it is the *parabolic closure* of H.

```
fix := func< G | &meet[EIGENSPACE(r, 1) : r in GENERATORS(G)] >;

parabolicClosure := function(ctx, K)
    W := ctx`group;
    F := fix(K);
    R := &join ctx`refs;
    T := {W| r : r in R | r notin K and F subset EIGENSPACE(r, 1)};
    L := K;
    while #T gt 0 do
        x := REP(T);
        L := sub<W | L, x >;
        T := { t : t in T | t notin L };
    end while;
    return L;
end function;

isParabolic := func< G, P | P eq parabolicClosure(G, P) >;
```

3 Miscellaneous checks

The procedure testExt checks to see if there is a non-parabolic subgroup with a parabolic simple extension of larger rank.

```
rank := func< G | DIMENSION(G) − DIMENSION(fix(G)) >;

testExt := procedure(n)
    ctx := setup(n);
    names, refgroup := rankOne(ctx);
    extension := ASSOCIATIVEARRAY(PARENT(""));
    simpleExtensions(~names, ~extension, ~refgroup, ctx);
    for tag in names do
        H := refgroup[tag];
        if not isParabolic(ctx, H) then
            m := rank(H);
            for extn in extension[tag] do
                K := refgroup[extn];
                if rank(K) gt m then
                    flag := isParabolic(ctx, K);
                    print tag, extn, flag;
                    if flag then print "~~~COUNTER EXAMPLE~~~~~~~~~"; end if;
                end if;
            end for;
        end if;
    end for;
end procedure;
```

The function extends checks whether the group with name B is a simple extension of the group with name A.

```
extends := function(A, B, refgroup, extension, W)
    flag, G := ISDEFINED(refgroup, B);
    if flag then
        flag, E := ISDEFINED(extension, A);
        if flag then
            return exists(X){ X : X in E | ISCONJUGATE(W, X, G) };
        else
            return "Extensions not defined";
        end if;
    else
        return "Identity not known";
    end if;
end function;
```

4 Tables

In this section we define a procedure which computes the simple extensions for a given complex reflection group and prints the results as a table.

For finer control over the process, use the functions and procedures from the preceding sections.

If SORTED is true, the list of reflection subgroups is arranged in lexicographic order of (`rank`, `number of generators`, `group order`).

```
getList := function(n : VERBOSE := false, SORTED := false)
    ctx := setup(n);
    names, refgroup := rankOne(ctx);
    extension := ASSOCIATIVEARRAY(PARENT(""));
    simpleExtensions(~names, ~extension, ~refgroup, ctx : VERBOSE := VERBOSE);
    if SORTED then
        fn := function(A, B)
            H := refgroup[A]; h := NGENS(H); rH := rank(H);
            K := refgroup[B]; k := NGENS(K); rK := rank(K);
            return (rH eq rK) select ((h eq k) select #H - #K else h - k)
                                else rH - rK;
        end function;
        SORT(~names, fn);
    end if;
    return names, refgroup, extension, ctx;
end function;

printTable := procedure(n : VERBOSE := false, SORTED := false)
    names, refgroup, extension, ctx := getList(n : VERBOSE := VERBOSE,
                SORTED := SORTED);
    maxlen := MAX([ #nm : nm in names ]);
    W := ctx`group;
    for X in names do
        H := refgroup[X];
        para := isParabolic(ctx, H) select "P" else " ";
        if SORTED then
            rk := rank(H);
            ngens := NGENS(H);
            nconj := INDEX(W, NORMALISER(W, H));
            h := maxlen - #X;
            pad := " "^h * "|";
            print para, rk, ngens, nconj, "|", X, pad, extension[X];
        else
            print para, "|", X, "|", extension[X];
        end if;
    end for;
end procedure;
```