MajoranaAlgebras

A package for constructing Majorana algebras and representations

1.5.2

7 July 2024

Markus Pfeiffer

Madeleine Whybrow

Markus Pfeiffer
Email: markus.pfeiffer@st-andrews.ac.uk
Homepage: https://markusp.morphism.de/

Madeleine Whybrow
Email: mlw10@ic.ac.uk
Homepage: https://madeleinewhybrow.wordpress.com
Address: Department of Mathematics, Imperial College, South Kensington, SW7 2AZ

Copyright

Contents

1 Introduction **4**
- 1.1 A quick guide . . . 4
- 1.2 Understanding the output . . . 5
- 1.3 Info levels . . . 6

2 Shapes of a Majorana representation **7**
- 2.1 The shapes functions . . . 7

3 Majorana representations **9**
- 3.1 The main function . . . 9
- 3.2 The n-closed function . . . 9

4 Functions for calculating with Majorana representations **11**
- 4.1 Calculating products . . . 11
- 4.2 Basic functions . . . 12
- 4.3 The subalgebra structure . . . 12

5 Functions for testing Majorana representations **14**
- 5.1 The main function . . . 14
- 5.2 Other functions . . . 14

6 Orbital Structures **16**
- 6.1 Examples . . . 16

7 Signed Permutations **20**
- 7.1 Different Representations . . . 21
- 7.2 Low-Level Descriptions . . . 21

Index **23**

Chapter 1

Introduction

1.1 A quick guide

In order to construct the Majorana representation of a group *G* with respect to a set of involutions *T*, you must first call `ShapesOfMajoranaRepresentation` (2.1.1).

Example
```
gap> G := AlternatingGroup(5);;
gap> T := AsList( ConjugacyClass(G, (1,2)(3,4)));;
gap> input := ShapesOfMajoranaRepresentation(G,T);;
```

This function outputs a record. One component of this record is labelled *shapes* and contains the possible shapes of a Majorana representation of the form (G,T,V).

Example
```
gap> input.shapes;
[ [ "1A", "2B", "5A", "3C", "5A" ], [ "1A", "2B", "5A", "3A", "5A" ],
  [ "1A", "2A", "5A", "3C", "5A" ], [ "1A", "2A", "5A", "3A", "5A" ] ]
```

To construct the Majorana representation with shape at position *i* of this list, call the function `MajoranaRepresentation` (3.1.1) with *input* as its first argument and *i* as its second.

Example
```
gap> rep := MajoranaRepresentation(input, 1);;
gap> rep.shape;
[ "1A", "2B", "5A", "3C", "5A" ]
```

There are then a number of functions (see 4) that one case use on the (potentially incomplete) Majorana representation that this function has outputted.

Example
```
gap> MAJORANA_IsComplete(rep);
true
gap> MAJORANA_Dimension(rep);
21
```

If an incomplete algebra is returned then the function `NClosedMajoranaRepresentation` (3.2.1) can be used to attempt to find the 3-closed part of the algebra.

Example
```
gap> G := AlternatingGroup(5);;
gap> T := AsList( ConjugacyClass(G, (1,2)(3,4)));;
gap> input := ShapesOfMajoranaRepresentation(G,T);;
gap> input.shapes;
[ [ "1A", "2B", "5A", "3C", "5A" ], [ "1A", "2B", "5A", "3A", "5A" ],
  [ "1A", "2A", "5A", "3C", "5A" ], [ "1A", "2A", "5A", "3A", "5A" ] ]
gap> rep := MajoranaRepresentation(input, 2);;
gap> MAJORANA_IsComplete(rep);
false
gap> NClosedMajoranaRepresentation(rep);;
gap> MAJORANA_IsComplete(rep);
true
gap> MAJORANA_Dimension(rep);
46
```

1.2 Understanding the output

Note that all vectors and matrices are given in sparse matrix format, as provided by the GAP package `Gauss`. *If* `mat` *is such a matrix then the integers in* `mat!.indices` *refer to a spanning set of the algebra indexed by the list* `rep.setup.coords`. *The list* `mat!.entries` *give their corresponding coefficients.*

The function `MajoranaRepresentation` (3.1.1) outputs a record that encodes the information required to perform calculations in the Majorana representation that has been calculated. The record contains the following components.

`group`
: The group *G*, as inputted by the user.

`involutions`
: The set *T*, as inputted by the user.

`shape`
: The shape of the representation, as chosen by the user in the input of `MajoranaRepresentation` (3.1.1).

`eigenvalues`
: A list whose values give the eigenvalues of the adjoint action of the axes of the algebra. In this case, it must be equal to (or a subset of) *[0, 1/4, 1/32]*. Note that we omit the eigenvalue 1 as we assume the algebra to be primitive.

`axioms`
: A string representing the axiomatic setting of the algebra's construction, potentially chosen by the user with the *options* record in the input of `MajoranaRepresentation` (3.1.1).

`setup`
: Is itself a record, containing (among others) the following components.

    `coords`
    : A list whose elements index a spanning set of the algebra.

`nullspace`
: Again a record such that *`nullspace.vectors`* gives a basis of the nullspace of the algebra (as the elements *`rep.setup.coords`* are not necessarily linearly independent).

`orbitreps`
: A list of indices giving the representatives of the orbits of the action of the group *`G`* on *`T`*.

`pairreps`
: A list of pairs of indices giving representatives of the orbitals of the action of the group *`G`* on *`rep.setup.coords`*.

`algebraproducts`
: A list where the vector at position *`i`* denotes the algebra product of the two spanning set vectors whose indices (in *`rep.setup.coords`*) are given by *`rep.setup.pairreps[i]`*. If the *`i`*th entry is set to *`false`* then this algebra product has not yet been found and the algebra is incomplete.

`innerproducts`
: Performs the same role as *`algebraproducts`* except that, instead of vectors, the entries are rational numbers denoting the inner product between two spanning set vectors.

`evecs`
: A list where if *`i`* is contained in *`rep.setup.orbitreps`* then *`rep.evecs[i]`* is bound to a record. This record has components *`"ev"`* where *`ev`* is an eigenvalue contained in *`rep.eigenvalues`*. This component gives a basis for the eigenspace of the axis corresponding to *`rep.involutions[i]`* with eigenvalue *`ev`*.

1.3 Info levels

1.3.1 InfoMajorana

▷ `InfoMajorana` (info class)

The default info level of *`InfoMajorana`* is 0. No information is printed at this level. If the info level is at least 10 then *`Success`* is printed if the algorithm has produced a complete Majorana algebra, otherwise *`Fail`* is printed. If the info level is at least 20 then more information is printed about the progress of the algorithm, up to a maximum info level of 100.

Chapter 2

Shapes of a Majorana representation

2.1 The shapes functions

2.1.1 ShapesOfMajoranaRepresentation

▷ `ShapesOfMajoranaRepresentation(G, T)` (function)

Returns: a record with a component *shapes*

Takes a group G and a G-invariant set of generating involutions T. Returns a list of possible shapes of a Majorana Representation of the form (G,T,V) that is stored in the *shapes* component of the output.

2.1.2 ShapesOfMajoranaRepresentationAxiomM8

▷ `ShapesOfMajoranaRepresentationAxiomM8(G, T)` (function)

Returns: a record with a component *shapes*

Performs exactly the same function as `ShapesOfMajoranaRepresentation` (2.1.1) but gives only those shapes at obey axiom M8. That is to say, we additionally assume that if $t,s \in T$ such that $|ts| = 2$ then the dihedral subalgebra $\langle\langle a_t, a_s\rangle\rangle$ is of type $2A$ if and only if $ts \in T$ (and otherwise is of type $2B$).

2.1.3 MAJORANA_IsSixTranspositionGroup

▷ `MAJORANA_IsSixTranspositionGroup(G, T)` (function)

Returns: true if (G,T) is a 6-transposition group, otherwise returns false

For a group G and a subset T of G, returns true if all of the following conditions are satisfied: *T is a set of involutions that generate G; *T is closed under conjugation by G; *the order of the product of two elements of T is at most 6.

2.1.4 MAJORANA_RemoveDuplicateShapes

▷ `MAJORANA_RemoveDuplicateShapes(input)` (function)

If an automorphism of the group G stabilises the set T then it induces an action on the pairs of elements of T and therefore on the shapes of a possible Majorana representation of the form (G,T,V). If one shape is mapped to another in this way then their corresponding algebras must be isomorphic.

This function takes the record *`input`* as produced by the function `ShapesOfMajoranaRepresentation` (2.1.1) or `ShapesOfMajoranaRepresentationAxiomM8` (2.1.2) and replaces *`input.shapes`* with a list of shapes such that no two can be mapped to each other by an automorphism of *`G`*.

Chapter 3

Majorana representations

3.1 The main function

3.1.1 MajoranaRepresentation

▷ `MajoranaRepresentation(`*`input, index[, options]`*`)` (function)

Returns: a record giving a Majorana representation

This takes two or three arguments, the first of which must be the output of the function `ShapesOfMajoranaRepresentation` (2.1.1) and the second of which is the index of the desired shape in list *`input.shapes`*.

If the optional argument *`options`* is given then it must be a record. The following components of *`options`* are recognised:

`axioms`

This component must be bound to the string *`"AllAxioms"`* or *`"NoAxioms"`*. If bound to *`"AllAxioms"`* then the algorithm assumes the axioms 2Aa, 2Ab, 3A, 4A and 5A as in Seress (2012). If bound to *`"NoAxioms"`* then the algorithm only assumes the Majorana axioms M1 - M7. The default value is *`"AllAxioms"`*.

`form`

If this is bound to *`true`* then the algorithm assume the existence of an inner product (as in the definition of a Majorana algebra). Otherwise, if bound to *`false`* then no inner product is assumed (and we are in fact constructing an axial algebra that satisfies the Majorana fusion law). The default value is *`true`*.

`embedding`

If this is bound to *`true`* then the algorithm first attempts to construct large subalgebras of the final representation before starting the main construction. The default value is *`false`*.

3.2 The n-closed function

A Majorana algebra V generated by a set of axes A is called n-closed if it is spanned as a vector space by products of elements of A of length at most n. As most known Majorana algebras are 2-closed, the function `MajoranaRepresentation` (3.1.1) only attempts to construct the 2-closed part.

If it is not successful then the output is a partial Majorana representation, i.e. a Majorana representation with some missing algebra products. In this case, the function `MAJORANA_IsComplete` (4.2.1) returns false.

If the user wishes, they may then pass this incomplete Majorana representation to the function `NClosedMajoranaRepresentation` (3.2.1) in order to attempt construction of the 3-closed part. This process may then be repeated as many times as the user wishes.

3.2.1 NClosedMajoranaRepresentation

▷ `NClosedMajoranaRepresentation(`*rep*`)` (function)

Takes as its input an incomplete Majorana representation rep that has been generated using the function `MajoranaRepresentation` (3.1.1). Again runs the main algorithm in order to attempt construction of the 3-closed part of the algebra. If the function `NClosedMajoranaRepresentation` is called n times on the same Majorana representation rep then this representation will be the $n+2$-closed part of the algebra.

Chapter 4

Functions for calculating with Majorana representations

4.1 Calculating products

4.1.1 MAJORANA_AlgebraProduct

▷ MAJORANA_AlgebraProduct(*u*, *v*, *algebraproducts*, *setup*) (function)

Returns: the algebra product of vectors *u* and *v*

The arguments *u* and *v* must be row vectors in sparse matrix format. The arguments *algebraproducts* and *setup* must be the components with these names of a representation as outputted by `MajoranaRepresentation` (3.1.1). The output is the algebra product of *u* and *v*, also in sparse matrix format.

4.1.2 MAJORANA_InnerProduct

▷ MAJORANA_InnerProduct(*u*, *v*, *innerproducts*, *setup*) (function)

Returns: the inner product of vectors *u* and *v*

The arguments *u* and *v* must be row vectors in sparse matrix format. The arguments *innerproducts* and *setup* must be the components with these names of a representation as outputted by `MajoranaRepresentation` (3.1.1). The output is the inner product of *u* and *v*.

Example
```
gap> G := AlternatingGroup(5);;
gap> T := AsList(ConjugacyClass(G, (1,2)(3,4)));;
gap> input := ShapesOfMajoranaRepresentation(G,T);;
gap> rep := MajoranaRepresentation(input, 1);;
gap> Size(rep.setup.coords);
21
gap> u := SparseMatrix( 1, 21, [ [ 1 ] ], [ [ 1 ] ], Rationals);;
gap> v := SparseMatrix( 1, 21, [ [ 17 ] ], [ [ 1 ] ], Rationals);;
gap> MAJORANA_AlgebraProduct(u, v, rep.algebraproducts, rep.setup);
<a 1 x 21 sparse matrix over Rationals>
gap> MAJORANA_InnerProduct(u, v, rep.innerproducts, rep.setup);
-1/8192
```

4.2 Basic functions

4.2.1 MAJORANA_IsComplete

▷ `MAJORANA_IsComplete(rep)` (function)

Returns: true is all algebra products have been found, otherwise returns false

Takes a Majorana representation *rep*, as outputted by `MajoranaRepresentation` (3.1.1). If the representation is complete, that is to say, if the vector space spanned by the basis vectors indexed by the elements in *rep.setup.coords* is closed under the algebra product given by *rep.algebraproducts*, return true. Otherwise, if some products are not known then return false.

4.2.2 MAJORANA_Dimension

▷ `MAJORANA_Dimension(rep)` (function)

Returns: the dimension of the representation *rep* as an integer

Takes a Majorana representation *rep*, as outputted by `MajoranaRepresentation` (3.1.1) and returns its dimension as a vector space. If the representation is not complete (cf. `MAJORANA_IsComplete` (4.2.1)) then this value might not be the true dimension of the algebra.

4.2.3 MAJORANA_Eigenvectors

▷ `MAJORANA_Eigenvectors(index, eval, rep)` (function)

Returns: a basis of the eigenspace of the axis as position *index* with eigenvalue *eval* as a sparse matrix

4.2.4 MAJORANA_Basis

▷ `MAJORANA_Basis(rep)` (function)

Returns: a sparse matrix that gives a basis of the algebra

4.2.5 MAJORANA_AdjointAction

▷ `MAJORANA_AdjointAction(axis, basis, rep)` (function)

Returns: a sparse matrix representing the adjoint action of *axis* on *basis*

Takes a Majorana representation *rep*, as outputted by `MajoranaRepresentation` (3.1.1), a row vector *axis* in sparse matrix format and a set of basis vectors, also in sparse matrix format. Returns a matrix, also in sparse matrix format, that represents the adjoint action of *axis* on *basis*.

4.3 The subalgebra structure

4.3.1 MAJORANA_Subalgebra

▷ `MAJORANA_Subalgebra(vecs, rep)` (function)

Returns: the subalgebra of the representation *rep* that is generated by *vecs*

Takes a Majorana representation *rep*, as outputted by `MajoranaRepresentation` (3.1.1) and a set of vectors *vecs* in sparse matrix format and returns the subalgebra generated by *vecs*, also in sparse matrix format.

4.3.2 MAJORANA_IsJordanAlgebra

▷ `MAJORANA_IsJordanAlgebra(`*`subalg, rep`*`)` (function)

Returns: true if the subalgebra *`subalg`* is a Jordan algebra, otherwise returns false

Takes a Majorana representation *`rep`*, as outputted by `MajoranaRepresentation` (3.1.1) and a subalgebra *`subalg`* of rep. If this subalgebra is a Jordan algebra then function returns true, otherwise returns false.

Example

```
gap> G := G := AlternatingGroup(5);;
gap> T := AsList( ConjugacyClass(G, (1,2)(3,4)));;
gap> input := ShapesOfMajoranaRepresentation(G,T);;
gap> rep := MajoranaRepresentation(input, 2);;
gap> MAJORANA_IsComplete(rep);
false
gap> NClosedMajoranaRepresentation(rep);;
gap> MAJORANA_IsComplete(rep);
true
gap> MAJORANA_Dimension(rep);
46
gap> basis := MAJORANA_Basis(rep);
<a 46 x 61 sparse matrix over Rationals>
gap> subalg := MAJORANA_Subalgebra(basis, rep);
<a 46 x 61 sparse matrix over Rationals>
gap> MAJORANA_IsJordanAlgebra(subalg, rep);
false
```

Chapter 5

Functions for testing Majorana representations

The output of the function `MajoranaRepresentation` (3.1.1) is guaranteed to be a commutative algebra generated by idempotents whose eigenspaces obey the Majorana fusion law. To check that the output is truly a Majorana algebra, one must also check that

- the inner product is a Frobenius form (see `MAJORANA_TestFrobeniusForm` (5.2.1));
- the inner product is positive definite (see `MAJORANA_TestInnerProduct` (5.2.2));
- the inner product obeys axiom M2 (Norton's inequality) (see `MAJORANA_TestAxiomM2` (5.2.3));
- the algebra is primitive (see `MAJORANA_TestPrimitivity` (5.2.4)).

5.1 The main function

5.1.1 MajoranaAlgebraTest

▷ `MajoranaAlgebraTest(`*rep*`)` (function)

Returns: *true* if the algebra given by *rep* is indeed a Majorana algebra.

Note: does not check that the algebra obeys axiom M2 (Norton's inequality), this can be separately tested using `MAJORANA_TestAxiomM2` (5.2.3).

5.2 Other functions

5.2.1 MAJORANA_TestFrobeniusForm

▷ `MAJORANA_TestFrobeniusForm(`*rep*`)` (function)

Returns: *true* if the inner product given by *rep.innerproducts* is a Frobenius form, otherwise returns false.

5.2.2 MAJORANA_TestInnerProduct

▷ `MAJORANA_TestInnerProduct(`*rep*`)` (function)

Returns: *true* if the inner product given by *rep.innerproducts* is positive definite, otherwise returns false.

5.2.3 MAJORANA_TestAxiomM2

▷ `MAJORANA_TestAxiomM2(`*rep*`)` (function)

Returns: *true* if the inner product given by *rep.innerproducts* obeys axiom M2 (Norton's inequality), otherwise returns false.

5.2.4 MAJORANA_TestPrimitivity

▷ `MAJORANA_TestPrimitivity(`*rep*`)` (function)

Returns: *true* if the 1-eigenspaces of all axes are 1-dimensional, otherwise returns false.

Example

```
gap> G := AlternatingGroup(5);;
gap> T := AsList( ConjugacyClass(G, (1,2)(3,4)));;
gap> input := ShapesOfMajoranaRepresentation(G,T);;
gap> rep := MajoranaRepresentation(input, 2);;
gap> NClosedMajoranaRepresentation(rep);;
gap> MAJORANA_IsComplete(rep);
true
gap> MajoranaAlgebraTest(rep);
true
gap> MAJORANA_TestFrobeniusForm(rep);
true
gap> MAJORANA_TestInnerProduct(rep);
true
gap> MAJORANA_TestAxiomM2(rep);
true
gap> MAJORANA_TestPrimitivity(rep);
true
```

Chapter 6

Orbital Structures

The functions for orbital structures are based on recent work in permutation group algorithms. An orbital structure contains information about orbits and stabilisers of a group acting on a set for the purposes of quickly determining representatives, canonising elements, and transversal elements (directed) orbitals (orbits of ordered pairs of elements of the domain), and undirected orbitals, i.e. orbits of sets of size two.

6.1 Examples

To create an orbital structure we need generators for a group, a set, and an action

Example

```
gap> os := OrbitalStructure([
> (1,13,4,14,5)(2,10,12,9,8)(3,7,15,6,11)(16,17,18,20,19),
> (1,2,3)(4,6,5)(7,10,13)(8,12,14)(9,11,15)(16,18,21)(17,19,20) ],
> [1..21],
> OnPoints);;
gap> OrbitalRepresentative(os, [16,15]);
[ 16, 1 ]
gap> c := OrbitalCanonizingElement(os, [16, 15]);
(1,10,9,5,15)(2,7,6,8,4)(3,13,14,11,12)(17,20,18,19,21)
gap> OnTuples(c, [16,15]);
[ 16, 1 ]
gap> UnorderedOrbitalRepresentative(os, [16,2]);
[ 1, 16 ]
gap> c := UnorderedOrbitalCanonizingElement(os, [16,15]);
(1,15)(2,4)(3,12)(5,10)(7,8)(11,13)(17,21)(19,20)
gap> OnSets(c, Set([16,15]));
[ 1, 16 ]
gap> AllOrbitalRepresentatives(os)
[ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 1, 16 ],
  [ 1, 18 ], [ 1, 20 ], [ 16, 1 ], [ 16, 2 ], [ 16, 3 ], [ 16, 16 ], [ 16, 17 ] ]
gap> AllUnorderedOrbitalRepresentatives(os)
[ [ 1, 1 ], [ 1, 2 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 1, 16 ], [ 1, 18 ],
  [ 1, 20 ], [ 16, 16 ], [ 16, 17 ] ]
```

6.1.1 IsOrbitalStructure (for IsObject)

▷ `IsOrbitalStructure(`*arg*`)` (filter)

Returns: `true` or `false`

6.1.2 OrbitalStructure

▷ `OrbitalStructure(`*gens, domain, act*`)` (function)

Returns: An orbital structure

Given generators, a set, and an action function create an orbital structure. An orbital structure contains a list of orbits of the group generated by *gens* on *domain*, a hashmap that maps any element of *domain* to the index of its orbit in the list of orbits. We choose the smallest element of each orbit as representative. For each orbit, the orbital structure also contains the stabilizer of the chosen orbit representative, together with all orbits of that stabilizer on *domain* with chosen representatives.

6.1.3 OS_OrbitRepresentative

▷ `OS_OrbitRepresentative(`*arg*`)` (function)

6.1.4 OS_CanonisingElement

▷ `OS_CanonisingElement(`*arg*`)` (function)

6.1.5 OS_CanonisingElementAndRepresentative

▷ `OS_CanonisingElementAndRepresentative(`*arg*`)` (function)

6.1.6 OS_StabilizerOf

▷ `OS_StabilizerOf(`*arg*`)` (function)

6.1.7 OrbitalRepresentative

▷ `OrbitalRepresentative(`*os, pair*`)` (function)

Returns: pair

Given an orbital structure *os* and a pair *pair* of elements of the domain that *os* is defined on, returns a canonical representative of *pair* in its orbit of ordered pairs.

6.1.8 AllOrbitalRepresentatives

▷ `AllOrbitalRepresentatives(`*os*`)` (function)

Return the set of canonical representatives of orbits of pairs under the action of the orbital structure.

6.1.9 OrbitalCanonizingElement

▷ `OrbitalCanonizingElement(os, pair)` (function)

Returns: a group element

Given an orbital structure `os` and the pair `pair` returns an element *g* of the group that maps `pair` to `OrbitalRepresentative(os, pair)`.

6.1.10 OrbitalCanonizingElementInverse

▷ `OrbitalCanonizingElementInverse(arg)` (function)

6.1.11 OrbitalTransversalIterator

▷ `OrbitalTransversalIterator(os, pair)` (function)

Returns: an iterator

Given an orbital structure `os` and a pair `pair`, returns an iterator that produces an element g for every element `e` in the orbit such that `OnTuples(OrbitalRepresentative(os, pair), g) = e`.

6.1.12 UnorderedOrbitalRepresentative

▷ `UnorderedOrbitalRepresentative(os, pair)` (function)

Returns: pair

Given an orbital structure `os` and a pair `pair` of elements of the domain that `os` is defined on, returns a canonical representative of `pair` in its orbit of sets.

6.1.13 AllUnorderedOrbitalRepresentatives

▷ `AllUnorderedOrbitalRepresentatives(os)` (function)

Return the set of canonical representatives of orbits of sets of size two under the action of the orbital structure.

6.1.14 UnorderedOrbitalTransversalIterator

▷ `UnorderedOrbitalTransversalIterator(os, pair)` (function)

Returns: an iterator

Given an orbital structure `os` and a pair `pair`, returns an iterator that produces an element g for every element `e` in the orbit such that `OnSets(UnorderedOrbitalRepresentative(os, pair), g) = e`.

6.1.15 UnorderedOrbitalCanonizingElement

▷ `UnorderedOrbitalCanonizingElement(os, pair)` (function)

Returns: a group element

Given an orbital structure `os` and the pair `pair` returns an element *g* of the group that maps `pair` to `UnorderedOrbitalRepresentative(os, pair)`.

6.1.16 UnorderedOrbitalCanonizingElementInverse

▷ `UnorderedOrbitalCanonizingElementInverse(`*arg*`)` (function)

Chapter 7

Signed Permutations

We provide *signed permutations*, that is permutations that can additionally change the sign of their result.

Assume $n \in \mathbb{N}$, then a signed permutation on n points is a permutation π on $\{1 \ldots n\}$ together with signs $sgn : \{1..n\} \to \{-1, 1\}$. A signed permutation on n points acts on the set $\{-n \ldots 1, 1 \ldots n\}$ by $\omega^{(\pi,sgn)} = sgn(\omega) \cdot sgn(|\omega|^{\pi}) \cdot (|\omega|^{\pi})$.

We provide two representations of signed permutations, one as a list of images `IsSignedPermListRep` (7.2.8) and one formed as pair of a permutation and a sign map `IsSignedPermRep` (7.2.7). Our benchmarks indicate that a list of images is the better representation, and hence this is the default.

To get started with signed permutations consider the following example

Example
```
gap> s := SignedPerm([2,-1]);
<signed permutation in list rep>
gap> 1 ^ s;
2
gap> 2 ^ s;
-1
gap> OnPoints(2, s);
-1
```

One can form groups out of signed permutations

Example
```
gap> r := SignedPerm([-1,3,-2,4]);; t := SignedPerm([3,1,4,2]);;
gap> G := Group(r,t);
<group with 2 generators>
gap> Size(G);
32
gap> Orbit(G, 1, OnPoints);
[ 1, -1, 3, -3, -2, 4, 2, -4 ]
gap> Stabilizer(G, 1, OnPoints);
<group of size 4 with 9 generators>
```

Note that currently the package does not make an effort to exploit the special structure of signed permutation groups as permutation groups.

7.1 Different Representations

To create signed permutations in the different representations, we provide a constructor.

Example
```
gap> r := NewSignedPerm(IsSignedPermRep, [-1,3,-2,4]);;
gap> t := SignedPerm(IsSignedPermRep, [3,1,4,2]);;
gap> G := Group(r,t);
<group with 2 generators>
gap> Size(G);
32
gap> r := NewSignedPerm(IsSignedPermListRep, [-1,3,-2,4]);;
gap> t := SignedPerm(IsSignedPermListRep, [3,1,4,2]);;
gap> G := Group(r,t);
<group with 2 generators>
gap> Size(G);
32
```

7.2 Low-Level Descriptions

7.2.1 IsSignedPerm (for IsAssociativeElement andIsExtLElement andIsExtRElement andIsMultiplicativeElement andIsMultiplicativeElementWithOne andIsMultiplicativeElementWithInverse andIsFiniteOrderElement)

▷ `IsSignedPerm(`*arg*`)` (filter)

Returns: `true` or `false`

Category of signed permutations

7.2.2 ListSignedPerm (for IsSignedPerm)

▷ `ListSignedPerm(`*perm*`)` (operation)

Convert a signed permutation into a list of images, equivalent to List([1..LargestMovedPoint(s)], x -> x^s);

7.2.3 ListSignedPerm (for IsSignedPerm, IsPosInt)

▷ `ListSignedPerm(`*arg1*`, `*arg2*`)` (operation)

Convert a signed permutation to a list of images of length `len`. Arguments perm, len

7.2.4 SignedPerm

▷ `SignedPerm(`*arg*`)` (function)

Given a list of signed images create a signed permutation object in `IsSignedPermListRep` (7.2.8).

7.2.5 NewSignedPerm (for IsSignedPerm, IsList)

▷ `NewSignedPerm(`*arg1, arg2*`)` (constructor)

7.2.6 NewSignedPerm (for IsSignedPerm, IsPerm, IsList)

▷ `NewSignedPerm(`*arg1, arg2, arg3*`)` (constructor)

7.2.7 IsSignedPermRep (for IsSignedPerm and IsPositionalObjectRep)

▷ `IsSignedPermRep(`*arg*`)` (filter)

Returns: `true` or `false`

Representation of signed permutations as a permutation and a vector of signs.

7.2.8 IsSignedPermListRep (for IsSignedPerm and IsPositionalObjectRep)

▷ `IsSignedPermListRep(`*arg*`)` (filter)

Returns: `true` or `false`

Representation of signed permutations as a list of signed images

7.2.9 OnPosPoints

▷ `OnPosPoints(`*arg*`)` (function)

Only act as a permutation on $\{1 \ldots n\}$

7.2.10 LargestMovedPoint (for IsSignedPerm)

▷ `LargestMovedPoint(`*arg*`)` (attribute)

The largest point that is moved by the signed permutation, where moving includes changing the sign.

7.2.11 RandomSignedPermList

▷ `RandomSignedPermList(`*arg*`)` (function)

Create a random list of images that can be used to create a signed permutation.

7.2.12 RandomSignedPerm

▷ `RandomSignedPerm(`*arg*`)` (function)

Create a random signed permutation

Index

AllOrbitalRepresentatives, 17
AllUnorderedOrbitalRepresentatives, 18

InfoMajorana, 6
IsOrbitalStructure
  for IsObject, 17
IsSignedPerm
  for IsAssociativeElement andIsExtLElement andIsExtRElement andIsMultiplicativeElement andIsMultiplicativeElementWithOne andIsMultiplicativeElementWithInverse andIsFiniteOrderElement, 21
IsSignedPermListRep
  for IsSignedPerm and IsPositionalObjectRep, 22
IsSignedPermRep
  for IsSignedPerm and IsPositionalObjectRep, 22

LargestMovedPoint
  for IsSignedPerm, 22
ListSignedPerm
  for IsSignedPerm, 21
  for IsSignedPerm, IsPosInt, 21

MajoranaAlgebraTest, 14
MajoranaRepresentation, 9
MAJORANA_AdjointAction, 12
MAJORANA_AlgebraProduct, 11
MAJORANA_Basis, 12
MAJORANA_Dimension, 12
MAJORANA_Eigenvectors, 12
MAJORANA_InnerProduct, 11
MAJORANA_IsComplete, 12
MAJORANA_IsJordanAlgebra, 13
MAJORANA_IsSixTranspositionGroup, 7
MAJORANA_RemoveDuplicateShapes, 7
MAJORANA_Subalgebra, 12
MAJORANA_TestAxiomM2, 15
MAJORANA_TestFrobeniusForm, 14
MAJORANA_TestInnerProduct, 14
MAJORANA_TestPrimitivity, 15

NClosedMajoranaRepresentation, 10
NewSignedPerm
  for IsSignedPerm, IsList, 22
  for IsSignedPerm, IsPerm, IsList, 22

OnPosPoints, 22
OrbitalCanonizingElement, 18
OrbitalCanonizingElementInverse, 18
OrbitalRepresentative, 17
OrbitalStructure, 17
OrbitalTransversalIterator, 18
OS_CanonisingElement, 17
OS_CanonisingElementAndRepresentative, 17
OS_OrbitRepresentative, 17
OS_StabilizerOf, 17

RandomSignedPerm, 22
RandomSignedPermList, 22

ShapesOfMajoranaRepresentation, 7
ShapesOfMajoranaRepresentationAxiomM8, 7
SignedPerm, 21

UnorderedOrbitalCanonizingElement, 18
UnorderedOrbitalCanonizingElementInverse, 19
UnorderedOrbitalRepresentative, 18
UnorderedOrbitalTransversalIterator, 18