

HANDBOOK FOR MAGMA REE PACKAGE

HENRIK BÄÄRNHIELM

The MAGMA Ree package provides functionality for constructive recognition and constructive membership testing of the small Ree groups ${}^2G_2(q) = \text{Ree}(q)$, with $q = 3^{2m+1}$ for some $m > 0$.

The main intrinsics of the package are `Ree(FldFin F)` which returns the standard copy of the small Ree group over the field F , `RecogniseRee(GrpMat G)` which performs constructive recognition of $G \cong \text{Ree}(q)$, `ReeElementToWord(GrpMat G, GrpMatElt g)` which returns a `GrpSLPElt` for g in the generators of G , and `ReeRecognition(GrpMat G)` which is a non-constructive test for isomorphism between G and $\text{Ree}(q)$.

Sylow p -subgroups can be found using the `ReeSylow(GrpMat G, RngIntElt p)` intrinsic. To find conjugating elements between Sylow p -subgroups, an intrinsic `ReeSylowConjugacy` is provided. The latter is not available when p is odd and divides $q^3 + 1$.

A list of representatives of the conjugacy classes of maximal subgroups can be found using the `ReeMaximalSubgroups(GrpMat G)` intrinsic, and the intrinsic `ReeMaximalSubgroupsConjugacy` can be used to find conjugating elements between maximal subgroups.

There are a few verbose flags used in the package.

- `ReeGeneral`, for the general routines.
- `ReeStandard`, for the routines related to the standard copy.
- `ReeConjugate`, for the routines related to conjugation.
- `ReeTensor`, for the routines related to tensor decomposition.
- `ReeMembership`, for the routines related to membership testing.
- `ReeCrossChar`, for the routines related to cross-characteristic representations.
- `ReeSylow`, for the routines related to Sylow subgroups.
- `ReeTrick`, for the routines related to the stabiliser trick.
- `ReeInvolution`, for the routines related to involution centralisers.
- `ReeSymSquare`, for the routines related to symmetric square decomposition.
- `ReeMaximals`, for the routines related to maximal subgroups.

All the flags can be set to values up to 10, with higher values resulting in more output.

1. INTRINSICS

`Ree(F) : FldFin -> GrpMat`

The field F must have size $q = 3^{2m+1}$ for some $m > 0$. Returns $\text{Ree}(q)$ on its standard generators.

`RecogniseRee(G : parameters) : GrpMat -> BoolElt, Map, Map, Map, Map`
Parameters:

`Verify, BOOLELT, Default : true`

Date: 2007-04-22.

FieldSize, RngIntElt

G is absolutely irreducible and defined over minimal field. Constructively recognise G as a Ree group. If G is isomorphic to $\text{Ree}(q)$ where q is the size of the defining field of G , then return:

- (1) Isomorphism from G to $\text{Ree}(q)$.
- (2) Isomorphism from $\text{Ree}(q)$ to G .
- (3) Map from G to the word group of G .
- (4) Map from the word group of G to G .

The isomorphisms are composed of maps that are defined by rules, so **Function** can be used on each component, hence avoiding unnecessary built-in membership testing.

The word group is the **GrpSLP** which is the parent of the elements returned by **ReeElementToWord**. In general this is not the same as **WordGroup(G)**, but is created from it using **AddRedundantGenerators**.

If **Verify** is true, then it is checked that G is isomorphic to $\text{Ree}(q)$, using **ReeRecognition**, otherwise this is not checked. In that case, **FieldSize** must be set to the correct value of q .

The algorithms for constructive recognition are those of [1].

ReeElementToWord(G, g) : GrpMat, GrpMatElt -> BoolElt, GrpSLPElt

If G has been constructively recognised as a Ree group, and if g is an element of G , then return **true** a **GrpSLPElt** from the word group of G which evaluates to g , else return **false**.

This facilitates membership testing in G .

ReeRecognition(G) : GrpMat -> BoolElt, RngIntElt

Determine (non-constructively) if G is isomorphic to $\text{Ree}(q)$. The corresponding q is also returned.

If G is over a field of characteristic not 3 or has degree greater than 7, the Monte Carlo algorithm of **IdentifyLieType** is used. If G has degree 7 and is over a field of characteristic 3, then a fast Las Vegas algorithm is used.

ReeIrreducibleRepresentation(F, twists : parameters) : FldFin, SeqEnum[RngIntElt] -> GrpMat

Parameters:

CheckInput, BOOLELT, Default : true

F must have size $q = 3^{2m+1}$ for some $m > 0$, and *twists* should be a sequence of n distinct pairs of integers (i, j) where i is 7 or 27 and j in the range $[0 \dots 2m]$.

Return an absolutely irreducible representation of $\text{Ree}(q)$, a tensor product of twisted powers of the representation of dimension 7 or 27, where the twists are given by the input sequence.

If **CheckInput** is true, then it is verified that F and *twists* satisfy the above requirements. Otherwise this is not checked.

ReeSylow(G, p) : GrpMat, RngIntElt -> GrpMat, SeqEnum

If G has been constructively recognised as a Ree group, and if p is a prime number, return a random Sylow p -subgroup S of G .

Also returns a list of **GrpSLPElt** from the word group of G , of the generators of S . If p does not divide $|G|$, then the trivial subgroup is returned.

`ReeSylowConjugacy(G, R, S, p) : GrpMat, GrpMat, GrpMat, RngIntElt -> GrpMatElt, GrpSLPElt`

If G has been constructively recognised as a Suzuki group, if p is a prime number and if R and S are Sylow p -subgroups of G , then return an element g of G that conjugates R to S . A `GrpSLPElt` from the word group of G , that evaluates to g , is also returned.

Currently, this is not implemented if p is odd and divides $q^3 + 1$.

`ReeMaximalSubgroups(G) : GrpMat -> SeqEnum, SeqEnum`

If G has been constructively recognised as a Ree group, find a list of representatives of the maximal subgroups of G .

Also returns lists of `GrpSLPElt` of the generators of the subgroups, from the word group of G .

`ReeMaximalSubgroupsConjugacy(G, R, S) : GrpMat, GrpMat, GrpMat -> GrpMatElt, GrpSLPElt`

If G has been constructively recognised as a Ree group and if R and S are conjugate maximal subgroups of G , then return an element g of G that conjugates R to S . A `GrpSLPElt` from the word group of G , that evaluates to g , is also returned.

Currently, this is not implemented if R and S are normalisers of Sylow p -subgroups with p odd and dividing $q^3 + 1$.

2. EXAMPLES

Example showing the basic features.

```
> // Let's try a conjugate of the the standard copy
> F := GF(3, 3);
> G := Ree(F);
> G ^:= Random(Generic(G));
> // perform non-constructive recognition
> flag, q := ReeRecognition(G);
> print flag, q eq #F;
true true
> // perform constructive recognition
> flag, iso, inv, g2slp, slp2g := RecognizeRee(G);
> print flag;
true
> // the explicit isomorphisms are defined by rules
> print iso, inv;
Mapping from: GrpMat: G to MatrixGroup(7, GF(3^3)) given by a rule [no inverse]
Mapping from: MatrixGroup(7, GF(3^3)) to GrpMat: G given by a rule [no inverse]
> // so we can use Function to avoid Magma built-in membership testing
> // we might not obtain the shortest possible SLP
> w := Function(g2slp)(G.1);
> print #w;
342
> // and the algorithm is probabilistic, so different executions will most
> // likely give different results
> ww := Function(g2slp)(G.1);
> print w eq ww;
false
> // the resulting SLPs are from another word group
```

```

> W := WordGroup(G);
> print NumberOfGenerators(Parent(w)), NumberOfGenerators(W);
7 3
> // but can be coerced into W
> flag, ww := IsCoercible(W, w);
> print flag;
true
> // so there are two ways to get the element back
> print slp2g(w) eq Evaluate(ww, UserGenerators(G));
true
> // an alternative is this intrinsic, which is better if the elements are not
> // known to lie in the group
> flag, ww := ReeElementToWord(G, G.1);
> print flag, slp2g(w) eq slp2g(ww);
true true
> // let's try something just outside the group
> H := Omega(7, #F);
> flag, ww := ReeElementToWord(G, H.1);
> print flag;
false
> // in this case we will not get an SLP
> ww := Function(g2slp)(H.1);
> print ww;
false

```

Example about Sylow subgroups.

```

> // let's try a small field
> m := 1;
> F := GF(3, 2 * m + 1);
> q := #F;
> print q;
27
> G := Ree(F);
> // let's try a conjugate
> G ^:= Random(Generic(G));
> // also move to another generating set
> G := DerivedGroupMonteCarlo(G);
> print NumberOfGenerators(G);
19
> // First we must recognise the group
> time flag, iso, inv, g2slp, slp2g := RecogniseRee(G);
Time: 2.000
> // what about creating Sylow subgroups?
> p := Random([x[1] : x in Factorization(q - 1) | x[1] gt 2]);
> print p;
13
> time R := ReeSylow(G, p);
Time: 0.020
> time S := ReeSylow(G, p);
Time: 0.010
> // that was easy, as is conjugating them
> time g, slp := ReeSylowConjugacy(G, R, S, p);
Time: 0.010

```

```

> time print R^g eq S;
true
Time: 0.000
> // in this case we also automatically get an SLP of the conjugating element
> print slp2g(slp) eq g;
true
> // those Sylow subgroups are cyclic, and we know the order
> print #R, NumberOfGenerators(R);
13 1
> // creating Sylow 3-subgroups is harder, since there are lots of generators
> time R := ReeSylow(G, 3);
Time: 0.000
> time S := ReeSylow(G, 3);
Time: 0.010
> print NumberOfGenerators(R), #R;
3 19683
> // but conjugation is easy
> time g, slp := ReeSylowConjugacy(G, R, S, 3);
Time: 0.000
> // verifying the conjugating element can be expensive
> time print R^g eq S;
true
Time: 0.870
> // Sylow 2-subgroups are small
> time R := ReeSylow(G, 2);
Time: 0.280
> print NumberOfGenerators(R), #R;
3 8
> // but slightly harder to conjugate
> time S := ReeSylow(G, 2);
Time: 0.200
> time g, slp := ReeSylowConjugacy(G, R, S, 2);
Time: 0.190
> time print R^g eq S;
true
Time: 0.000

```

Example showing a larger field case.

```

> // let's try a larger field
> m := 15;
> F := GF(3, 2 * m + 1);
> q := #F;
> print q;
617673396283947
> G := Ree(F);
> // let's try a conjugate
> G^ := Random(Generic(G));
> // also move to another generating set
> G := DerivedGroupMonteCarlo(G);
> print NumberOfGenerators(G);
19
> // non-constructive recognition is now a bit harder
> time ReeRecognition(G);

```

```

true 617673396283947
Time: 34.410
> // and is your machine up for this?
> time flag, iso, inv, g2slp, slp2g := RecogniseRee(G);
Time: 368.190
> // each call to constructive membership is then easy
> R := RandomProcess(G);
> g := Random(R);
> time w := Function(g2slp)(g);
Time: 2.080
> // evaluating SLPs always takes some time
> time print slp2g(w) eq g;
true
Time: 1.910

```

Example about maximal subgroups.

```

> // let's try a small field
> m := 1;
> F := GF(3, 2 * m + 1);
> q := #F;
> print q;
27
> G := Ree(F);
> // let's try a conjugate
> G ^:= Random(Generic(G));
> // also move to another generating set
> G := DerivedGroupMonteCarlo(G);
> print NumberOfGenerators(G);
19
> // first we must recognise the group
> time flag, iso, inv, g2slp, slp2g := RecogniseRee(G);
Time: 1.870
> // try finding the maximal subgroups
> time l, slps := ReeMaximalSubgroups(G);
Time: 0.400
> // should have a parabolic, an involution centraliser, three Frobenius group\
s
> print #l;
5
> // we can conjugate all except Frobenius groups
> // but verifying the conjugating element can be expensive
> r := Random(G'RandomProcess);
> time g, slp := ReeMaximalSubgroupsConjugacy(G, l[1], l[1]^r);
Time: 0.040
> time print l[1]^g eq l[1]^r;
true
Time: 1.240
> time g, slp := ReeMaximalSubgroupsConjugacy(G, l[2], l[2]^r);
Time: 0.040
> time print l[2]^g eq l[2]^r;
true

```

REFERENCES

1. H. Bäärnhielm, *Recognising the Ree groups in their natural representations*, (2006), preprint.

SCHOOL OF MATHEMATICAL SCIENCES, QUEEN MARY, UNIVERSITY OF LONDON, MILE END
ROAD, LONDON E1 4NS, UNITED KINGDOM

URL: <http://www.maths.qmul.ac.uk/~hb/>

E-mail address: h.baarnhielm@qmul.ac.uk