The ALCO Package

Version 0.1

Benjamin Nasmith

Benjamin Nasmith Email: bnasmith@proton.me

Copyright

Contents

Chapter 1

Introduction

The ALCO package provides tools for algebraic combinatorics, most of which was written for GAP during the author's Ph.D. program [Nas23]. This package provides implementations in GAP of octonion algebras, Jordan algebras, and certain important integer subrings of those algebras. It also provides tools to compute the parameters of t-designs in spherical and projective spaces (modeled as manifolds of primitive idempotent elements in a simple Euclidean Jordan algebra). Finally, this package provides tools to explore octonion lattice constructions, including octonion Leech lattices. The following examples illustrate how one might use this package to explore these structures.

The ALCO package allows users to construct the octonion arithmetic (integer ring). In the example below, we construct the octonion arithmetic and verify that the basis vectors define an E8 lattice relative to the inner product shown:

Example

```
gap> LoadPackage("alco");
true
gap> A := OctonionArithmetic(Integers);
<algebra of dimension 8 over Integers>
gap> g := List(Basis(A), x -> List(Basis(A), y -> Norm(x+y) - Norm(x) - Norm(y)));;
gap> Display(g);
[ [   2,   0,  -1,   0,   0,   0,   0,   0 ],
  [   0,   2,   0,  -1,   0,   0,   0,   0 ],
  [  -1,   0,   2,  -1,   0,   0,   0,   0 ],
  [   0,  -1,  -1,   2,  -1,   0,   0,   0 ],
  [   0,   0,   0,  -1,   2,  -1,   0,   0 ],
  [   0,   0,   0,   0,  -1,   2,  -1,   0 ],
  [   0,   0,   0,   0,   0,  -1,   2,  -1 ],
  [   0,   0,   0,   0,   0,   0,  -1,   2 ] ]
gap> IsGossetLatticeGramMatrix(g);
true
```

We can also construct simple Euclidean Jordan algebras, including the Albert algebra:

Example

```
gap> J := AlbertAlgebra(Rationals);
<algebra of dimension 27 over Rationals>
gap> SemiSimpleType(Derivations(Basis(J)));
"F4"
gap> AsList(Basis(J));
[ i1, i2, i3, i4, i5, i6, i7, i8, j1, j2, j3, j4, j5, j6, j7, j8, k1, k2, k3, k4, k5, k6,
  k7, k8, ei, ej, ek ]
```

```
gap> List(Basis(J), x -> Trace(x));
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 ]
gap> List(Basis(J), x -> Norm(x));
[ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1/2, 1/2, 1/2 ]
gap> List(Basis(J), x -> Determinant(x));
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> One(J);
ei+ej+ek
gap> Determinant(One(J));
1
```

The ALCO package also provides tools to construct octonion lattices, including octonion Leech lattices:

Example

```
gap> short := Set(ShortestVectors(GramMatrix(A),4).vectors, y -> LinearCombination(Basis(A), y));;
gap> filt := Filtered(short, x -> x^2 + x + 2*One(x) = Zero(x));;
gap> Length(filt);
576
gap> s := Random(filt);
a3+a4+a5+a7+a8
gap> gens := List(Basis(A), x -> x*[[s,s,0],[0,s,s],ComplexConjugate([s,s,s])]);;
gap> gens := Concatenation(gens);;
gap> L := OctonionLatticeByGenerators(gens, One(A)*IdentityMat(3)/2);
<free left module over Integers, with 24 generators>
gap> IsLeechLatticeGramMatrix(GramMatrix(L));
true
```

Chapter 2

Octonions

GAP contains limited built-in functionality for constructing and manipulating octonions. The built-in `OctaveAlgebra` function constructs the split-octonion algebra over some field. The ALCO package provides constructions of *non-split* octonion algebras in various bases.

2.1 Octonion Algebras

2.1.1 Octonion Filters

▷ `IsOctonion` (filter)
▷ `IsOctonionArithmeticElement` (filter)
▷ `IsOctonionCollection` (filter)
▷ `IsOctonionAlgebra` (filter)

These filters determine whether an element is an octonion, an octonion arithmetic element, and octonion collection, or an octonion algebra.

2.1.2 OctonionAlgebra

▷ `OctonionAlgebra(F)` (function)

Returns an octonion algebra over field F in a standard orthonormal basis $\{e_i, i = 1,...,8\}$ such that $1 = e_8$ is the identity element and $e_i = e_{i+1}e_{i+3} = -e_{i+3}e_{i+1}$ for $i = 1,...,7$, with indices evaluated modulo 7.

Example
```
gap> O := OctonionAlgebra(Rationals); e := Basis(O);;
<algebra of dimension 8 over Rationals>
gap> LeftActingDomain(O);
Rationals
gap> AsList(e);
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
gap> One(O);
e8
gap> e[1]*e[2];
e4
gap> e[2]*e[1];
```

```
(-1)*e4
gap> Derivations(Basis(O)); SemiSimpleType(last);
<Lie algebra of dimension 14 over Rationals>
"G2"
```

2.1.3 OctonionArithmetic

▷ `OctonionArithmetic(R[, option])` (function)

Returns an octonion algebra over R, for R a field or `R = Integers`, in a basis with the geometry of the simple roots of E_8. The $\mathbb{Z}$-linear combinations of the basis vectors form both an octonion arithmetic and the E_8 lattice with respect to inner product `Norm(x+y) - Norm(x) - Norm(y)`. The algebra is constructed using the structure constants defined by the basis vectors `OctonionE8Basis` (2.1.5). If the argument *option* is supplied, then the algebra is instead constructed using the structure constants of the basis vectors `OctonionE8Basis*Inverse(OctonionE8Basis[8])`, which ensures that the final basis vector is the identity element.

Example
```
gap> A := OctonionArithmetic(Integers); a := Basis(A);;
<algebra of dimension 8 over Integers>
gap> LeftActingDomain(A);
Integers
gap> AsList(a);
[ a1, a2, a3, a4, a5, a6, a7, a8 ]
gap> One(A);
(-2)*a1+(-3)*a2+(-4)*a3+(-6)*a4+(-5)*a5+(-4)*a6+(-3)*a7+(-2)*a8
gap> List(a{[1..7]}, x -> x^2 = - One(A));
[ true, true, true, true, true, true, true ]
gap> Order(a[8]);
3
gap> Random(A)*Random(A) in A;
true
gap> List(Basis(A), x -> Order(x));
[ 4, 4, 4, 4, 4, 4, 4, 3 ]
gap> B := OctonionArithmetic(Rationals, true);
<algebra of dimension 8 over Rationals>
gap> List(Basis(B), x -> Order(x));
[ 4, 4, 4, 4, 4, 4, 3, 1 ]
```

2.1.4 Oct

▷ `Oct` (global variable)

The ALCO package loads an instance of `OctonionAlgebra` (2.1.2) over $\mathbb{Q}$ as `Oct`.

Example
```
gap> Oct;
<algebra of dimension 8 over Rationals>
```

2.1.5 OctonionE8Basis

▷ `OctonionE8Basis` (global variable)

The ALCO package also loads a basis for `Oct` (2.1.4) which also serves as the $\mathbb{Z}$-span of an octonion arithmetic. This basis also serves as the basis vectors for the `OctonionArithmetic` (2.1.3) algebra.

Example
```
gap> 2*BasisVectors(OctonionE8Basis);
[ (-1)*e1+e5+e6+e7, (-1)*e1+(-1)*e2+(-1)*e4+(-1)*e7, e2+e3+(-1)*e5+(-1)*e7,
  e1+(-1)*e3+e4+e5, (-1)*e2+e3+(-1)*e5+e7, e2+(-1)*e4+e5+(-1)*e6,
  (-1)*e1+(-1)*e3+e4+(-1)*e5, e1+(-1)*e4+e6+(-1)*e8 ]
```

2.1.6 \mod

▷ `\mod(x, n)` (function)

For x an octonion arithmetic element (namely an element of algebra `OctonionArithmetic(F)`), and n and integer, the expression `x mod n` returns the octonion where each of the coefficients in the arithmetic canonical basis have been evaluated modulo n.

Example
```
gap> A := OctonionArithmetic(Integers);
<algebra of dimension 8 over Integers>
gap> x := Random(A);
(-2)*a2+(3)*a3+a4+(-2)*a6+(-1)*a7+(-1)*a8
gap> x mod 2;
a3+a4+a7+a8
gap> \mod(x,2);
a3+a4+a7+a8
```

2.2 Properties of Octonions

2.2.1 Norm (Octonions)

▷ `Norm(x)` (method)

Returns the norm of octonion x. Recall that an octonion algebra satisfies the composition property $N(xy) = N(x)N(y)$.

Example
```
gap> List(Basis(Oct), x -> Norm(x));
[ 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> x := Random(Oct);; y := Random(Oct);;
gap> Norm(x*y) = Norm(x)*Norm(y);
true
```

2.2.2 Trace (Octonions)

▷ `Trace(x)` (method)

Returns the trace of octonion `x`. Note that `Trace(x)` is an element of `LeftActingDomain(A)`, where `A` is the octonion algebra containing `x`. The trace and real part are related via `RealPart(x) = Trace(x)*One(x)/2`.

Example
```
gap> List(Basis(Oct), x -> Trace(x));
[ 0, 0, 0, 0, 0, 0, 0, 2 ]
gap> List(Basis(Oct), x -> RealPart(x));
[ 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, e8 ]
```

2.2.3 GramMatrix (GramMatrixOctonion)

▷ `GramMatrix(O)` (attribute)

Returns the Gram matrix on the canonical basis of octonion algebra (or arithmetic) `O` on the basis given by inner product $(x,y) = N(x+y) - N(x) - N(y)$. Of note, the Gram matrix of octonion arithmetic `A` shown below is the Gram matrix of an E_8 unimodular lattice.

Example
```
gap> O := OctonionAlgebra(Rationals); Display(GramMatrix(O));
<algebra of dimension 8 over Rationals>
[ [  2,  0,  0,  0,  0,  0,  0,  0 ],
  [  0,  2,  0,  0,  0,  0,  0,  0 ],
  [  0,  0,  2,  0,  0,  0,  0,  0 ],
  [  0,  0,  0,  2,  0,  0,  0,  0 ],
  [  0,  0,  0,  0,  2,  0,  0,  0 ],
  [  0,  0,  0,  0,  0,  2,  0,  0 ],
  [  0,  0,  0,  0,  0,  0,  2,  0 ],
  [  0,  0,  0,  0,  0,  0,  0,  2 ] ]
gap> A := OctonionArithmetic(Rationals); Display(GramMatrix(A));
<algebra of dimension 8 over Rationals>
[ [   2,   0,  -1,   0,   0,   0,   0,   0 ],
  [   0,   2,   0,  -1,   0,   0,   0,   0 ],
  [  -1,   0,   2,  -1,   0,   0,   0,   0 ],
  [   0,  -1,  -1,   2,  -1,   0,   0,   0 ],
  [   0,   0,   0,  -1,   2,  -1,   0,   0 ],
  [   0,   0,   0,   0,  -1,   2,  -1,   0 ],
  [   0,   0,   0,   0,   0,  -1,   2,  -1 ],
  [   0,   0,   0,   0,   0,   0,  -1,   2 ] ]
```

2.2.4 ComplexConjugate (Octonions)

▷ `ComplexConjugate(x)` (method)

Returns the octonion conjugate of octonion `x`, defined by `One(x)*Trace(x) - x`.

2.2.5 RealPart (Octonions)

▷ `RealPart(x)` (method)

Returns the real component of octonion `x`, defined by `(1/2)*One(x)*Trace(x)`.

2.3 Other Octonion Tools

2.3.1 OctonionToRealVector

▷ `OctonionToRealVector(Basis, x)` (function)

Let x be an octonion vector of the form $x = (x_1, x_2, \ldots, x_n)$, for x_i octonion valued coefficients. Let *Basis* be a basis for the octonion algebra containing coefficients x_i. This function returns a vector y of length $8n$ containing the concatenation of the coefficients of x_i in the octonion basis given by *Basis*.

Example
```
gap> x := Basis(Oct)[1];
e1
gap> OctonionToRealVector(Basis(Oct), [x,x]);
[ 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ]
```

2.3.2 RealToOctonionVector

▷ `RealToOctonionVector(Basis, y)` (function)

This function is the is the inverse operation to `OctonionToRealVector` (2.3.1).

Example
```
gap> A := OctonionArithmetic(Integers);
<algebra of dimension 8 over Integers>
gap> a := Basis(A);; AsList(a);
[ a1, a2, a3, a4, a5, a6, a7, a8 ]
gap> x := List([1..3], n -> Random(A));
[ (-1)*a1+(-1)*a2+(-1)*a3+a4+(-1)*a5+a6+(-2)*a7+(-1)*a8, (-2)*a1+(-1)*a3+(2)*a4+(-2)*a5+(
    -1)*a6+(2)*a7+(-3)*a8, (-1)*a1+(3)*a2+(-2)*a4+a5+(-4)*a6+a8 ]
gap> OctonionToRealVector(a, x);
[ -1, -1, -1, 1, -1, 1, -2, -1, -2, 0, -1, 2, -2, -1, 2, -3, -1, 3, 0, -2, 1, -4, 0, 1 ]
gap> RealToOctonionVector(a,last) = last2;
true
```

2.3.3 VectorToIdempotentMatrix

▷ `VectorToIdempotentMatrix(x)` (function)

Let x be a vector satisfying `IsHomogeneousList` and `IsAssociative` with elements that are `IsCyc`, `IsQuaternion`, or `IsOctonion`. Then this function returns the idempotent matrix `TransposedMat([ComplexConjugate(x)])*[x]/Trace(x)`.

Example
```
gap> x := [One(Oct), Basis(Oct)[1], Basis(Oct)[2]];
[ e8, e1, e2 ]
gap> y := VectorToIdempotentMatrix(x);; Display(y);
[ [   (1/3)*e8,    (1/3)*e1,    (1/3)*e2 ],
  [  (-1/3)*e1,    (1/3)*e8,  (-1/3)*e4 ],
  [  (-1/3)*e2,    (1/3)*e4,    (1/3)*e8 ] ]
gap> IsIdempotent(y);
true
```

2.3.4 WeylReflection

▷ `WeylReflection(r, x)` (function)

Let *r* be a vector satisfying `IsHomogeneousList` and `IsAssociative` with elements in `IsCyc`, `IsQuaternion`, or `IsOctonion` and let `IsHomogeneousList(Flat([r,x]))`. Then this function returns the Weyl reflection of vector *x* using the projector defined by `VectorToIdempotentMatrix(r)`. Specifically, the result is `x - 2*x*VectorToIdempotentMatrix(r)`.

Example
```
gap> WeylReflection([1,0,1],[0,1,1]);
[ -1, 1, 0 ]
```

2.4 Quaternion and Icosian Tools

2.4.1 Norm (Quaternions)

▷ `Norm(x)` (method)

Returns the norm of quaternion *x*. Recall that a quaternion algebra satisfies the composition property $N(xy) = N(x)N(y)$.

Example
```
gap> H := QuaternionAlgebra(Rationals); AsList(Basis(H));
<algebra-with-one of dimension 4 over Rationals>
[ e, i, j, k ]
gap> List(Basis(H), x -> Norm(x));
[ 1, 1, 1, 1 ]
gap> x := Random(H);; y := Random(H);; Norm(x*y) = Norm(x)*Norm(y);
true
```

2.4.2 Trace (Quaternions)

▷ `Trace(x)` (method)

Returns the trace of quaternion *x*, such that `RealPart(x) = Trace(x)*One(x)/2`.

Example
```
gap> H := QuaternionAlgebra(Rationals); AsList(Basis(H));
<algebra-with-one of dimension 4 over Rationals>
[ e, i, j, k ]
gap> List(Basis(H), x -> Trace(x));
[ 2, 0, 0, 0 ]
```

2.4.3 ComplexConjugate (Quaternions)

▷ `ComplexConjugate(x)` (method)

Returns the quaternion conjugate of quaternion *x*, defined by `One(x)*Trace(x) - x`.

2.4.4 RealPart (Quaternions)

▷ `RealPart(x)` (method)

Using the built in GAP function, returns the real component of quaternion `x`, equivalent to `(1/2)*One(x)*Trace(x)`. Of note, the value of `ImaginaryPart(x)` as defined in GAP can yield surprising results (due to dividing by the imaginary unit i) and should be used with caution.

Example

```
gap> H := QuaternionAlgebra(Rationals); AsList(Basis(H));
<algebra-with-one of dimension 4 over Rationals>
[ e, i, j, k ]
gap> List(Basis(H), x -> ComplexConjugate(x));
[ e, (-1)*i, (-1)*j, (-1)*k ]
gap> List(Basis(H), x -> RealPart(x));
[ e, 0*e, 0*e, 0*e ]
gap> List(Basis(H), x -> ImaginaryPart(x));
[ 0*e, e, k, (-1)*j ]
```

2.4.5 QuaternionD4basis

▷ `QuaternionD4basis` (global variable)

The ALCO package loads a basis for a quaternion algebra over $\mathbb{Q}$ with the geometry of a D_4 simple root system. The $\mathbb{Z}$-span of this basis is the Hurwitz ring. These basis vectors close under pairwise reflection or multiplication to form a D_4 root system.

Example

```
gap> B := QuaternionD4basis;;
gap> for x in BasisVectors(B) do Display(x); od;
(-1/2)*e+(-1/2)*i+(-1/2)*j+(1/2)*k
(-1/2)*e+(-1/2)*i+(1/2)*j+(-1/2)*k
(-1/2)*e+(1/2)*i+(-1/2)*j+(-1/2)*k
e
```

2.4.6 GoldenModSigma

▷ `GoldenModSigma(x)` (function)

For x in the golden field `NF(5,[ 1, 4 ])`, this function returns the rational coefficient of 1 in the basis `Basis(NF(5,[ 1, 4 ]), [1, (1-Sqrt(5))/2])`.

Example

```
gap> sigma := (1-Sqrt(5))/2;; tau := (1+Sqrt(5))/2;;
gap> x := 5 + 3*sigma;; GoldenModSigma(x);
5
gap> GoldenModSigma(sigma);
0
gap> GoldenModSigma(tau);
1
```

2.4.7 IcosianH4basis

▷ IcosianH4basis (global variable)

The ALCO package loads a basis for a quaternion algebra over NF(5,[1,4]). The $\mathbb{Z}$-span of this basis is the icosian ring. These basis vectors close under pairwise reflection or multiplication to form a H_4 set of vectors.

Example

```
gap> B := IcosianH4basis;;
gap> for x in BasisVectors(B) do Display(x); od;
(-1)*i
(-1/2*E(5)^2-1/2*E(5)^3)*i+(1/2)*j+(-1/2*E(5)-1/2*E(5)^4)*k
(-1)*j
(-1/2*E(5)-1/2*E(5)^4)*e+(1/2)*j+(-1/2*E(5)^2-1/2*E(5)^3)*k
```

Chapter 3

Simple Euclidean Jordan Algebras

The ALCO package provides a number of tools to construct and manipulate simple Euclidean Jordan algebras (described well in [FK94]), including their homotope and isotopes algebras (defined in [McC04, p. 86]). Among other applications, these tools can reproduce many of the examples found in [EG96] and [EG01].

3.1 Filters and Basic Attributes

3.1.1 Jordan Filters

▷ `IsJordanAlgebra` (filter)
▷ `IsJordanAlgebraObj` (filter)

These filters determine whether an element is a Jordan algebra (`IsJordanAlgebra`) or is an element in a Jordan algebra (`IsJordanAlgebraObj`).

A simple Euclidean Jordan algebra V has rank r and degree d. The following methods return the properties of either a Jordan algebra or of the Jordan algebra containing the object.

3.1.2 JordanRank

▷ `JordanRank(x)` (method)

Returns the rank of `x` when `IsJordanAlgebra(x)` or the rank of the Jordan algebra containing `x` when `IsJordanAlgebraObj(x)`.

3.1.3 JordanDegree

▷ `JordanDegree(x)` (method)

Returns the degree of `x` when `IsJordanAlgebra(x)` or the degree of the Jordan algebra containing `x` when `IsJordanAlgebraObj(x)` .

3.1.4 Trace (Jordan Algebras)

▷ `Trace(x)` (method)

Returns the Jordan trace of *x* when `IsJordanAlgebraObj(x)`.

3.1.5 Norm (Jordan Algebras)

▷ `Norm(x)` (method)

Returns the Jordan norm of *x* when `IsJordanAlgebraObj(x)`. The Jordan norm has the value `Trace(x^2)/2`.

3.1.6 GenericMinimalPolynomial

▷ `GenericMinimalPolynomial(x)` (attribute)

Returns the generic minimal polynomial of *x* when `IsJordanAlgebraObj(x)` as defined in [FKK+00, p. 478]. The output is given as a list of polynomial coefficients.

3.1.7 Determinant (Jordan Algebras)

▷ `Determinant(x)` (method)

Returns the Jordan determinant of *x* when `IsJordanAlgebraObj(x)` .

3.2 Jordan Algebra Constructions

3.2.1 SimpleEuclideanJordanAlgebra

▷ `SimpleEuclideanJordanAlgebra(rho, d[, args])` (function)

Returns a simple Euclidean Jordan algebra over $\mathbb{Q}$ in an orthogonal basis.

Example
```
gap> J := SimpleEuclideanJordanAlgebra(3,8);
<algebra of dimension 27 over Rationals>
gap> SemiSimpleType(Derivations(Basis(J)));
"F4"
```

3.2.2 JordanSpinFactor

▷ `JordanSpinFactor(G)` (function)

Returns a Jordan spin factor algebra when *G* is a positive definite Gram matrix.

Example
```
gap> J := JordanSpinFactor(IdentityMat(8));
<algebra of dimension 9 over Rationals>
gap> One(J);
v.1
```

```
gap> [JordanRank(J), JordanDegree(J)];
[ 2, 7 ]
gap> Derivations(Basis(J));
<Lie algebra of dimension 28 over Rationals>
gap> SemiSimpleType(last);
"D4"
gap> x := Random(J);
v.2+(-1)*v.3+(-1)*v.4+(1/2)*v.5+(-2)*v.7+(1/2)*v.8+(-3/2)*v.9
gap> [Trace(x), Determinant(x)];
[ 0, -39/4 ]
gap> p := GenericMinimalPolynomial(x);
[ -39/4, 0, 1 ]
gap> ValuePol(p, x);
0*v.1
```

3.2.3 HermitianSimpleJordanAlgebra

▷ `HermitianSimpleJordanAlgebra(r, B)` (function)

Returns a simple Euclidean Jordan algebra of rank `r` with the basis for the off-diagonal components defined using composition algebra basis `B`.

Example

```
gap> B := OctonionE8Basis;;
gap> J := HermitianSimpleJordanAlgebra(3,B);
<algebra of dimension 27 over Rationals>
gap> [JordanRank(J), JordanDegree(J)];
[ 3, 8 ]
gap> Derivations(Basis(J));
<Lie algebra of dimension 52 over Rationals>
gap> SemiSimpleType(last);
"F4"
```

3.2.4 JordanHomotope

▷ `JordanHomotope(J, u[, s])` (function)

For `J` a Jordan algebra satisfying `IsJordanAlgebra(J)`, and for `u` a vector in `J`, this function returns the corresponding `u`-homotope algebra with the product of x and y defined as $x(uy)+(xu)y-u(xy)$. The `u`-homotope algebra also belongs to the filter `IsJordanAlgebra`. Of note, if `u` is invertible in `J` then the corresponding `u`-homotope algebra is called a `u`-isotope. The optional argument `s` is a string that determines the labels of the canonical basis vectors in the new algebra.

Example

```
gap> J := SimpleEuclideanJordanAlgebra(2,7);
<algebra of dimension 9 over Rationals>
gap> u := Random(J);
(-1/6)*v.1+(3)*v.2+(1/3)*v.3+(-2)*v.4+(-4)*v.6+(-1)*v.8+(-3)*v.9
gap> GenericMinimalPolynomial(u);
[ -469/12, 1/3, 1 ]
gap> H := JordanHomotope(J, u);
<algebra of dimension 9 over Rationals>
```

```
gap> SemiSimpleType(Derivations(Basis(J)));
"D4"
gap> SemiSimpleType(Derivations(Basis(H)));
"D4"
```

3.3 The Albert Algebra

The exceptional simple Euclidean Jordan algebra, or Albert algebra, may be constructed using `SimpleEuclideanJordanAlgebra` (3.2.1) with rank 3 and degree 8. However, that construction uses the upper triangular entries of the Hermitian matrices define the basis vectors (i.e., the `[1][2]`, `[2][3]`, `[1][3]` entries). Much of the literature on the Albert algebra instead uses the `[1][2]`, `[2][3]`, `[3][1]` entries of the Hermitian matrices to define the basis vectors (see for example [Wil09a, pp. 147-148]). The ALCO provides a specific construction of the Albert algebra that uses this convention for defining basis vectors, described below.

3.3.1 AlbertAlgebra

▷ `AlbertAlgebra(F)` (function)

For *F* a field, this function returns an Albert algebra over *F*. For `F = Rationals`, this algebra is isomorphic to `HermitianSimpleJordanAlgebra(3,8,Basis(Oct))` but in a basis that is more convenient for reproducing certain calculations in the literature. Specifically, while `HermitianSimpleJordanAlgebra(3,8,Basis(Oct))` uses the upper-triangular elements of a Hermitian matrix as representative, `AlbertAlgebra(F)` uses the `[1][2]`, `[2][3]`, `[3][1]` entries as representative. These are respectively labeled using `k,i,j`.

Example

```
gap> A := AlbertAlgebra(Rationals);
<algebra of dimension 27 over Rationals>
gap> i := Basis(A){[1..8]}; j := Basis(A){[9..16]}; k := Basis(A){[17..24]}; e := Basis(A){[
25..27]};
[ i1, i2, i3, i4, i5, i6, i7, i8 ]
[ j1, j2, j3, j4, j5, j6, j7, j8 ]
[ k1, k2, k3, k4, k5, k6, k7, k8 ]
[ ei, ej, ek ]
```

3.3.2 Alb

▷ `Alb` (global variable)

The ALCO package includes a loaded instance of the Albert algebra over the rationals.

Example

```
gap> Alb;
<algebra of dimension 27 over Rationals>
```

3.3.3 AlbertVectorToHermitianMatrix

▷ `AlbertVectorToHermitianMatrix(x)` (function)

For an element `x` in `Alb` (see `Alb` (3.3.2)), this function returns the corresponding 3 x 3 Hermitian matrix with octonion entries in `Oct` (see `Oct` (2.1.4)).

3.3.4 HermitianMatrixToAlbertVector

▷ `HermitianMatrixToAlbertVector(x)` (function)

For 3 x 3 Hermitian matrix with elements in `Oct` (see `Oct` (2.1.4)), this function returns the corresponding vector in in `Alb` (see `Alb` (3.3.2)).

Example

```
gap> j := Basis(Alb){[9..16]};
[ j1, j2, j3, j4, j5, j6, j7, j8 ]
gap> mat := AlbertVectorToHermitianMatrix(j[3]);; Display(mat);
[ [     0*e1,      0*e1,   (-1)*e3 ],
  [     0*e1,      0*e1,      0*e1 ],
  [       e3,      0*e1,      0*e1 ] ]
gap> HermitianMatrixToAlbertVector(mat);
j3
```

3.4 The Quadratic Representation

Many important features of simple Euclidean Jordan algebra and their isotopes are related to the quadratic representation. This aspect of Jordan algebras is described well in [McC04, pp.82-86] and [FK94, pp. 32-38]. The following methods allow for the construction of Jordan operators, including the left translation and the quadratic maps.

3.4.1 P

▷ `P(x[, y])` (operation)

For `x` and `y` Jordan algebra elements, satisfying `IsJordanAlgebraObj` this operation applies two methods. In the case of `P(x, y)`, this operation returns `2*x*(x*y) - (x^2)*y`. In the case of `P(x)`, this operation returns the matrix representing the quadratic map in the canonical basis of the Jordan algebra J containing `x`. For `L(x)` the matrix `AdjointMatrix(CanonicalBasis(J), x)`, the operation `P(x)` returns the matrix `2 L(x)^2 - L(x^2)`.

Example

```
gap> J := JordanSpinFactor(IdentityMat(3));
<algebra of dimension 4 over Rationals>
gap> x := Random(J); y := Random(J);
(-1)*v.1+(4/3)*v.2+(-1)*v.3+v.4
(-1)*v.1+(-1/2)*v.2+(2)*v.3+(-1/2)*v.4
gap> P(x,y);
(14/9)*v.1+(-79/18)*v.2+(-11/9)*v.3+(-53/18)*v.4
gap> P(x);; Display(last);
[ [  43/9,  -8/3,     2,    -2 ],
  [  -8/3,   7/9,  -8/3,   8/3 ],
  [     2,  -8/3,  -7/9,    -2 ],
  [    -2,   8/3,    -2,  -7/9 ] ]
gap> LinearCombination(Basis(J), P(x)*ExtRepOfObj(y)) = P(x,y);
```

```
true
gap> ExtRepOfObj(P(x,y)) = P(x)*ExtRepOfObj(y);
true
gap> P(2*x) = 4*P(x);
true
```

3.4.2 JTS

▷ JTS(*x*, *y*, *z*) (operation)

For Jordan algebra elements *x*, *y*, *z* satisfying IsJordanAlgebraObj, JTS(*x*,*y*,*z*) returns the Jordan triple product defined in terms of the Jordan product as *x**(*y**z) + (*x**y)*z - *y**(*x**z). Equivalently, 2*JTS(*x*,*y*,*z*) is equal to P(x+z, y) - P(x, y) - P(z, y).

Example

```
gap> List(Basis(Alb), x -> JTS(i[1],i[1],x));
[ i1, i2, i3, i4, i5, i6, i7, i8, (1/2)*j1, (1/2)*j2, (1/2)*j3, (1/2)*j4, (1/2)*j5,
  (1/2)*j6, (1/2)*j7, (1/2)*j8, (1/2)*k1, (1/2)*k2, (1/2)*k3, (1/2)*k4, (1/2)*k5,
  (1/2)*k6, (1/2)*k7, (1/2)*k8, 0*i1, ej, ek ]
```

3.5 Additional Tools and Properties

3.5.1 HermitianJordanAlgebraBasis

▷ HermitianJordanAlgebraBasis(*r*, *B*) (function)

Returns a set of Hermitian matrices to serve as a basis for the Jordan algebra with or rank *r* and degree given by the cardinality of composition algebra basis *B*. The elements spanning each off-diagonal components are determined by basis *B*.

Example

```
gap> H := QuaternionAlgebra(Rationals);; AsList(Basis(H));
[ e, i, j, k ]
gap> for x in HermitianJordanAlgebraBasis(2, Basis(H)) do Display(x); od;
[ [    e,  0*e ],
  [  0*e,  0*e ] ]
[ [  0*e,  0*e ],
  [  0*e,    e ] ]
[ [  0*e,    e ],
  [    e,  0*e ] ]
[ [     0*e,       i ],
  [  (-1)*i,     0*e ] ]
[ [     0*e,       j ],
  [  (-1)*j,     0*e ] ]
[ [     0*e,       k ],
  [  (-1)*k,     0*e ] ]
```

3.5.2 JordanMatrixBasis

▷ JordanMatrixBasis(*J*) (attribute)

If `IsJordanAlgebra( J )` and J has been constructed using a matrix basis, then the set of matrices corresponding to `CanonicalBasis( J )` can be obtained using `JordanMatrixBasis( J )`.

3.5.3 HermitianMatrixToJordanVector

▷ `HermitianMatrixToJordanVector(mat, J)` (function)

Converts matrix *mat* into an element of Jordan algebra J.

Example
```
gap> H := QuaternionAlgebra(Rationals);; AsList(Basis(H));
[ e, i, j, k ]
gap> J := HermitianSimpleJordanAlgebra(2,Basis(H));
<algebra of dimension 6 over Rationals>
gap> AsList(CanonicalBasis(J));
[ v.1, v.2, v.3, v.4, v.5, v.6 ]
gap> JordanMatrixBasis(J);
[ [ [ e, 0*e ], [ 0*e, 0*e ] ], [ [ 0*e, 0*e ], [ 0*e, e ] ], [ [ 0*e, e ], [ e, 0*e ] ],
  [ [ 0*e, i ], [ (-1)*i, 0*e ] ], [ [ 0*e, j ], [ (-1)*j, 0*e ] ],
  [ [ 0*e, k ], [ (-1)*k, 0*e ] ] ]
gap> List(JordanMatrixBasis(J), x -> HermitianMatrixToJordanVector(x, J));
[ v.1, v.2, v.3, v.4, v.5, v.6 ]
```

3.5.4 JordanAlgebraGramMatrix

▷ `JordanAlgebraGramMatrix(J)` (attribute)

For `IsJordanAlgebra( J )`, returns the Gram matrix on `CanonicalBasis( J )` using inner product `Trace(x*y)`.

Example
```
gap> J := HermitianSimpleJordanAlgebra(2,OctonionE8Basis);
<algebra of dimension 10 over Rationals>
gap> Display(JordanAlgebraGramMatrix(J));
[ [   1,   0,   0,   0,   0,   0,   0,   0,   0,   0 ],
  [   0,   1,   0,   0,   0,   0,   0,   0,   0,   0 ],
  [   0,   0,   2,   0,  -1,   0,   0,   0,   0,   0 ],
  [   0,   0,   0,   2,   0,  -1,   0,   0,   0,   0 ],
  [   0,   0,  -1,   0,   2,  -1,   0,   0,   0,   0 ],
  [   0,   0,   0,  -1,  -1,   2,  -1,   0,   0,   0 ],
  [   0,   0,   0,   0,   0,  -1,   2,  -1,   0,   0 ],
  [   0,   0,   0,   0,   0,   0,  -1,   2,  -1,   0 ],
  [   0,   0,   0,   0,   0,   0,   0,  -1,   2,  -1 ],
  [   0,   0,   0,   0,   0,   0,   0,   0,  -1,   2 ] ]
```

3.5.5 JordanAdjugate

▷ `JordanAdjugate(x)` (function)

For `IsJordanAlgebraObj( x )`, returns the adjugate of `x`, which satisfies `x*JordanAdjugate(x) = One(x)*Determinant(x)`. When `Determinant(x)` is non-zero, `JordanAdjugate(x)` is proportional to `Inverse(x)`.

3.5.6 IsPositiveDefinite

▷ `IsPositiveDefinite(x)` (filter)

For `IsJordanAlgebraObj( x )`, returns `true` when `x` is positive definite and `false` otherwise. This filter uses `GenericMinimalPolynomial` (3.1.6) to determine whether `x` is positive definite.

Chapter 4

Jordan Designs and their Association Schemes

A spherical or projective design is simply a finite subset of a sphere or projective space. The ALCO examines both types of designs as finite subsets of the manifolds of primitive idempotents in a simple Euclidean Jordan algebra. This requires converting the angle $\cos(x)$ between two unit vectors in a sphere into the corresponding angle on a rank 2 Jordan manifold of primitive idempotents [Nas23, p. 72]: $(1+\cos(x))/2$. The tools below allow one to construct a GAP object to represent a design and collect various computed attributes. Constructing a design and its parameters using these tools does not guarantee the existence of such a design, although known examples a possible instances may be studied using these tools.

4.1 Jacobi Polynomials

4.1.1 JacobiPolynomial

▷ JacobiPolynomial(*k*, *a*, *b*) (function)

This function returns the Jacobi polynomial $P(x)$ of degree `k` and type `(a,b)` as defined in [AS72, chap. 22].

Example
```
gap> a := Indeterminate(Rationals, "a");;
gap> b := Indeterminate(Rationals, "b");;
gap> x := Indeterminate(Rationals, "x");;
gap> JacobiPolynomial(0,a,b);
[ 1 ]
gap> JacobiPolynomial(1,a,b);
[ 1/2*a-1/2*b, 1/2*a+1/2*b+1 ]
gap> ValuePol(last,x);
1/2*a*x+1/2*b*x+1/2*a-1/2*b+x
```

4.1.2 Renormalized Jacobi Polynomials

▷ Q_k_epsilon(*k*, *epsilon*, *rank*, *degree*, *x*) (function)

▷ R_k_epsilon(*k*, *epsilon*, *rank*, *degree*, *x*) (function)

These functions return polynomials of degree *k* in the indeterminate *x* corresponding the the renormalized Jacobi polynomials given in [Hog82]. The value of *epsilon* must be 0 or 1. The arguments *rank* and *degree* correspond to the rank and degree of the relevant simple Euclidean Jordan algebra.

4.2 Jordan Designs

4.2.1 Jordan Design Filters

▷ `IsDesign` (filter)
▷ `IsSphericalDesign` (filter)
▷ `IsProjectiveDesign` (filter)

These filters determine whether an object is a Jordan design and whether the design is constructed in a spherical or projective manifold of Jordan primitive idempotents.

4.2.2 DesignByJordanParameters

▷ `DesignByJordanParameters(`*rank*`, `*degree*`)` (function)

This function constructs a Jordan design in the manifold of Jordan primitive idempotents of rank *rank* and degree *degree*.

Example
```
gap> D := DesignByJordanParameters(3,8);
<design with rank 3 and degree 8>
gap> IsDesign(D);
true
gap> IsSphericalDesign(D);
false
gap> IsProjectiveDesign(D);
true
```

4.2.3 Jordan Rank and Degree

▷ `DesignJordanRank(`*D*`)` (attribute)
▷ `DesignJordanDegree(`*D*`)` (attribute)

The rank and degree of an object satisfying filter `IsDesign` are stored as attributes.

Example
```
gap> D := DesignByJordanParameters(3,8);
<design with rank 3 and degree 8>
gap> [DesignJordanRank(D), DesignJordanDegree(D)];
[ 3, 8 ]
```

4.2.4 DesignQPolynomial

▷ `DesignQPolynomial(D)` (attribute)

This attribute stores a function on non-negative integers that returns the coefficients of the renormalized Jacobi polynomial in the manifold of Jordan primitive idempotents corresponding to the design `D`.

Example
```
gap> D := DesignByJordanParameters(3,8);
<design with rank 3 and degree 8>
gap> r := DesignJordanRank(D);; d := DesignJordanDegree(D);;
gap> x := Indeterminate(Rationals, "x");;
gap> DesignQPolynomial(D);
function( k ) ... end
gap> DesignQPolynomial(D)(2);
[ 90, -585, 819 ]
gap> CoefficientsOfUnivariatePolynomial(Q_k_epsilon(2,0,r,d,x));
[ 90, -585, 819 ]
```

4.2.5 DesignConnectionCoefficients

▷ `DesignConnectionCoefficients(D)` (attribute)

This attribute stores the connection coefficients, defined in [Hog92, p. 261], which determine the linear combinations of `DesignQPolynomial(D)` polynomials that yield each power of the indeterminate.

Example
```
gap> D := DesignByJordanParameters(3,8);
<design with rank 3 and degree 8>
gap> DesignConnectionCoefficients(D);
function( s ) ... end
gap> f := DesignConnectionCoefficients(D)(3);
[ [ 1, 0, 0, 0 ], [ 1/3, 1/39, 0, 0 ], [ 5/39, 5/273, 1/819, 0 ],
  [ 5/91, 1/91, 1/728, 1/12376 ] ]
gap> for j in [1..4] do Display(Sum(List([1..4], i -> f[j][i]*DesignQPolynomial(D)(i-1))));
od;
[ 1, 0, 0, 0 ]
[ 0, 1, 0, 0 ]
[ 0, 0, 1, 0 ]
[ 0, 0, 0, 1 ]
```

4.3 Designs with an Angle Set

We can compute a number of properties of a design once the angle set is given.

4.3.1 IsDesignWithAngleSet

▷ `IsDesignWithAngleSet` (filter)

This filter identifies the design as equipped with an angle set.

4.3.2 DesignAddAngleSet

▷ `DesignAddAngleSet(D, A)` (operation)

For a design *D* without an angle set, records the angle set *A* as an attribute `DesignAngleSet`.

Example
```
gap> D := DesignByJordanParameters(4,4);
<design with rank 4 and degree 4>
gap> DesignAddAngleSet(D, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> DesignAngleSet(D);
[ 1/9, 1/3 ]
```

4.3.3 DesignByAngleSet

▷ `DesignByAngleSet(rank, degree, A)` (function)

Constructs a new design with Jordan rank and degree given by *rank* and *degree*, with angle set *A*.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> DesignAngleSet(D);
[ 1/9, 1/3 ]
```

4.3.4 DesignNormalizedAnnihilatorPolynomial

▷ `DesignNormalizedAnnihilatorPolynomial(D)` (attribute)

The normalized annihilator polynomial is defined for an angle set in [BBIT21, p. 185]. This polynomial is stored as an attribute of a design with an angle set.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> DesignNormalizedAnnihilatorPolynomial(D);
[ 1/16, -3/4, 27/16 ]
```

4.3.5 DesignNormalizedIndicatorCoefficients

▷ `DesignNormalizedIndicatorCoefficients(D)` (attribute)

The normalized indicator coefficients are the `DesignQPolynomial(D)`-expansion coefficients of `DesignNormalizedAnnihilatorPolynomial(D)`, discussed for the spherical case in [BBIT21, p. 185]. These coefficients are stored as an attribute of a design with an angle set.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> f := DesignNormalizedIndicatorCoefficients(D);
[ 1/64, 7/960, 9/3520 ]
```

```
gap> Sum(List([1..3], i -> f[i]*DesignQPolynomial(D)(i-1)));
[ 1/16, -3/4, 27/16 ]
gap> DesignNormalizedAnnihilatorPolynomial(D);
[ 1/16, -3/4, 27/16 ]
```

4.3.6 IsDesignWithPositiveIndicatorCoefficients

▷ `IsDesignWithPositiveIndicatorCoefficients` (filter)

This filter determins whether the normalized indicator coefficients of a design are positive, which has significance for certain theorems about designs.

4.3.7 DesignSpecialBound

▷ `DesignSpecialBound(D)` (attribute)

The special bound of a design satisfying `IsDesignWithPositiveIndicatorCoefficients` is the upper limit on the possible cardinality for the given angle set.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> IsDesignWithPositiveIndicatorCoefficients(D);
true
gap> DesignSpecialBound(D);
64
```

4.4 Designs with Cardinality and Angle Set

More properties of a design with an angle set can be computed once the cardinality is also given.

4.4.1 Some Filters

▷ `IsDesignWithCardinality` (filter)
▷ `IsRegularSchemeDesign` (filter)
▷ `IsSpecialBoundDesign` (filter)
▷ `IsAssociationSchemeDesign` (filter)
▷ `IsTightDesign` (filter)

A design with cardinality has a specified number of points. Given a design with v points and angle set A, it is possible to compute the strength t of a design and write s as the size of set A. When a design satisfies $t >= s-1$ it admits a regular scheme. A design at the special bound satisfies $t >= s$. When a design satisfies $t >= 2s-2$ it admits an association scheme. Finally, when a design satisfies $t = 2s-1$ for 0 in A or $t = 2s$ otherwise, it is a tight design (these properties are discussed in [Hog92]).

4.4.2 DesignCardinality

▷ `DesignCardinality(D)` (attribute)

Returns the cardinality of design *D* when that design satisfies `IsDesignWithCardinality`.

4.4.3 DesignAddCardinality

▷ `DesignAddCardinality(D, v)` (function)

This function stores the the specified cardinality *v* as attribute `DesignCardinality` of design *D*. The method requires the *D* satisfies `IsDesignWithAngleSet`.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> DesignSpecialBound(D);
64
gap> DesignAddCardinality(D, 64);
<design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> IsSpecialBoundDesign(D);
true
gap> DesignCardinality(D);
64
```

4.4.4 IsDesignWithStrength

▷ `IsDesignWithStrength` (filter)

This filter identifies designs for which the attribute `DesignStrength` is known.

4.4.5 DesignStrength

▷ `DesignStrength(D)` (attribute)

For a design *D* that satisfies `IsDesignWithPositiveIndicatorCoefficients`, `IsDesignWithCardinality`, and `IsSpecialBoundDesign`, we can compute the strength *t* of the design using the normalized indicator coefficients. This allows us to immediately determine whether the design also satisfies `IsTightDesign` or `IsAssociationSchemeDesign`.

Example
```
gap> D;
<design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> IsAssociationSchemeDesign(D);
false
gap> DesignStrength(D);
2
gap> D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
```

4.4.6 DesignAnnihilatorPolynomial

▷ `DesignAnnihilatorPolynomial(D)` (attribute)

The annihilator polynomial for design *D* is defined by multiplying the `DesignNormalizedAnnihilatorPolynomial(D)` by `DesignCardinality(D)`.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);; DesignAddCardinality(D, 64);; D;
<design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignAnnihilatorPolynomial(D);
[ 4, -48, 108 ]
```

4.4.7 DesignIndicatorCoefficients

▷ `DesignIndicatorCoefficients(D)` (attribute)

The indicator coefficients for design D are defined by multiplying `DesignNormalizedIndicatorCoefficients(D)` by `DesignCardinality(D)`. These indicator coefficients are often useful for directly determining the strength of a design at the special bound.

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);; DesignAddCardinality(D, 64);; D;
<design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignIndicatorCoefficients(D);
[ 1, 7/15, 9/55 ]
```

4.5 Designs Admitting a Regular Scheme

4.5.1 DesignSubdegrees

▷ `DesignSubdegrees(D)` (attribute)

For a design D with cardinality and angle set that satisfies `IsRegularSchemeDesign`, namely $t >= s - 1$, we can compute the regular subdegrees as described in [Hog92, Theorem 3.2].

Example
```
gap> D := DesignByAngleSet(4, 4, [1/3, 1/9]);; DesignAddCardinality(D, 64);; D;
<design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignSubdegrees(D);
[ 27, 36 ]
```

4.6 Designs Admitting an Association Scheme

When a design satisfies $t >= 2s - 2$ then it also admits an association scheme. We can use results given in [Hog92] to determine the parameters of the corresponding association scheme.

4.6.1 DesignBoseMesnerAlgebra

▷ `DesignBoseMesnerAlgebra(D)` (attribute)

For a design that satisfies `IsAssociationSchemeDesign`, we can define the corresponding Bose-Mesner algebra [BBIT21, pp. 53-57]. The canonical basis for this algebra corresponds to the adjacency matrices A_i, with the `s+1`-th basis vector corresponding to A_0.

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> B := DesignBoseMesnerAlgebra(D);
<algebra of dimension 3 over Rationals>
gap> BasisVectors(CanonicalBasis(B));
[ A1, A2, A3 ]
gap> One(B);
A3
```

4.6.2 DesignBoseMesnerIdempotentBasis

▷ `DesignBoseMesnerIdempotentBasis(D)` (attribute)

For a design that satisfies `IsAssociationSchemeDesign`, we can also define the idempotent basis of the corresponding Bose-Mesner algebra [BBIT21, pp. 53-57].

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignBoseMesnerIdempotentBasis(D);
Basis( <algebra of dimension 3 over Rationals>, [ (-5/64)*A1+(3/64)*A2+(27/64)*A3,
  (1/16)*A1+(-1/16)*A2+(9/16)*A3, (1/64)*A1+(1/64)*A2+(1/64)*A3 ] )
gap> List(last, x -> x^2 = x);
[ true, true, true ]
```

4.6.3 DesignIntersectionNumbers

▷ `DesignIntersectionNumbers(D)` (attribute)

The intersection numbers $p^k_{i,j}$ are given by `DesignIntersectionNumbers(D)[k][i][j]`. These intersection numbers serve as the structure constants for the `DesignBoseMesnerAlgebra(D)`. Namely, $A_i A_j = \sum_{k=1}^{s+1} p^k_{i,j} A_k$ (see [BBIT21, pp. 53-57]).

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> A := BasisVectors(Basis(DesignBoseMesnerAlgebra(D)));;
[ A1, A2, A3 ]
gap> p := DesignIntersectionNumbers(D);;
gap> A[1]*A[2] = Sum(List([1..3]), k -> p[k][1][2]*A[k]);
true
```

4.6.4 DesignKreinNumbers

▷ `DesignKreinNumbers(D)` (attribute)

The Krein numbers $q^k_{i,j}$ are given by `DesignKreinNumbers(D)[k][i][j]`. The Krein numbers serve as the structure constants for the `DesignBoseMesnerAlgebra(D)` in the idempotent basis given by `DesignBoseMesnerIdempotentBasis(D)` using the Hadamard matrix product $\circ$. Namely, for idempotent basis E_i and Hadamard product $\circ$, we have $E_i \circ E_j = \sum_{k=1}^{s+1} q_{i,j}^k E_k$ (see [BBIT21, pp. 53-57]).

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> q := DesignKreinNumbers(D);
[ [ [ 10, 16, 1 ], [ 16, 20, 0 ], [ 1, 0, 0 ] ],
  [ [ 12, 15, 0 ], [ 15, 20, 1 ], [ 0, 1, 0 ] ],
  [ [ 27, 0, 0 ], [ 0, 36, 0 ], [ 0, 0, 1 ] ] ]
```

4.6.5 DesignFirstEigenmatrix

▷ `DesignFirstEigenmatrix(D)` (attribute)

As describe in [BBIT21, p. 58], the first eigenmatrix of a Bose-Mesner algebra $P_i(j)$ defines the expansion of the adjacency matrix basis A_i in terms of the idempotent basis E_j as follows: $A_i = \sum_{j=1}^{s+1} P_i(j) E_j$. This attribute returns the component $P_i(j)$ as `DesignFirstEigenmatrix(D)[i][j]`.

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> a := Basis(DesignBoseMesnerAlgebra(D));; e := DesignBoseMesnerIdempotentBasis(D);;
gap> List([1..3], i -> a[i] = Sum([1..3], j -> DesignFirstEigenmatrix(D)[i][j]*e[j]));
[ true, true, true ]
```

4.6.6 DesignSecondEigenmatrix

▷ `DesignSecondEigenmatrix(D)` (attribute)

As describe in [BBIT21, p. 58], the second eigenmatrix of a Bose-Mesner algebra $Q_i(j)$ defines the expansion of the idempotent basis E_i in terms of the adjacency matrix basis A_j as follows: $E_i = (1/v)\sum_{j=1}^{s+1} Q_i(j) A_j$. This attribute returns the component $Q_i(j)$ as `DesignSecondEigenmatrix(D)[i][j]`.

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> a := Basis(DesignBoseMesnerAlgebra(D));; e := DesignBoseMesnerIdempotentBasis(D);;
gap> List([1..3], i -> e[i]*DesignCardinality(D) = Sum([1..3], j -> DesignSecondEigenmatrix(D)[i
[ true, true, true ]
gap> DesignFirstEigenmatrix(D) = Inverse(DesignSecondEigenmatrix(D))*DesignCardinality(D);
true
```

4.6.7 DesignMultiplicities

▷ `DesignMultiplicities(D)` (attribute)

As describe in [BBIT21, pp. 58-59], the design multiplicy m_i is defined as the dimension of the space that idempotent matrix E_i projects onto, or $m_i = trace(E_i)$. We also have $m_i = Q_i(s+1)$.

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignMultiplicities(D);
[ 27, 36, 1 ]
```

4.6.8 DesignValencies

▷ `DesignValencies(D)` (attribute)

As describe in [BBIT21, pp. 55, 59], the design valency k_i is defined as the fixed number of i-associates of any element in the association scheme (also known as the subdegree). We also have $k_i = P_i(s+1)$.

Example
```
gap> D := DesignByAngleSet(4,4,[1/3,1/9]);; DesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignValencies(D);
[ 27, 36, 1 ]
```

4.6.9 DesignReducedAdjacencyMatrices

▷ `DesignReducedAdjacencyMatrices(D)` (attribute)

As defined in [CVL91, p. 201], the reduced adjacency matrices multiply with the same structure constants as the adjacency matrices, which allows for a simpler construction of an algebra isomorphic to the Bose-Mesner algebra. The matrices `DesignReducedAdjacencyMatrices(D)` are used to construct `DesignBoseMesnerAlgebra(D)`.

4.7 Examples

This section provides a number of known examples that can be studied using the ALCO package. The following tight projective t-designs are identified in [Hog82, Examples 1-11].

Example
```
gap> DesignByAngleSet(2, 1, [0,1/2]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 2, degree 1, cardinality 4, and angle set [ 0, 1/2 ]>
gap> DesignByAngleSet(2, 2, [0,1/2]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 2, degree 2, cardinality 6, and angle set [ 0, 1/2 ]>
gap> DesignByAngleSet(2, 4, [0,1/2]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 2, degree 4, cardinality 10, and angle set [ 0, 1/2 ]>
gap> DesignByAngleSet(2, 8, [0,1/2]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 2, degree 8, cardinality 18, and angle set [ 0, 1/2 ]>
gap> DesignByAngleSet(3, 2, [1/4]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 2-design with rank 3, degree 2, cardinality 9, and angle set [ 1/4 ]>
gap> DesignByAngleSet(4, 2, [0,1/3]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 4, degree 2, cardinality 40, and angle set [ 0, 1/3 ]>
gap> DesignByAngleSet(6, 2, [0,1/4]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 6, degree 2, cardinality 126, and angle set [ 0, 1/4 ]>
gap> DesignByAngleSet(8, 2, [1/9]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 2-design with rank 8, degree 2, cardinality 64, and angle set [ 1/9 ]>
```

```
gap> DesignByAngleSet(5, 4, [0,1/4]);; DesignAddCardinality(last, DesignSpecialBound(last));
<Tight 3-design with rank 5, degree 4, cardinality 165, and angle set [ 0, 1/4 ]>
gap> DesignByAngleSet(3, 8, [0,1/4,1/2]);; DesignAddCardinality(last, DesignSpecialBound(las
t));
<Tight 5-design with rank 3, degree 8, cardinality 819, and angle set [ 0, 1/4, 1/2 ]>
gap> DesignByAngleSet(24, 1, [0,1/16,1/4]);; DesignAddCardinality(last, DesignSpecialBound(l
ast));
<Tight 5-design with rank 24, degree 1, cardinality 98280, and angle set [ 0, 1/16, 1/4 ]>
```

An additional icosahedron projective example is identified in [Lyu09].

Example

```
gap> DesignByAngleSet(2, 2, [ 0, (5-Sqrt(5))/10, (5+Sqrt(5))/10 ]);; DesignAddCardinality(last, D
<Tight 5-design with rank 2, degree 2, cardinality 12, and angle set
[ 0, -3/5*E(5)-2/5*E(5)^2-2/5*E(5)^3-3/5*E(5)^4,
  -2/5*E(5)-3/5*E(5)^2-3/5*E(5)^3-2/5*E(5)^4 ]>
```

The Leech lattice short vector design and several other tight spherical designs are given below:

Example

```
gap> DesignByAngleSet(2, 23, [ 0, 1/4, 3/8, 1/2, 5/8, 3/4 ]);; DesignAddCardinality(last, De
signSpecialBound(last));
<Tight 11-design with rank 2, degree 23, cardinality 196560, and angle set
[ 0, 1/4, 3/8, 1/2, 5/8, 3/4 ]>
gap> DesignByAngleSet(2, 5, [ 1/4, 5/8 ]);; DesignAddCardinality(last, DesignSpecialBound(la
st));
<Tight 4-design with rank 2, degree 5, cardinality 27, and angle set [ 1/4, 5/8 ]>
gap> DesignByAngleSet(2, 6, [0,1/3,2/3]);; DesignAddCardinality(last, DesignSpecialBound(las
t));
<Tight 5-design with rank 2, degree 6, cardinality 56, and angle set [ 0, 1/3, 2/3 ]>
gap> DesignByAngleSet(2, 21, [3/8, 7/12]);; DesignAddCardinality(last, DesignSpecialBound(la
st));
<Tight 4-design with rank 2, degree 21, cardinality 275, and angle set [ 3/8, 7/12 ]>
gap> DesignByAngleSet(2, 22, [0,2/5,3/5]);; DesignAddCardinality(last, DesignSpecialBound(la
st));
<Tight 5-design with rank 2, degree 22, cardinality 552, and angle set [ 0, 2/5, 3/5 ]>
gap> DesignByAngleSet(2, 7, [0,1/4,1/2,3/4]);; DesignAddCardinality(last, DesignSpecialBound
(last));
<Tight 7-design with rank 2, degree 7, cardinality 240, and angle set [ 0, 1/4, 1/2, 3/4
 ]>
gap> DesignByAngleSet(2, 22, [0,1/3,1/2,2/3]);; DesignAddCardinality(last, DesignSpecialBoun
d(last));
<Tight 7-design with rank 2, degree 22, cardinality 4600, and angle set
[ 0, 1/3, 1/2, 2/3 ]>
```

Some projective designs meeting the special bound are given in [Hog82, Examples 1-11]:

Example

```
gap> DesignByAngleSet(4, 4, [0,1/4,1/2]);; DesignAddCardinality(last, DesignSpecialBound(las
t));
<3-design with rank 4, degree 4, cardinality 180, and angle set [ 0, 1/4, 1/2 ]>
gap> DesignByAngleSet(3, 2, [0,1/3]);; DesignAddCardinality(last, DesignSpecialBound(last));
<2-design with rank 3, degree 2, cardinality 12, and angle set [ 0, 1/3 ]>
gap> DesignByAngleSet(5, 2, [0,1/4]);; DesignAddCardinality(last, DesignSpecialBound(last));
<2-design with rank 5, degree 2, cardinality 45, and angle set [ 0, 1/4 ]>
```

```
gap> DesignByAngleSet(9, 2, [0,1/9]);; DesignAddCardinality(last, DesignSpecialBound(last));
<2-design with rank 9, degree 2, cardinality 90, and angle set [ 0, 1/9 ]>
gap> DesignByAngleSet(28, 2, [0,1/16]);; DesignAddCardinality(last, DesignSpecialBound(last)
);
<2-design with rank 28, degree 2, cardinality 4060, and angle set [ 0, 1/16 ]>
gap> DesignByAngleSet(4, 4, [0,1/4]);; DesignAddCardinality(last, DesignSpecialBound(last));
<2-design with rank 4, degree 4, cardinality 36, and angle set [ 0, 1/4 ]>
gap> DesignByAngleSet(4, 4, [1/9,1/3]);; DesignAddCardinality(last, DesignSpecialBound(last)
);
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> DesignByAngleSet(16, 1, [0,1/9]);; DesignAddCardinality(last, DesignSpecialBound(last))
;
<2-design with rank 16, degree 1, cardinality 256, and angle set [ 0, 1/9 ]>
gap> DesignByAngleSet(4, 2, [0,1/4,1/2]);; DesignAddCardinality(last, DesignSpecialBound(las
t));
<3-design with rank 4, degree 2, cardinality 60, and angle set [ 0, 1/4, 1/2 ]>
gap> DesignByAngleSet(16, 1, [0,1/16,1/4]);; DesignAddCardinality(last, DesignSpecialBound(l
ast));
<3-design with rank 16, degree 1, cardinality 2160, and angle set [ 0, 1/16, 1/4 ]>
gap> DesignByAngleSet(3, 4, [0,1/4,1/2]);; DesignAddCardinality(last, DesignSpecialBound(las
t));
<3-design with rank 3, degree 4, cardinality 63, and angle set [ 0, 1/4, 1/2 ]>
gap> DesignByAngleSet(3, 4, [0,1/4,1/2,(3+Sqrt(5))/8, (3-Sqrt(5))/8]);; DesignAddCardinality
(last, DesignSpecialBound(last));
<5-design with rank 3, degree 4, cardinality 315, and angle set
[ 0, 1/4, 1/2, -1/2*E(5)-1/4*E(5)^2-1/4*E(5)^3-1/2*E(5)^4,
  -1/4*E(5)-1/2*E(5)^2-1/2*E(5)^3-1/4*E(5)^4 ]>
gap> DesignByAngleSet(12, 2, [0,1/3,1/4,1/12]);; DesignAddCardinality(last, DesignSpecialBou
nd(last));
<5-design with rank 12, degree 2, cardinality 32760, and angle set [ 0, 1/12, 1/4, 1/3 ]>
```

Two important designs related to the H_4 Weyl group are as follows:

Example

```
gap> A := [ 0, 1/4, 1/2, 3/4, (5-Sqrt(5))/8, (5+Sqrt(5))/8, (3-Sqrt(5))/8, (3+Sqrt(5))/8 ];;
gap> D := DesignByAngleSet(2, 3, A);; DesignAddCardinality(D, DesignSpecialBound(D));
<11-design with rank 2, degree 3, cardinality 120, and angle set [ 0, 1/4, 1/2, 3/4, -3/4*E(5)-1/
gap> A := [ 0, 1/4, (3-Sqrt(5))/8, (3+Sqrt(5))/8 ];;
gap> D := DesignByAngleSet(4, 1, A);; DesignAddCardinality(D, DesignSpecialBound(D));
<5-design with rank 4, degree 1, cardinality 60, and angle set
[ 0, 1/4, -1/2*E(5)-1/4*E(5)^2-1/4*E(5)^3-1/2*E(5)^4,
  -1/4*E(5)-1/2*E(5)^2-1/2*E(5)^3-1/4*E(5)^4 ]>
```

Chapter 5

Octonion Lattice Constructions

The ALCO package provides tools to construct lattices from octonion vectors. This permits one to reproduce the results found in [EG96], [Wil09b], [Wil11], [Nas22], [Nas23].

In what follows let L be a free left $\mathbb{Z}$-module that satisfies `IsOctonionLattice`.

5.1 Gram Matrix Filters

5.1.1 IsLeechLatticeGramMatrix

▷ `IsLeechLatticeGramMatrix(G)` (function)

This function returns `true` when `G` is a Gram matrix of a Leech lattice and `false` otherwise. Specifically, this function confirms that the lattice defined by `G` is unimodular with shortest vectors of length at least 4.

5.1.2 IsGossetLatticeGramMatrix

▷ `IsGossetLatticeGramMatrix(G)` (function)

This function returns `true` when `G` is a Gram matrix of a Gosset (E_8) lattice and `false` otherwise. Specifically, this function confirms that the lattice defined by `G` is unimodular with shortest vectors of length at least 2.

5.1.3 IsOctonionLattice

▷ `IsOctonionLattice` (filter)

This is a subcategory of `IsFreeLeftModule` used below to construct octonion lattices with an inner product defined via an octonion gram matrix.

5.1.4 MOGLeechLatticeGeneratorMatrix

▷ `MOGLeechLatticeGeneratorMatrix` (global variable)

▷ `MOGLeechLatticeGramMatrix` (global variable)

The variable `MOGLeechLatticeGeneratorMatrix` stores the 24 x 24 integer matrix that span the Leech lattice [CS13, p. 133]. The variable `MOGLeechLatticeGramMatrix` stores the Gram matrix of the generator matrix rows, with the inner product computed as `x*y/8`.

Example
```
gap> IsLeechLatticeGramMatrix(MOGLeechLatticeGramMatrix);
true
```

5.2 Octonion Lattice Constructions

5.2.1 OctonionLatticeByGenerators

▷ `OctonionLatticeByGenerators(gens[, g])` (function)

For *gens* a list of octonion vectors, so that *gens* satisfies `IsOctonionCollColl`, this function constructs a free left $\mathbb{Z}$-module that satisfies `IsOctonionLattice`. The attribute `LeftActingDomain` is set to `Integers` and the input *gens* is stored as the attribute `GeneratorsOfLeftOperatorAdditiveGroup`. The inner product on the lattice is defined by optional argument *g*, which is an octonion square matrix that defaults to the identity matrix. For *x*,*y* octonion vectors in the lattice, the inner product is computed as `ScalarProduct(L, x, y) = Trace(x*g*ComplexConjugate(y))`.

Example
```
gap> O := OctonionArithmetic(Integers);;
gap> gens := Concatenation(List(Basis(O), x -> x*IdentityMat(3)));;
gap> O3 := OctonionLatticeByGenerators(gens);
<free left module over Integers, with 24 generators>
```

5.3 Octonion Lattice Attributes

5.3.1 UnderlyingOctonionRing

▷ `UnderlyingOctonionRing(L)` (attribute)

This attribute stores the octonion algebra containing the octonion coefficients of the generating vectors, stored as `GeneratorsOfLeftOperatorAdditiveGroup(L)`.

5.3.2 OctonionGramMatrix

▷ `OctonionGramMatrix(L)` (attribute)

This attribute stores the optional argument *g* of `OctonionLatticeByGenerators(gens [,g])`. This attribute stores the octonion matrix used to calculate the inner product on the lattice via `Trace(x*g*ComplexConjugate(y))`. The default value of this attribute is the identity matrix.

5.3.3 Dimension

▷ `Dimension(L)` (attribute)

For L satisfying `IsOctonionLattice` these attributes determine the lattice rank, which is equivalent to the lattice dimension. The value is computed by determining `Rank(GeneratorsAsCoefficients(L))`.

5.3.4 GeneratorsAsCoefficients

▷ `GeneratorsAsCoefficients(L)` (attribute)

This attributes converts the lattice generators, stored as `GeneratorsOfLeftOperatorAdditiveGroup(L)`, into a list of coefficients. For each generating vector x, the coefficient list `OctonionToRealVector(CanonicalBasis(UnderlyingOctonionRing(L)), x)` is added to the list `GeneratorsAsCoefficients(L)`.

5.3.5 LLLReducedBasisCoefficients

▷ `LLLReducedBasisCoefficients(L)` (attribute)

This attribute stores the result of `LLLReducedBasis(L, GeneratorsAsCoefficients(L)).basis`. This provides a set of basis vectors as coefficients for L, since there is no test to ensure that `GeneratorsOfLeftOperatorAdditiveGroup` form a $\mathbb{Z}$-module basis.

5.3.6 GramMatrix (GramMatrixLattice)

▷ `GramMatrix(L)` (attribute)

This attribute stores the Gram matrix of vectors `LLLReducedBasisCoefficients(L)` relative to `ScalarProduct(L, x, y)`.

5.3.7 TotallyIsotropicCode

▷ `TotallyIsotropicCode(L)` (attribute)

This attribute stores the vectorspace over `GF(2)` generated by the vectors `LLLReducedBasisCoefficients(L)` multiplied by `Z(2)` (see [LM82] for more details).

5.3.8 Lattice Basis

▷ `Basis(L)` (attribute)
▷ `CanonicalBasis(L)` (attribute)
▷ `BasisVectors(B)` (attribute)
▷ `IsOctonionLatticeBasis` (filter)

For L satisfying `IsOctonionLattice` the attributes `Basis(L)` and `CanonicalBasis(L)` are equivalent. The corresponding basis satisfies `IsOctonionLatticeBasis(B)` and provides a basis for octonion lattice L as a left free Z-module. In turn, `BasisVectors(B)` are given by `LLLReducedBasisCoefficients(L)`.

5.4 Octonion Lattice Operations

5.4.1 Rank

▷ `Rank(L)` (operation)

For *L* satisfying `IsOctonionLattice` these attributes determine the lattice rank, which is equivalent to the lattice dimension. The value is computed by determining `Rank(GeneratorsAsCoefficients(L))`.

5.4.2 ScalarProduct

▷ `ScalarProduct(L, x, y)` (operation)

For *L* that satisfies `IsOctonionLattice` and *x*, *y* either octonion vectors or coefficient vectors, this operation computes `Trace(x*g*ComplexConjugate(y))` where *g* is equal to `OctonionGramMatrix(L)`.

5.4.3 \in

▷ `\in(x, L)` (operation)

For *x* an octonion vector (satisfies `IsOctonionCollection` and *L* an octonion lattice (satisfies `IsOctonionLattice`), this function evaluates inclusion of *x* in *L*. Note that `\in(x,L)` and `x in L` are equivalent expression.

5.4.4 Sublattice Identification

▷ `IsSublattice(L, M)` (operation)
▷ `IsSubset(L, M)` (operation)

For both *L* and *M* octonion lattices (satisfies `IsOctonionLattice`) these two functions determine whether the elements of *M* are contained in *L*.

5.4.5 \=

▷ `\=(L, M)` (operation)

For both *L* and *M* octonion lattices (satisfies `IsOctonionLattice`) the expression `L = M` return true when `IsSublattice(L, M) and IsSublattice(L, M)`.

5.4.6 Converting Between Real and Octonion Vectors

▷ `RealToOctonionVector(L, x)` (function)
▷ `OctonionToRealVector(L, y)` (function)

Let *L* be an octonion lattice, satisfying `IsOctonionLattice`, and let *B* be a basis for the octonion algebra `UnderlyingOctonionRing(L)`. Let *x* be a real vector with `Length(x) mod 8 = 0` and let

y be an octonion vector of length `Dimension(L)/8`. The function `RealToOctonionVector(B, x)` returns an octonion vector constructed by taking each successive octonion entry as the linear combination in the eight basis vectors of B of the corresponding eight real coefficients. Likewise, the function `OctonionToRealVector(B, y)` is the concatenation of the real coefficients of the octonion entries computed using the basis B. In contrast, `RealToOctonionVector(L, x)` returns the linear combination of the octonion lattice canonical basis vectors defined by `LLLReducedBasisCoefficients(L)` given by the coefficients x. The function `OctonionToRealVector(L, y)` determines the lattice coefficients of octonion vector y in the canonical basis of octonion lattice L.

Example
```
gap> O := OctonionArithmetic(Integers); B := Basis(O);
<algebra of dimension 8 over Integers>
CanonicalBasis( <algebra of dimension 8 over Integers> )
gap> L := OctonionLatticeByGenerators(Concatenation(List(B, x -> x*IdentityMat(3))));
<free left module over Integers, with 24 generators>
Time of last command: 464 ms
gap> List(IdentityMat(24), x -> RealToOctonionVector(L, x)) = List(LLLReducedBasisCoefficien
ts(L), y -> RealToOctonionVector(Basis(O), y));
true
```

Another example illustrates the inverse properties of these functions.

Example
```
gap> OctonionToRealVector(L, RealToOctonionVector(L, [1..24])) = [1..24];
true
gap> OctonionToRealVector(Basis(O),RealToOctonionVector(Basis(O), [1..24])) = [1..24];
true
```

Chapter 6

Closure Tools

The ALCO package provides some basic tools to compute the closure of a set with respect to a binary operation. Some of these tools compute the closure by brute force, while others use random selection of pairs to attempt to find new members not in the set.

6.1 Brute Force Method

6.1.1 Closure

▷ `Closure(gens, op[, option])` (function)

For *gens* satisfying `IsHomogeneousList`, this function computes the closure of *gens* by addition of all elements of the form *op*`(x,y)`, starting with `x` and `y` any elements in *gens*. The function will not terminate until no new members are produced when applying *op* to all ordered pairs of generated elements. The argument *option*, if supplied, ensures that the function treats *op* as a commutative operation.

Caution must be exercised when using this function to prevent attempting to compute the closure of infinite sets.

Example

```
gap> Closure([1,E(7)], \*);
[ 1, E(7)^6, E(7)^5, E(7)^4, E(7)^3, E(7)^2, E(7) ]
gap> QuaternionD4Basis;
Basis( <algebra-with-one of dimension 4 over Rationals>,
[ (-1/2)*e+(-1/2)*i+(-1/2)*j+(1/2)*k, (-1/2)*e+(-1/2)*i+(1/2)*j+(-1/2)*k,
  (-1/2)*e+(1/2)*i+(-1/2)*j+(-1/2)*k, e ] )
gap> Closure(QuaternionD4Basis,\*);
[ (-1)*e, (-1/2)*e+(-1/2)*i+(-1/2)*j+(-1/2)*k, (-1/2)*e+(-1/2)*i+(-1/2)*j+(1/2)*k,
  (-1/2)*e+(-1/2)*i+(1/2)*j+(-1/2)*k, (-1/2)*e+(-1/2)*i+(1/2)*j+(1/2)*k,
  (-1/2)*e+(1/2)*i+(-1/2)*j+(-1/2)*k, (-1/2)*e+(1/2)*i+(-1/2)*j+(1/2)*k,
  (-1/2)*e+(1/2)*i+(1/2)*j+(-1/2)*k, (-1/2)*e+(1/2)*i+(1/2)*j+(1/2)*k, (-1)*i, (-1)*j,
  (-1)*k, k, j, i, (1/2)*e+(-1/2)*i+(-1/2)*j+(-1/2)*k, (1/2)*e+(-1/2)*i+(-1/2)*j+(1/2)*k,
  (1/2)*e+(-1/2)*i+(1/2)*j+(-1/2)*k, (1/2)*e+(-1/2)*i+(1/2)*j+(1/2)*k,
  (1/2)*e+(1/2)*i+(-1/2)*j+(-1/2)*k, (1/2)*e+(1/2)*i+(-1/2)*j+(1/2)*k,
  (1/2)*e+(1/2)*i+(1/2)*j+(-1/2)*k, (1/2)*e+(1/2)*i+(1/2)*j+(1/2)*k, e ]
gap> start := AsList(Basis(OctonionArithmetic(Rationals)));
[ a1, a2, a3, a4, a5, a6, a7, a8 ]
gap> units := Closure(start, \*);;
```

```
gap> Length(units);
240
```

6.2 Random Choice Methods

In many cases the `Closure` (6.1.1) function is impractical to use due to the long computation time of the brute force method. The following functions provide tools to generate more elements of a set under a binary operation without directly proving closure.

6.2.1 RandomClosure

▷ `RandomClosure(gens, op[, N[, print]])` (function)

For *gens* satisfying `IsHomogeneousList`, this function selects a random element *r* in *gens* and computes all elements of the form *op*(*r*,*x*) for *x* either in *gens* or obtained in a previous closure step. Once this process yields a set of elements with equal cardinality *N*+1 times, the function terminates and returns all elements obtained so far as a set. The default value of *N* is 1. The optional *print* argument, if supplied, prints the cardinality of the set of elements obtain so far at each iteration.

Caution must be exercised when using this function to prevent attempting to compute the random closure of an infinite set. Caution is also required in interpreting the output. Even for large values of *N*, the result is not necessarily the full closure of set *gens*. Furthermore, repeated calls to this function may result in different outputs due to the random selection of elements involved throughout.

Example
```
gap> AsList(Basis(Oct));
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
gap> RandomClosure(Basis(Oct), \*, 2, true);
8
12
14
16
16
16
[ (-1)*e1, (-1)*e2, (-1)*e3, (-1)*e4, (-1)*e5, (-1)*e6, (-1)*e7, (-1)*e8, e8, e7, e6, e5, e4, e3,
```

6.2.2 RandomOrbitOnSets

▷ `RandomOrbitOnSets(gens, start, op[, N[, print]])` (function)

This function proceeds in a manner very similar to `RandomClosure` (6.2.1) with the following differences. This function instead selects a random element random element *r* in *gens* and then for every *x* in *start*, or the set of previously generated elements, computes *op*(*r*,*x*). At each step the cardinality of the union of *start* and any previously generated elements is computed. Once the cardinality is fixed for *N + 1* steps, the function returns the set of generated elements.

The same cautions as described in `RandomClosure` (6.2.1) apply. Note that while *start* is always a subset of the output, `Difference(`*gens*`,`*start*`)` is not a subset of the output.

Example
```
gap> gens := Basis(Oct){[1,2,3]};
[ e1, e2, e3 ]
```

```
gap> start := Basis(Oct){[8]};
[ e8 ]
gap> RandomOrbitOnSets(gens, start, {x,y} -> x*y, 3, true);
1
2
4
6
7
8
8
16
16
16
[ (-1)*e1, (-1)*e2, (-1)*e3, (-1)*e4, (-1)*e5, (-1)*e6, (-1)*e7, (-1)*e8, e8, e7, e6, e5, e4, e3,
```

References

[AS72] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of Mathematical Functions*. Dover, New York, 1972. 22

[BBIT21] Eiichi Bannai, Etsuko Bannai, Tatsuro Ito, and Rie Tanaka. *Algebraic Combinatorics*. Number 5 in De Gruyter, Series in Discrete Mathematics and Applications. Walter de Gruyter GmbH, Berlin/Boston, 2021. OCLC: on1243061900. 25, 28, 29, 30, 31

[CS13] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013. 35

[CVL91] Peter Jephson Cameron and Jacobus Hendricus Van Lint. *Designs, graphs, codes and their links*, volume 3. Cambridge University Press, Cambridge, 1991. 31

[EG96] Noam D. Elkies and Benedict H. Gross. The exceptional cone and the Leech lattice. *International Mathematics Research Notices*, 1996(14):665–698, 1996. Publisher: Citeseer. 14, 34

[EG01] Noam D Elkies and Benedict H Gross. Cubic Rings and the Exceptional Jordan Algebra. *Duke Mathematical Journal*, 109(2):383–410, 2001. 14

[FK94] Jacques Faraut and Adam Koranyi. *Analysis on Symmetric Cones*. Clarendon Press, 1994. 14, 18

[FKK+00] Jacques Faraut, Soji Kaneyuki, Adam Koranyi, Qi-keng Lu, and Guy Roos. *Analysis and Geometry on Complex Homogeneous Domains*. Number 185 in Progress in mathematics. BirkhÃ¤user, Boston, 2000. 15

[Hog82] Stuart G. Hoggar. t-Designs in projective spaces. *European Journal of Combinatorics*, 3(3):233–254, 1982. 23, 31, 32

[Hog92] Stuart G. Hoggar. t-Designs with general angle set. *European Journal of Combinatorics*, 13(4):257–271, 1992. 24, 26, 28

[LM82] James Lepowsky and Arne Meurman. An E8-approach to the Leech lattice and the Conway group. *Journal of Algebra*, 77(2):484–504, August 1982. 36

[Lyu09] Yu. I. Lyubich. On tight projective designs. *Designs, Codes and Cryptography*, 51(1):21–31, April 2009. 32

[McC04] Kevin McCrimmon. *A Taste of Jordan Algebras*. Universitext. Springer-Verlag, New York, 2004. 14, 18

[Nas22] Benjamin Nasmith. Octonions and the two strictly projective tight 5-designs. *Algebraic Combinatorics*, 5(3):401–411, 2022. 34

[Nas23] Benjamin Nasmith. *Tight Projective 5-Designs and Exceptional Structures*. PhD Thesis, Royal Military College of Canada, Kingston ON, 2023. Accepted: 2023-07-27T12:24:01Z. 4, 22, 34

[Wil09a] Robert A. Wilson. *The Finite Simple Groups*. Graduate Texts in Mathematics. Springer-Verlag, London, 2009. 17

[Wil09b] Robert A. Wilson. Octonions and the Leech lattice. *Journal of Algebra*, 322(6):2186–2190, September 2009. 34

[Wil11] Robert A. Wilson. Conway's group and octonions. *Journal of Group Theory*, 14(1):1–8, January 2011. 34

Index