ALCO

Tools for algebraic combinatorics

1.1.2

5 September 2025

Benjamin Nasmith

Benjamin Nasmith
Email: bnasmith@proton.me
Homepage: https://github.com/BNasmith/

Abstract

ALCO 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.

Copyright

Acknowledgements

This documentation was prepared using the GAPDoc package.

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 work with the octavian integer ring (also known as the octonion arithmetic), which is described carefully in [CS03, chaps. 9-11]. In the example below, we verify that the octavian integers define an E_8 (Gossett) lattice relative to the standard octonion inner product:

Example

```
gap> O := OctavianIntegers;
OctavianIntegers
gap> g := List(Basis(O), x -> List(Basis(O), 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
```

The ALCO package also provides tools to construct octonion lattices, including octonion Leech lattices (see for example [Wil09b]). In the following example we compute the shortest vectors in the `OctavianIntegers` lattice and select one that is a root of polynomial x^2+x+2. We use this root `s` to define a set `gens` of octonion triples to serve as generators for the lattice. Finally, we construct the lattice `L` and confirm that it is a Leech lattice.

Example

```
gap> short := Set(ShortestVectors(g,4).vectors, y ->
> LinearCombination(Basis(OctavianIntegers), y));;
```

```
gap> s := First(short, x -> x^2 + x + 2*One(x) = Zero(x));
(-1)*e1+(-1/2)*e2+(-1/2)*e3+(-1/2)*e4+(-1/2)*e8
gap> gens := List(Basis(OctavianIntegers), x ->
> x*[[s,s,0],[0,s,s],ComplexConjugate([s,s,s])]);;
gap> gens := Concatenation(gens);;
gap> L := OctonionLatticeByGenerators(gens, One(0)*IdentityMat(3)/2);
<free left module over Integers, with 24 generators>
gap> IsLeechLatticeGramMatrix(GramMatrix(L));
true
```

We can also construct and study simple Euclidean Jordan algebras (described well in [FK94]), including the Albert algebra:

Example

```
gap> J := AlbertAlgebra(Rationals);
<algebra-with-one of dimension 27 over Rationals>
gap> SemiSimpleType(Derivations(Basis(J)));
"F4"
gap> i := Basis(J){[1..8]};
[ i1, i2, i3, i4, i5, i6, i7, i8 ]
gap> j := Basis(J){[9..16]};
[ j1, j2, j3, j4, j5, j6, j7, j8 ]
gap> k := Basis(J){[17..24]};
[ k1, k2, k3, k4, k5, k6, k7, k8 ]
gap> e := Basis(J){[25..27]};
[ ei, ej, ek ]
gap> ForAll(e, IsIdempotent);
true
gap> Set(i, x -> x^2);
[ ej+ek ]
gap> Set(j, x -> x^2);
[ ei+ek ]
gap> One(J);
ei+ej+ek
gap> Determinant(One(J));
1
gap> Trace(One(J));
3
```

Chapter 2

Octonions

Let C be a vector space over field k equipped with a non-degenerate quadratic form $N : C \to k$. If C is also an algebra with a product that satifies the composition rule $N(xy) = N(x)N(y)$ for all x, y in C then we call C a *composition algebra*. Quaternions and octonions are examples of composition algebras.

As described in [SV00, Theorm 1.6.2], a composition algebra has dimension 1, 2, 4, or 8. Compositions algebras of dimension 1 or 2 are commutative and associative. A *quaternion algebra* is a composition algebra of dimension 4. Quaternion algebras are associative but noncommutative. An *octonion algebra* is a composition algebra of dimension 8. Octonion algebras are both noncommutative and nonassociative but still have many interesting properties.

The non-degenerate quadratic form N of a composition algebra is called the *norm* of that algebra. In fact, a composition algebra is determined up to isomorphism by its norm so the task of classifying composition algebras is equivalent to the task of classifying the possible norms in that vector space [SV00, chap. 1, sec. 7]. A norm is either isotropic or anisotropic according to whether or not there exists a non-zero element x such that $N(x) = 0$. A composition algebra with a isotropic norm is called a *split composition algebra*. A composition algebra with an anisotropic norm is called a *division composition algebra* since each non-zero element has an inverse. An important theorem [SV00, Theorem 1.8.1] shows that for each field k there exists up to isomorphism one split composition algebra of dimension 2, 4, and 8. The built-in `OctaveAlgebra(F)` function in GAP constructs the split-octonion algebra over the field given as the argument.

As described in [SV00, chap. 1, sec. 10], there are precisely one division composition algebra of dimension 4 and one of dimension 8 over the real number field (likewise over the rationals). The ALCO package provides constructions of *non-split* octonion algebras, provided that the algebra is constructed over a suitable field (e.g., octonions over any finite field are split).

2.1 Octonion Algebras

2.1.1 Octonion Filters

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

These filters determine whether an element is an octonion, an 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. This corresponds to the basis provided in [Bae02] and [CS03] (except that `e7` corresponds to e_0 in the literature, since the first entry in a GAP list has index 1). Whether or not the algebra constructed is a division algebra or a split-octonion algebra depends on the choice of field `F`. For example, `OctonionAlgebra(Rationals)` is a division composition algebra, but `OctonionAlgebra(GF(3))` is a split composition algebra. Other examples are discussed in [SV00, chap. 1, sec. 10].

Example

```
gap> O := OctonionAlgebra(Rationals);
<algebra-with-one of dimension 8 over Rationals>
gap> LeftActingDomain(O);
Rationals
gap> IsAssociative(O);
false
gap> e := BasisVectors(Basis(O));
[ 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));
<Lie algebra of dimension 14 over Rationals>
gap> SemiSimpleType(last);
"G2"
```

2.1.3 Octavian Integers

▷ `OctavianIntegers` (global variable)

▷ `IsOctavianInt(x)` (operation)

The `OctavianIntegers` are a subring of the octonion algebra with elements that have the geometry of scaled E_8 lattice. This ring is named and studied in [CS03, p. 105]. `CanonicalBasis(OctavianIntegers)` returns `OctonionE8Basis` (2.1.4). We can test whether an octonion is in `OctavianIntegers` using the operation `IsOctavianInt(x)`.

Example

```
gap> a := BasisVectors(Basis(OctavianIntegers));;
gap> for x in a do Display(x); od;
(-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7
(-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7
(1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7
(1/2)*e1+(-1/2)*e3+(1/2)*e4+(1/2)*e5
(-1/2)*e2+(1/2)*e3+(-1/2)*e5+(1/2)*e7
(1/2)*e2+(-1/2)*e4+(1/2)*e5+(-1/2)*e6
(-1/2)*e1+(-1/2)*e3+(1/2)*e4+(-1/2)*e5
(1/2)*e1+(-1/2)*e4+(1/2)*e6+(-1/2)*e8
```

```
gap> ForAll(a, IsOctavianInt);
true
gap> ForAll(a/2, IsOctavianInt);
false
```

2.1.4 OctonionE8Basis

▷ `OctonionE8Basis` (global variable)

The ALCO package also loads a basis for `OctonionAlgebra(`*`Rationals`*`)` which also serves as generators for `OctavianIntegers` (2.1.3). This octonion integer lattice has the geometry of a E_8 (Gossett) lattice relative to the inner product defined by the octonion norm.

Example

```
gap> BasisVectors(OctonionE8Basis) = BasisVectors(Basis(OctavianIntegers));
true
gap> g := List(OctonionE8Basis, x -> List(OctonionE8Basis, y ->
> Norm(x+y) - Norm(x) - Norm(y)));;
gap> IsGossetLatticeGramMatrix(g);
true
```

2.2 Properties of Octonions

2.2.1 Norm (Octonions)

▷ `Norm(`*x*`)` (method)

Returns the norm of octonion *x*. Recall that an octonion algebra with norm N satisfies the composition property $N(xy) = N(x)N(y)$. In the canonical basis for `OctonionAlgebra` (2.1.2), the norm is the sum of the squares of the coefficients.

Example

```
gap> Oct := OctonionAlgebra(Rationals);;
gap> List(Basis(Oct), Norm);
[ 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*. In the canonical basis for `OctonionAlgebra` (2.1.2), the trace is twice the coefficient of the identity element `e8`. The trace and real part are related via `RealPart(`*x*`) = Trace(`*x*`)*One(`*x*`)/2`. Note that `Trace(`*x*`)` is an element of `LeftActingDomain(`*A*`)`, where *A* is the octonion algebra containing *x*.

Example

```
gap> e := BasisVectors(Basis(OctonionAlgebra(Rationals)));
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
```

```
gap> List(e, Trace);
[ 0, 0, 0, 0, 0, 0, 0, 2 ]
gap> List(e, RealPart);
[ 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, e8 ]
```

2.2.3 ComplexConjugate (Octonions)

▷ `ComplexConjugate(x)` (method)

Returns the octonion conjugate of octonion `x`, defined by `One(x)*Trace(x) - x`. In the canonical basis of `OctonionAlgebra` (2.1.2), this method negates the coefficients of $e_1, e_2, \ldots, e_7$ but leaves the identity e_8 fixed.

Example

```
gap> e := BasisVectors(Basis(OctonionAlgebra(Rationals)));
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
gap> List(e, ComplexConjugate);
[ (-1)*e1, (-1)*e2, (-1)*e3, (-1)*e4, (-1)*e5, (-1)*e6, (-1)*e7, e8 ]
```

2.2.4 RealPart (Octonions)

▷ `RealPart(x)` (method)

Returns the real component of octonion `x`, defined by `(1/2)*One(x)*Trace(x)`. Note that `RealPart` returns an octonion in the subspace spanned by the octonion identity element while `Trace` returns an element in the coefficient field of the octonion algebra.

Example

```
gap> e := BasisVectors(Basis(OctonionAlgebra(Rationals)));
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
gap> List(e, RealPart);
[ 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, 0*e1, e8 ]
```

2.3 Other Octonion Tools

2.3.1 Converting Octonion Vectors

▷ `OctonionToRealVector(B, x)` (function)
▷ `RealToOctonionVector(B, y)` (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 `B` be a basis for the octonion algebra containing coefficients x_i. The function `OctonionToRealVector(B, x)` returns a vector `y` of length $8n$ containing the concatenation of the coefficients of x_i in the octonion basis given by `B`. The function `RealToOctonionVector(B, y)` provides the inverse operation.

Example

```
gap> O := UnderlyingLeftModule(OctonionE8Basis);
<algebra-with-one of dimension 8 over Rationals>
gap> BasisVectors(CanonicalBasis(O));
[ e1, e2, e3, e4, e5, e6, e7, e8 ]
```

```
gap> x := 2*OctonionE8Basis{[1,2]};
[ (-1)*e1+e5+e6+e7, (-1)*e1+(-1)*e2+(-1)*e4+(-1)*e7 ]
gap> y := OctonionToRealVector(CanonicalBasis(O), x);
[ -1, 0, 0, 0, 1, 1, 1, 0, -1, -1, 0, -1, 0, 0, -1, 0 ]
gap> RealToOctonionVector(CanonicalBasis(O), y);
[ (-1)*e1+e5+e6+e7, (-1)*e1+(-1)*e2+(-1)*e4+(-1)*e7 ]
```

2.3.2 VectorToIdempotentMatrix

▷ `VectorToIdempotentMatrix(x)` (function)

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

Example

```
gap> Oct := OctonionAlgebra(Rationals);;
gap> x := Basis(Oct){[8,1,2]};
[ 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
```

2.3.3 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 Tools

The ALCO package provides some additional tools for studying quaternion algebras in GAP. These tools include methods to compute the quaternion norm and trace, certain important bases, and the subrings generated by those bases. GAP already provides some built-in tools to construct and study the quaternion algebra in the standard Hamiltonian basis. The following example illustrates these tools:

Example

```
gap> H := QuaternionAlgebra(Rationals);
<algebra-with-one of dimension 4 over Rationals>
```

```
gap> IsQuaternion(Random(H));
true
gap> IsAssociative(H);
true
gap> IsCommutative(H);
false
gap> One(H);
e
gap> b := BasisVectors(CanonicalBasis(H));
[ e, i, j, k ]
gap> List(b, ComplexConjugate);
[ e, (-1)*i, (-1)*j, (-1)*k ]
gap> List(b, Inverse);
[ e, (-1)*i, (-1)*j, (-1)*k ]
gap> List(b, RealPart);
[ e, 0*e, 0*e, 0*e ]
gap> List(b, ImaginaryPart);
[ 0*e, e, k, (-1)*j ]
```

Note that the GAP method `ImaginaryPart` acting on objects that satisfy `IsQuaternion` involves dividing by `i`, which yields a different result than the `x - RealPart(x)` that some users may expect. The ALCO package does not define an `ImaginaryPart` method for octonions in order to avoid confusion about the behaviour of that method when applied to quaternions or octonions.

2.4.1 Norm (Quaternions)

▷ `Norm(x)` (method)

Returns the norm of quaternion `x`. Recall that a quaternion algebra with norm N satisfies the composition property $N(xy) = N(x)N(y)$. In the canonical basis of `QuaternionAlgebra`, the norm is the sum of the squares of the coefficients.

Example

```
gap> H := QuaternionAlgebra(Rationals);;
gap> b := BasisVectors(CanonicalBasis(H));
[ e, i, j, k ]
gap> List(b, Norm);
[ 1, 1, 1, 1 ]
gap> x := Random(H);; y := Random(H);;
gap> 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`. In the canonical basis of `QuaternionAlgebra`, the trace is twice the coefficient of the identity element.

Example

```
gap> H := QuaternionAlgebra(Rationals);;
gap> b := BasisVectors(CanonicalBasis(H));
```

```
[ e, i, j, k ]
gap> List(b, Trace);
[ 2, 0, 0, 0 ]
```

2.4.3 Hurwitz Integers

▷ HurwitzIntegers (global variable)
▷ IsHurwitzInt(x) (operation)

The HurwitzIntegers are a subring of the quaternion algebra with elements that have the geometry of scaled D_4 lattice. This ring is named and studied in [CS03, p. 55]. CanonicalBasis(HurwitzIntegers) returns QuaternionD4Basis (2.4.4). We can test whether a quaternion is in HurwitzIntegers using the operation IsHurwitzInt(x).

Example

```
gap> f := BasisVectors(Basis(HurwitzIntegers));;
gap> for x in f 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
gap> ForAll(f, IsHurwitzInt);
true
gap> ForAll(f/2, IsHurwitzInt);
false
```

2.4.4 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 HurwitzIntegers (2.4.3) 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
```

2.5 Icosian Tools

The icosian ring is a subring of the quaternion algebra over the "golden field" $\mathbb{Q}(\sqrt{5})$ This ring is described and studied in [CS13, pp. 207-211] and [Wil09a, p. 220]. The icosian ring has 120 units, or elements with quaternion norm of 1. These units are closed under quaternion multiplication. These icosian units also exhibit a H_4 geometry in the sense that they are closed under Weyl reflection relative to the standard Euclidean inner product defined by the quaternion norm. The Coxeter group generated by these reflections is the group $W(H_4)$. The coefficients of an icosian in the standard quaternion basis

belong to the integer subring of the golden field, meaning that these coefficients have the form $a+b\sigma$, where a,b are rational integers and $\sigma = (1-\sqrt{5})/2$. The Euclidean inner product between any two icosians will be a golden field integer. As described in the references given above, if we define a new inner product between icosians that discards the σ-coefficient b then the icosian ring will exhibit the geometry of an E_8 lattice relative to this new inner product. It is also possible to define a Leech lattice geometry on icosian triples.

The ALCO package provides tools to explore these icosian properties and construct certain important lattices using icosians. Note that the golden field can be constructed in GAP as `NF(5, [ 1, 4 ])` or `Field(Sqrt(5))` and that σ is given by `-EB(5)`.

2.5.1 Icosian Ring

▷ `IcosianRing` (global variable)
▷ `IsIcosian(x)` (operation)

The `IcosianRing` is a subring of the the quaternion algebra over `NF(5,[1,4])` generated by a set of vectors with an H_4 geometry. This ring is described well in [Wil09a, p. 220]. `CanonicalBasis(IcosianRing)` returns `IcosianH4Generators` (2.5.2). We can test whether a quaternion is in `IcosianRing` using the operation `IsIcosian(x)`. Note that a quaternion is an icosian when it is a $\mathbb{Z}$-linear combination of the union of `Basis(IcosianRing)` and `Basis(IcosianRing)*EB(5)`.

Example

```
gap> f := BasisVectors(Basis(IcosianRing));;
gap> for x in f 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
gap> ForAll(f, IsIcosian);
true
gap> ForAll(f/2, IsIcosian);
false
gap> ForAll(f*EB(5), IsIcosian);
true
gap> ForAll(f*Sqrt(5), IsIcosian);
true
gap> ForAll(f*(1-Sqrt(5))/2, IsIcosian);
true
gap> ForAll(f*(1+Sqrt(5))/2, IsIcosian);
true
```

2.5.2 IcosianH4Generators

▷ `IcosianH4Generators` (global variable)

The ALCO package loads this variable as a basis for a quaternion algebra over `NF(5,[1,4])`. Note that a quaternion is an icosian when it is a $\mathbb{Z}$-linear combination of the union of `IcosianH4Generators` and `IcosianH4Generators*EB(5)`. These basis vectors close under pairwise reflection or multiplication to form a H_4 set of vectors.

Example

```
gap> f := BasisVectors(IcosianH4Generators);;
gap> for x in f 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
```

2.5.3 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
```

2.6 Other Integer Rings

Certain addition integer subrings of elements satisfying `IsCyc` are also included in the ALCO package. The rings constructed below are described in [CS03, pp. 16-18].

2.6.1 EisensteinIntegers

▷ `EisensteinIntegers` (global variable)

▷ `IsEisenInt(x)` (operation)

The `EisensteinIntegers` is a subring of the complex numbers generated by `1` and `E(3)`. This subring has the geometry of an A_2 lattice. This ring is described well in [CS03, p. 16]. We can test whether an element in `IsCyc` is an Eisenstein integer using the operation `IsEisenInt(x)`.

Example

```
gap> f := BasisVectors(Basis(EisensteinIntegers));
[ 1, E(3) ]
gap> IsEisenInt(E(4));
false
gap> IsEisenInt(1+E(3)^2);
true
```

2.6.2 KleinianIntegers

▷ `KleinianIntegers` (global variable)

▷ `IsKleinInt(x)` (operation)

The KleinianIntegers is a subring of the complex numbers generated by 1 and (1/2)*(-1+Sqrt(-7)). This ring is described in [CS03, p. 18]. We can test whether an element in IsCyc is an Kleinian integer using the operation IsKleinInt(x).

Example

```
gap> f := BasisVectors(Basis(KleinianIntegers));
[ 1, E(7)+E(7)^2+E(7)^4 ]
gap> IsKleinInt(E(4));
false
gap> IsKleinInt(1+E(7)+E(7)^2+E(7)^4);
true
```

Chapter 3

Simple Euclidean Jordan Algebras

A Jordan algebra is a commutative yet nonassociative algebra with product $\circ$ that satisfies the Jordan identity $x \circ (x^2 \circ y) = x^2 \circ (x \circ y)$. Given an associative algebra, we can define a Jordan algebra on the same elements using the product $x \circ y = (xy + yx)/2$. A Jordan algebra V is *Euclidean* when there exists an inner product (x, y) on V that satisfies $(x \circ y, z) = (y, x \circ z)$ for all x, y, z in V [FK94, p. 42]. Euclidean Jordan algebras are in one-to-one correspondence with structures known as symmetric cones, and any Euclidean Jordan algebra is the direct sum of simple Euclidean Jordan algebras [FK94, chap. 3].

The simple Euclidean Jordan algebras, in turn, are classified by rank and degree into four families and one exception [FK94, chap. 5]. The first family consists of rank 2 algebras with degree any positive integer. The remaining three families consist Jordan algebras with degree 1, 2, or 4 with rank a positive integer greater than 2. The exceptional algebra has rank 3 and degree 8.

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`).

3.1.2 Jordan Rank

▷ `JordanRank(x)` (method)
▷ `Rank(x)` (method)

The rank of a Jordan algeba is the size of a maximal set of mutually orthogonal primitive idempotents in the algebra. The rank and degree are used to classify the simple Euclidean Jordan algebras. This method returns the rank of `x` when `IsJordanAlgebra(x)` or the rank of the Jordan alge-

bra containing `x` (computed as `FamilyObj(x)!.fullSCAlgebra`) when `IsJordanAlgebraObj(x)`. The method `Rank(x)` returns `JordanRank(x)` when `x` satisfies either `IsJordanAlgebra` or `IsJordanAlgebraObj`.

3.1.3 Jordan Degree

▷ `JordanDegree(x)` (method)
▷ `Degree(x)` (method)

The degree of a Jordan algebra is the dimension of the off-diagonal entries in a Pierce decomposition of the Jordan algebra. For example, a Jordan algebra of quaternion hermitian matrices has degree 4. This method returns the degree of `x` when `IsJordanAlgebra(x)` or the degree of the Jordan algebra containing `x` (computed as `FamilyObj(x)!.fullSCAlgebra`) when `IsJordanAlgebraObj(x)`. The method `Degree(x)` returns `JordanDegree(x)` when `x` satisfies either `IsJordanAlgebra` or `IsJordanAlgebraObj`.

Each vector in a simple Euclidean Jordan algebra can be written as a $\mathbb{R}$-linear combination of mutually orthogonal primitive idempotents. This is called the *spectral decomposition* of a Jordan algebra element. The coefficients in the decomposition are the *eigenvalues* of the element. The Jordan trace and determinant, described below, are respectively the sum and product of these eigenvalues with multiplicities included [FK94, p. 44].

3.1.4 Trace (Jordan Algebras)

▷ `Trace(x)` (method)

Returns the Jordan trace of `x` when `IsJordanAlgebraObj(x)`. The trace of a Jordan algebra element is the sum of the eigenvalues of that element (with multiplicies included).

3.1.5 Determinant (Jordan Algebras)

▷ `Determinant(x)` (method)

Returns the Jordan determinant of `x` when `IsJordanAlgebraObj(x)`. The determinant of a Jordan algebra element is the product of the eigenvalues of that element (with multiplicies included).

3.1.6 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`.

3.1.7 GenericMinimalPolynomial

▷ `GenericMinimalPolynomial(x)` (attribute)

Returns the generic minimal polynomial of *x* when `IsJordanAlgebraObj(x)` as defined in [FKK+00, p. 478] (see also [FK94, pp. 27-31]). The output is given as a list of polynomial coefficients. Note that the generic minimal polynomial is a monic polynomial of degree equal to the rank of the Jordan algebra. The trace and determinant of a Jordan algebra element are, to within a sign, given by the coefficients of the second highest degree term and the constant term.

Example

```
gap> J := AlbertAlgebra(Rationals);;
gap> x := Sum(Basis(J){[4,5,6,25,26,27]});
i4+i5+i6+ei+ej+ek
gap> [JordanRank(J), JordanDegree(J)];
[ 3, 8 ]
gap> [JordanRank(x), JordanDegree(x)];
[ 3, 8 ]
gap> p := GenericMinimalPolynomial(x);
[ 2, 0, -3, 1 ]
gap> Trace(x);
3
gap> Determinant(x);
-2
gap> Norm(x);
9/2
```

3.2 Jordan Algebra Constructions

The classification of simple Euclidean Jordan algebras is described in [FK94, chap. 5]. A simple Euclidean Jordan algebra can be constructed in the following two ways. A rank 2 algebra can be constructed from a positive definite Gram matrix in the manner described in [FK94, p. 25]. A degree 1, 2, 4, or 8 algebra can be constructed using Hermitian matrices over a composition algebra of dimension equal to the degree with the product $(xy+yx)/2$. In certain cases both constructions are possible. The ALCO package provides tools to use both constructions to create simple Euclidean Jordan algebras with elements that satisfy `IsSCAlgebraObj`.

3.2.1 SimpleEuclideanJordanAlgebra

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

Returns a simple Euclidean Jordan algebra over $\mathbb{Q}$. The construction used depends on the arguments given in the following manner.

For Jordan algebras of rank *rho* equal to 2, the `JordanSpinFactor` (3.2.2) construction is used. If optional *args* is empty then the result is `JordanSpinFactor(IdentityMat(`*d*`+1))`. If optional *args* is a symmetric matrix of dimension *d*+1, then `JordanSpinFactor(args)` is used. If neither of these rank 2 cases apply, and *d* is equal to 1,2,4, or 8, and if *args* is a composition algebra basis, then `HermitianSimpleJordanAlgebra(`*rho*`, `*args*`)` is used.

In the cases where rank *rho* is greater than 2, we must have *d* equal to one of 1, 2, 4, or 8. Note that *d* equals 8 is only permitted when *rho* equals 3. When optional *args* is a composition algebra basis of dimension *d*, `HermitianSimpleJordanAlgebra(`*rho*`, `*args*`)` is used. Otherwise, when optional *args* is empty, this function uses `HermitianSimpleJordanAlgebra(`*rho*`, `*B*`)` for *B* either `CanonicalBasis(Rationals)`,

`Basis(CF(4), [1, E(4)])`, `CanonicalBasis(QuaternionAlgebra(Rationals))`, or `CanonicalBasis(OctonionAlgebra(Rationals))`.

Note that (in contrast to `AlbertAlgebra` (3.3.1)) the Hermitian Jordan algebras constructed using `SimpleEuclideanJordanAlgebra` uses the upper triangular entries of the Hermitian matrices define the basis vectors.

Example

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

3.2.2 JordanSpinFactor

▷ `JordanSpinFactor(G)` (function)

Returns a Jordan spin factor algebra when `G` is a positive definite Gram matrix. This is the Jordan algebra of rank 2 constructed from a symmetric bilinear form, as described in [FK94, p. 25].

Example

```
gap> J := JordanSpinFactor(IdentityMat(8));
<algebra-with-one 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 := Sum(Basis(J){[4,5,6,7]});
v.4+v.5+v.6+v.7
gap> [Trace(x), Determinant(x)];
[ 0, -4 ]
gap> p := GenericMinimalPolynomial(x);
[ -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> J := HermitianSimpleJordanAlgebra(3,QuaternionD4Basis);
<algebra-with-one of dimension 15 over Rationals>
gap> [JordanRank(J), JordanDegree(J)];
[ 3, 4 ]
```

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. The main definitions and properties of Jordan homotopes and isotopes are given in [McC04, pp.82-86].

Example

```
gap> J := SimpleEuclideanJordanAlgebra(2,7);
<algebra-with-one of dimension 9 over Rationals>
gap> u := Sum(Basis(J){[1,2,7,8]});
v.1+v.2+v.7+v.8
gap> Inverse(u);
(-1/2)*v.1+(1/2)*v.2+(1/2)*v.7+(1/2)*v.8
gap> GenericMinimalPolynomial(u);
[ -2, -2, 1 ]
gap> H := JordanHomotope(J, u, "w.");
<algebra-with-one of dimension 9 over Rationals>
gap> One(H);
(-1/2)*w.1+(1/2)*w.2+(1/2)*w.7+(1/2)*w.8
```

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 package 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-with-one of dimension 27 over Rationals>
```

```
gap> i := Basis(A){[1..8]};;
gap> j := Basis(A){[9..16]};;
gap> k := Basis(A){[17..24]};;
gap> e := Basis(A){[25..27]};;
gap> Display(i); Display(j); Display(k); Display(e);
[ 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 AlbertVectorToHermitianMatrix

▷ `AlbertVectorToHermitianMatrix(`*x*`)` (function)

For an element *x* in `AlbertAlgebra(`*Rationals*`)`, this function returns the corresponding 3 x 3 Hermitian matrix with octonion entries in `OctonionAlgebra(`*Rationals*`)`.

3.3.3 HermitianMatrixToAlbertVector

▷ `HermitianMatrixToAlbertVector(`*x*`)` (function)

For 3 x 3 Hermitian matrix with elements in `OctonionAlgebra(`*Rationals*`)`, this function returns the corresponding vector in in `AlbertAlgebra(`*Rationals*`)`.

Example

```
gap> j := Basis(AlbertAlgebra(Rationals)){[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 quadratic maps and the standard triple product on a Jordan algebra.

3.4.1 JordanQuadraticOperator

▷ `JordanQuadraticOperator(`*x*`[, `*y*`])` (operation)

For *x* and *y* Jordan algebra elements, satisfying `IsJordanAlgebraObj` this operation applies two methods. In the case of `JordanQuadraticOperator(`*x*`, `*y*`)`, this operation returns `2*x*(x*y) - (x^2)*y`. In the case of `JordanQuadraticOperator(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 JordanQuadraticOperator(x) returns the matrix 2 L(x)^2 - L(x^2).

Example

```
gap> J := JordanSpinFactor(IdentityMat(3));
<algebra-with-one of dimension 4 over Rationals>
gap> x := [-1,4/3,-1,1]*Basis(J);
(-1)*v.1+(4/3)*v.2+(-1)*v.3+v.4
gap> y := [-1, -1/2, 2, -1/2]*Basis(J);
(-1)*v.1+(-1/2)*v.2+(2)*v.3+(-1/2)*v.4
gap> JordanQuadraticOperator(x,y);
(14/9)*v.1+(-79/18)*v.2+(-11/9)*v.3+(-53/18)*v.4
gap> JordanQuadraticOperator(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), JordanQuadraticOperator(x)
> *ExtRepOfObj(y)) = JordanQuadraticOperator(x,y);
true
gap> ExtRepOfObj(JordanQuadraticOperator(x,y)) =
> JordanQuadraticOperator(x)*ExtRepOfObj(y);
true
gap> JordanQuadraticOperator(2*x) = 4*JordanQuadraticOperator(x);
true
```

3.4.2 JordanTripleSystem

▷ JordanTripleSystem(*x*, *y*, *z*) (operation)

For Jordan algebra elements *x*, *y*, *z* satisfying IsJordanAlgebraObj, the operation JordanTripleSystem(*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*JordanTripleSystem(*x*,*y*,*z*) is equal to JordanQuadraticOperator(x+z, y) - JordanQuadraticOperator(x, y) - JordanQuadraticOperator(z, y).

Example

```
gap> J := AlbertAlgebra(Rationals);
<algebra-with-one of dimension 27 over Rationals>
gap> i := Basis(J){[1..8]};
[ i1, i2, i3, i4, i5, i6, i7, i8 ]
gap> j := Basis(J){[9..16]};
[ j1, j2, j3, j4, j5, j6, j7, j8 ]
gap> k := Basis(J){[17..24]};
[ k1, k2, k3, k4, k5, k6, k7, k8 ]
gap> e := Basis(J){[25..27]};
[ ei, ej, ek ]
gap> List(i, x -> JordanTripleSystem(i[1],i[1],x));
[ i1, i2, i3, i4, i5, i6, i7, i8 ]
gap> List(j, x -> 2*JordanTripleSystem(i[1],i[1],x));
[ j1, j2, j3, j4, j5, j6, j7, j8 ]
gap> List(k, x -> 2*JordanTripleSystem(i[1],i[1],x));
[ k1, k2, k3, k4, k5, k6, k7, k8 ]
```

```
gap> List(e, x -> JordanTripleSystem(i[1],i[1],x));
[ 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 of 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);;
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 ] ]
gap> AsList(Basis(H));
[ e, i, j, k ]
```

3.5.2 JordanMatrixBasis

▷ JordanMatrixBasis(*J*) (attribute)

If IsJordanAlgebra(*J*) and *J* has been constructed using HermitianSimpleJordanAlgebra (3.2.3), 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);;
gap> J := HermitianSimpleJordanAlgebra(2,Basis(H));
<algebra-with-one of dimension 6 over Rationals>
gap> AsList(CanonicalBasis(J));
[ v.1, v.2, v.3, v.4, v.5, v.6 ]
```

```
gap> JordanMatrixBasis(J);; for x in last 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 ] ]
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-with-one of dimension 10 over Rationals>
gap> List(Basis(J), x -> List(Basis(J), y -> Trace(x*y))) =
> JordanAlgebraGramMatrix(J);
true
gap> DiagonalOfMat(JordanAlgebraGramMatrix(J));
[ 1, 1, 2, 2, 2, 2, 2, 2, 2, 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` (3.1.7) to determine whether x is positive definite.

Chapter 4

Spherical and Projective Designs

A spherical or projective design is a finite subset of a sphere or projective space (see [DGS77] and [Hog82] for more details). Certain designs have special properties and interesting symmetries. The ALCO package allows users to study both spherical and projective designs by modelling both as finite sets of primitive idempotents of a simple Euclidean Jordan algebra.

Specifically, the primitive idempotents of simple Euclidean Jordan algebras of rank 2 have the geometry of a sphere. The correspondence involves converting Euclidean inner product $\cos(\alpha)$ between two unit vectors on a sphere into the corresponding Jordan inner product $\mathrm{Tr}(x \circ y)$ given by $(1+\cos(\alpha))/2$ (described in [Nas23, p. 72]). Likewise, the primitive idempotents of a simple Euclidean Jordan algebras of degrees 1, 2, 4, or 8 have the geometry of a real, complex, quaternion, or octonion projective space.

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 and possible instances may be studied using these tools.

4.1 Jacobi Polynomials

One advantage of studying spherical and projective designs together as sets of Jordan primitive idempotents is both the spherical and projective cases can be studied together using Jacobi polynomials, with suitable parameters chosen for the appropriate simple Euclidean Jordan algebra.

4.1.1 JacobiPolynomial

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

This function returns the Jacobi polynomial $P_k^{(a,b)}(x)$ of degree `k` and type `(a,b)` as defined in [AS72, chap. 22]. The argument `k` must be a non-negative integer. The arguments `a` and `b` must be either rational numbers greater than -1 or must satisfy `IsPolynomial`.

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

The ALCO package defines new categories within `IsObject` in order to construct and study Jordan designs.

4.2.1 Jordan Design Categories

▷ `IsJordanDesign` (filter)
▷ `IsSphericalJordanDesign` (filter)
▷ `IsProjectiveJordanDesign` (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 JordanDesignByParameters

▷ `JordanDesignByParameters(`*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 := JordanDesignByParameters(3,8);
<design with rank 3 and degree 8>
gap> IsJordanDesign(D);
true
gap> IsSphericalJordanDesign(D);
false
gap> IsProjectiveJordanDesign(D);
true
gap> JordanDesignByParameters(4,8);
fail
gap> JordanDesignByParameters(3,9);
fail
```

4.2.3 Jordan Rank and Degree

▷ JordanDesignRank(*D*) (attribute)
▷ JordanDesignDegree(*D*) (attribute)

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

Example

```
gap> D := JordanDesignByParameters(3,8);
<design with rank 3 and degree 8>
gap> [JordanDesignRank(D), JordanDesignDegree(D)];
[ 3, 8 ]
```

4.2.4 JordanDesignQPolynomials

▷ JordanDesignQPolynomials(*D*) (attribute)

Many properties of a Jordan design are computed using the family of renormalized Jacobi polynomials that correspond to the spherical or projective space in question. This attribute stores a function DesignQPolynomial(*D*)(*k*) that returns the *k*-th polynomial in the family, as a list of coefficients, where *k* is a non-negative integer.

Example

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

4.2.5 JordanDesignConnectionCoefficients

▷ JordanDesignConnectionCoefficients(*D*) (attribute)

The connection coefficients of a design *D* determine which linear combinations of JordanDesignQPolynomials(*D*) yield each power of the indeterminate [Hog92, p. 261]. This attribute stores a function JordanDesignConnectionCoefficients(*D*)(*s*) that computes the connection coefficients of each power up to positive integer *s*.

Example

```
gap> D := JordanDesignByParameters(3,8);
<design with rank 3 and degree 8>
gap> JordanDesignConnectionCoefficients(D);
function( s ) ... end
gap> f := JordanDesignConnectionCoefficients(D)(3);; Display(f);
[ [        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]*JordanDesignQPolynomials(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

The angle set of a design is the set of all inner products between distinct elements in the design. In the case of a Jordan design, each inner product is computed as $\mathrm{Tr}(x \circ y)$ for x and y primitive idempotents. This means that the angle set of a design is a set of real numbers in the interval $[0,1)$. We can compute many additional properties of a design once the angle set is known.

4.3.1 IsJordanDesignWithAngleSet

▷ `IsJordanDesignWithAngleSet` (filter)

This filter identifies whether an object that satisfies `IsJordanDesign` is equipped with an angle set.

4.3.2 Design Angle Sets

▷ `JordanDesignAddAngleSet(`*D*`, `*A*`)` (operation)
▷ `JordanDesignAngleSet(`*D*`)` (attribute)

For a design *D* without an angle set, records the angle set *A* as an attribute `JordanDesignAngleSet`. The angle set *A* must be a list of real-valued elements in `IsCyc` in the interval $[0,1)$. Note that when *A* contains irrational elements for which < does not provide an ordering, inclusion in the interval given above is not checked. Also note that the angle set cannot be modified once set as an attribute of the design.

Example

```
gap> D := JordanDesignByParameters(4,4);
<design with rank 4 and degree 4>
gap> JordanDesignAddAngleSet(D, [2]);
fail
gap> D;
<design with rank 4 and degree 4>
gap> JordanDesignAddAngleSet(D, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignAngleSet(D);
[ 1/9, 1/3 ]
```

4.3.3 JordanDesignByAngleSet

▷ `JordanDesignByAngleSet(`*rank*`, `*degree*`, `*A*`)` (function)

This function constructs a new design with Jordan rank and degree given by *rank* and *degree*, with angle set *A*. The same constrains on angle set *A* given in JordanDesignAddAngleSet (4.3.2) apply.

Example

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

4.3.4 JordanDesignNormalizedAnnihilatorPolynomial

▷ JordanDesignNormalizedAnnihilatorPolynomial(*D*) (attribute)

The normalized annihilator polynomial is defined for an angle set A as the polynomial $p(x)$ of degree equal to the cardinality of A with the elements of A for roots and normalization such that $p(1) = 1$ [BBIT21, p. 185]. The coefficients of this polynomial are stored as an attribute of a design with an angle set.

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> p := JordanDesignNormalizedAnnihilatorPolynomial(D);
[ 1/16, -3/4, 27/16 ]
gap> ValuePol(p, 1/9);
0
gap> ValuePol(p, 1/3);
0
gap> ValuePol(p, 1);
1
```

4.3.5 JordanDesignNormalizedIndicatorCoefficients

▷ JordanDesignNormalizedIndicatorCoefficients(*D*) (attribute)

The normalized indicator coefficients are the JordanDesignQPolynomials(*D*)-expansion coefficients of JordanDesignNormalizedAnnihilatorPolynomial(*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 := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> f := JordanDesignNormalizedIndicatorCoefficients(D);
[ 1/64, 7/960, 9/3520 ]
gap> Sum(List([1..3], i -> f[i]*JordanDesignQPolynomials(D)(i-1)));
[ 1/16, -3/4, 27/16 ]
gap> JordanDesignNormalizedAnnihilatorPolynomial(D);
[ 1/16, -3/4, 27/16 ]
```

4.3.6 IsJordanDesignWithPositiveIndicatorCoefficients

▷ IsJordanDesignWithPositiveIndicatorCoefficients (filter)

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

4.3.7 JordanDesignSpecialBound

▷ `JordanDesignSpecialBound(D)` (attribute)

The special bound of a design satisfying `IsJordanDesignWithPositiveIndicatorCoefficients` is the upper limit on the possible cardinality for the given angle set [Hog92, pp. 257-258]. This attribute stores the special bound when it exists for a design.

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> IsJordanDesignWithPositiveIndicatorCoefficients(D);
true
gap> JordanDesignSpecialBound(D);
64
```

4.4 Designs with Angle Set and Cardinality

Many more properties of a design with an angle set can be computed once the cardinality of the design is also known. In what follows let v be the cardinality of a design and let s be the cardinality of the angle set A of that design.

4.4.1 Design Cardinality

▷ `IsJordanDesignWithCardinality` (filter)
▷ `JordanDesignAddCardinality(D, v)` (operation)
▷ `JordanDesignCardinality(D)` (attribute)

As a finite set, each design has a cardinality. When this cardinality is known for an object `D` that satisfies `IsJordanDesign(D)`, the cardinality is stored as the attribute `JordanDesignCardinality(D)`. In order to set the cardinality of a design, we can use the operation `JordanDesignAddCardinality(D, v)`. When `JordanDesignAddCardinality` is called, the ALCO package immediately attempts to compute `JordanDesignStrength` (4.4.5).

Example

```
gap> D := JordanDesignByAngleSet(4,4, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> HasJordanDesignCardinality(D);
false
gap> JordanDesignAddCardinality(D, 64);
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignCardinality(D);
64
```

4.4.2 Designs at the Special Bound

▷ `IsSpecialBoundJordanDesign` (filter)

As described in JordanDesignSpecialBound (4.3.7), we can compute the special bound of a design using the angle set. Once the cardinality is also known we can assess whether the design reaches the special bound. This filter identifies when a design with an angle set and cardinality also meets the special bound.

4.4.3 JordanDesignAnnihilatorPolynomial

▷ JordanDesignAnnihilatorPolynomial(*D*) (attribute)

The annihilator polynomial for design *D* is defined by multiplying the JordanDesignNormalizedAnnihilatorPolynomial(*D*) by JordanDesignCardinality(*D*).

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);;
gap> JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignAnnihilatorPolynomial(D);
[ 4, -48, 108 ]
gap> ValuePol(last, 1);
64
```

4.4.4 JordanDesignIndicatorCoefficients

▷ JordanDesignIndicatorCoefficients(*D*) (attribute)

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

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignIndicatorCoefficients(D);
[ 1, 7/15, 9/55 ]
```

4.4.5 Design Strength

▷ IsJordanDesignWithStrength (filter)
▷ JordanDesignStrength(*D*) (attribute)

The *t*-design is a design with the following special property: the integral of any degree *t* polynomial over the sphere or projective space containing the design is equal to the average value of that polynomial evaluated at the points of the *t*-design (see [DGS77] and [Hog82] for detailed definitions). The parameter *t* is called the *strength* of the design.

For a design *D* that satisfies IsJordanDesignWithPositiveIndicatorCoefficients, IsJordanDesignWithCardinality, and IsSpecialBoundJordanDesign, we can compute the strength *t* of the design using a theorem given in [Hog92, p. 258] that examines the indicator coefficients. The filter IsJordanDesignWithStrength indicates when the attribute JordanDesignStrength has been successfully computed.

Example

```
gap> D := JordanDesignByAngleSet(4,4, [1/3,1/9]);
<design with rank 4, degree 4, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignAddCardinality(D, 64);
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> IsJordanDesignWithStrength(D);
true
gap> JordanDesignStrength(D);
2
```

4.4.6 Schemes and Tight Designs

▷ IsRegularSchemeJordanDesign (filter)
▷ IsAssociationSchemeJordanDesign (filter)
▷ IsTightJordanDesign (filter)

These filters identify various special categories of designs that satisfy IsJordanDesignWithStrength (4.4.5). In what follows recall that t denotes the strength of the design and s denotes the cardinality of the angle set A. The definitions below are provided in [Hog92].

A design admits a *regular scheme* when $t \geq s-1$. The filter IsRegularSchemeJordanDesign returns true when both t and s are known and satisfy the regular scheme inequality given above.

A design admits an *association scheme* when $t \geq 2s-2$. The filter IsAssociationSchemeJordanDesign returns true when both t and s are known and satisfy the association scheme inequality given above.

Finally, a design is *tight* when it satisfies $t = 2s-1$ for 0 in A or $t = 2s$ otherwise. The filter IsTightJordanDesign returns true when the appropriate equality is satisfied for a design.

4.5 Designs Admitting a Regular Scheme

4.5.1 JordanDesignSubdegrees

▷ JordanDesignSubdegrees(*D*) (attribute)

For a design D with cardinality and angle set that satisfies IsRegularSchemeJordanDesign, namely $t \geq s-1$, we can compute the regular subdegrees as described in [Hog92, Theorem 3.2]. The subdegrees count the number of elements forming each angle with some representative element in the design. So, in the example below, there are 27 elements forming an angle (inner product) of $1/9$ with some representative design element.

Example

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

4.6 Designs Admitting an Association Scheme

When a design satisfies $t \geq 2s-2$ then it also admits an association scheme. We can use results given in [Hog92] to determine the parameters of the corresponding association scheme. For more details about association schemes see [CVL91] or [BBIT21].

4.6.1 JordanDesignBoseMesnerAlgebra

▷ `JordanDesignBoseMesnerAlgebra(D)` (attribute)

For a design that satisfies `IsAssociationSchemeJordanDesign`, 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. The adjacenty matrices themselves are not provided and the algebra is constructed from the known structure constants so that elements of this algebra satisfy `IsSCAlgebraObj`.

Example

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

4.6.2 JordanDesignBoseMesnerIdempotentBasis

▷ `JordanDesignBoseMesnerIdempotentBasis(D)` (attribute)

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

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> for x in BasisVectors(JordanDesignBoseMesnerIdempotentBasis(D)) do Display(x);
> od;
(-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> ForAll(JordanDesignBoseMesnerIdempotentBasis(D), IsIdempotent);
true
```

4.6.3 JordanDesignIntersectionNumbers

▷ `JordanDesignIntersectionNumbers(D)` (attribute)

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

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> A := BasisVectors(Basis(JordanDesignBoseMesnerAlgebra(D)));;
gap> p := JordanDesignIntersectionNumbers(D);;
gap> A[1]*A[2] = Sum(List([1..3]), k -> p[k][1][2]*A[k]);
true
```

4.6.4 JordanDesignKreinNumbers

▷ `JordanDesignKreinNumbers(D)` (attribute)

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

Example

```
gap> D := JordanDesignByAngleSet(4, 4, [1/3, 1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> q := JordanDesignKreinNumbers(D);;
gap> Display(q);
[ [ [ 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 JordanDesignFirstEigenmatrix

▷ `JordanDesignFirstEigenmatrix(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 `JordanDesignFirstEigenmatrix(D)[i][j]`.

Example

```
gap> D := JordanDesignByAngleSet(4,4,[1/3,1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> a := Basis(JordanDesignBoseMesnerAlgebra(D));;
gap> e := JordanDesignBoseMesnerIdempotentBasis(D);;
gap> ForAll([1..3], i -> a[i] = Sum([1..3], j ->
> JordanDesignFirstEigenmatrix(D)[i][j]*e[j]));
true
```

4.6.6 JordanDesignSecondEigenmatrix

▷ `JordanDesignSecondEigenmatrix(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 `JordanDesignSecondEigenmatrix(D)[i][j]`.

Example

```
gap> D := JordanDesignByAngleSet(4,4,[1/3,1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> a := Basis(JordanDesignBoseMesnerAlgebra(D));;
gap> e := JordanDesignBoseMesnerIdempotentBasis(D);;
gap> ForAll([1..3], i -> e[i]*JordanDesignCardinality(D) =
> Sum([1..3], j -> JordanDesignSecondEigenmatrix(D)[i][j]*a[j]));
true
gap> JordanDesignFirstEigenmatrix(D) = Inverse(JordanDesignSecondEigenmatrix(D))
> *JordanDesignCardinality(D);
true
```

4.6.7 JordanDesignMultiplicities

▷ `JordanDesignMultiplicities(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 $m_i = \mathrm{Tr}(E_i)$. We also have $m_i = Q_i(s+1)$.

Example

```
gap> D := JordanDesignByAngleSet(4,4,[1/3,1/9]);; JordanDesignAddCardinality(D, 64);; D;
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignMultiplicities(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 := JordanDesignByAngleSet(4,4,[1/3,1/9]);; JordanDesignAddCardinality(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 JordanDesignReducedAdjacencyMatrices

▷ `JordanDesignReducedAdjacencyMatrices(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 `JordanDesignReducedAdjacencyMatrices(D)` are used to construct `JordanDesignBoseMesnerAlgebra(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> JordanDesignByAngleSet(2, 1, [0,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 2, degree 1, cardinality 4, and angle set
[ 0, 1/2 ]>
gap> JordanDesignByAngleSet(2, 2, [0,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 2, degree 2, cardinality 6, and angle set
[ 0, 1/2 ]>
gap> JordanDesignByAngleSet(2, 4, [0,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 2, degree 4, cardinality 10, and angle set
[ 0, 1/2 ]>
gap> JordanDesignByAngleSet(2, 8, [0,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 2, degree 8, cardinality 18, and angle set
[ 0, 1/2 ]>
gap> JordanDesignByAngleSet(3, 2, [1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 2-design with rank 3, degree 2, cardinality 9, and angle set [ 1/4 ]>
gap> JordanDesignByAngleSet(4, 2, [0,1/3]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 4, degree 2, cardinality 40, and angle set
[ 0, 1/3 ]>
gap> JordanDesignByAngleSet(6, 2, [0,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 6, degree 2, cardinality 126, and angle set
[ 0, 1/4 ]>
gap> JordanDesignByAngleSet(8, 2, [1/9]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 2-design with rank 8, degree 2, cardinality 64, and angle set [ 1/9 ]>
gap> JordanDesignByAngleSet(5, 4, [0,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 3-design with rank 5, degree 4, cardinality 165, and angle set
[ 0, 1/4 ]>
gap> JordanDesignByAngleSet(3, 8, [0,1/4,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 5-design with rank 3, degree 8, cardinality 819, and angle set
[ 0, 1/4, 1/2 ]>
gap> JordanDesignByAngleSet(24, 1, [0,1/16,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<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> JordanDesignByAngleSet(2, 2, [ 0, (5-Sqrt(5))/10, (5+Sqrt(5))/10 ]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
```

```
<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> JordanDesignByAngleSet(2, 23, [ 0, 1/4, 3/8, 1/2, 5/8, 3/4 ]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(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> JordanDesignByAngleSet(2, 5, [ 1/4, 5/8 ]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 4-design with rank 2, degree 5, cardinality 27, and angle set
[ 1/4, 5/8 ]>
gap> JordanDesignByAngleSet(2, 6, [0,1/3,2/3]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 5-design with rank 2, degree 6, cardinality 56, and angle set
[ 0, 1/3, 2/3 ]>
gap> JordanDesignByAngleSet(2, 21, [3/8, 7/12]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 4-design with rank 2, degree 21, cardinality 275, and angle set
[ 3/8, 7/12 ]>
gap> JordanDesignByAngleSet(2, 22, [0,2/5,3/5]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 5-design with rank 2, degree 22, cardinality 552, and angle set
[ 0, 2/5, 3/5 ]>
gap> JordanDesignByAngleSet(2, 7, [0,1/4,1/2,3/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<Tight 7-design with rank 2, degree 7, cardinality 240, and angle set
[ 0, 1/4, 1/2, 3/4 ]>
gap> JordanDesignByAngleSet(2, 22, [0,1/3,1/2,2/3]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(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> JordanDesignByAngleSet(4, 4, [0,1/4,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<3-design with rank 4, degree 4, cardinality 180, and angle set
[ 0, 1/4, 1/2 ]>
gap> JordanDesignByAngleSet(3, 2, [0,1/3]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 3, degree 2, cardinality 12, and angle set [ 0, 1/3 ]>
gap> JordanDesignByAngleSet(5, 2, [0,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 5, degree 2, cardinality 45, and angle set [ 0, 1/4 ]>
gap> JordanDesignByAngleSet(9, 2, [0,1/9]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 9, degree 2, cardinality 90, and angle set [ 0, 1/9 ]>
gap> JordanDesignByAngleSet(28, 2, [0,1/16]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
```

```
<2-design with rank 28, degree 2, cardinality 4060, and angle set [ 0, 1/16 ]>
gap> JordanDesignByAngleSet(4, 4, [0,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 4, degree 4, cardinality 36, and angle set [ 0, 1/4 ]>
gap> JordanDesignByAngleSet(4, 4, [1/9,1/3]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 4, degree 4, cardinality 64, and angle set [ 1/9, 1/3 ]>
gap> JordanDesignByAngleSet(16, 1, [0,1/9]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<2-design with rank 16, degree 1, cardinality 256, and angle set [ 0, 1/9 ]>
gap> JordanDesignByAngleSet(4, 2, [0,1/4,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<3-design with rank 4, degree 2, cardinality 60, and angle set
[ 0, 1/4, 1/2 ]>
gap> JordanDesignByAngleSet(16, 1, [0,1/16,1/4]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<3-design with rank 16, degree 1, cardinality 2160, and angle set
[ 0, 1/16, 1/4 ]>
gap> JordanDesignByAngleSet(3, 4, [0,1/4,1/2]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<3-design with rank 3, degree 4, cardinality 63, and angle set
[ 0, 1/4, 1/2 ]>
gap> JordanDesignByAngleSet(3, 4, [0,1/4,1/2,(3+Sqrt(5))/8, (3-Sqrt(5))/8]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(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> JordanDesignByAngleSet(12, 2, [0,1/3,1/4,1/12]);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(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 := JordanDesignByAngleSet(2, 3, A);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<11-design with rank 2, degree 3, cardinality 120, and angle set
[ 0, 1/4, 1/2, 3/4, -3/4*E(5)-1/2*E(5)^2-1/2*E(5)^3-3/4*E(5)^4,
  -1/2*E(5)-3/4*E(5)^2-3/4*E(5)^3-1/2*E(5)^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 ]>
gap> A := [ 0, 1/4, (3-Sqrt(5))/8, (3+Sqrt(5))/8 ];;
gap> D := JordanDesignByAngleSet(4, 1, A);;
gap> JordanDesignAddCardinality(last, JordanDesignSpecialBound(last));
<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

An *octonion vector* is a tuple of octonions. Vector addition and scalar multiplication are defined in the usual way, as component-wise addition and scalar multiplication. We can define the *octonion vector norm* and corresponding inner product for an octonion vector in a number of ways, but require that the octonion norm of an octonion vector belongs to the ring of scalars.

An *octonion lattice* is a free left $\mathbb{Z}$-module generated by a finite set of octonion vectors along with some octonion vector norm. Octonion lattice constructions of the Leech lattice are explored in several places, including [EG96], [Wil09b], [Wil11], [Nas22], [Nas23].

The ALCO package provides tools to construct and study octonion lattices, including the Leech lattice constructions given in the references above. This package constructs octonion lattices in GAP as free left modules (that satisfy `IsFreeLeftModule`) with the additional structure needed to compute the norms of lattice points and their inner product. It also includes tools to identify the Leech lattice and Gosset (E_8) lattice by examining the Gram matrix of a lattice.

5.1 Gram Matrices and Octonion Lattices

The Gram matrix of a lattice basis is a computed by taking the inner products of the basis vectors. When the inner product is real-valued, the Gram matrix of a lattice basis must be a positive definite symmetric matrix (see [CS13, chap. 1]). Certain important lattices may be identified using a Gram matrix of that lattice. The ALCO package provides the following tools to identify a Gosset or Leech lattice Gram matrix.

5.1.1 IsLeechLatticeGramMatrix

▷ `IsLeechLatticeGramMatrix(G)` (function)

This function tests whether `G` is a Gram matrix of the Leech lattice. In order for the function to return `true`, argument `G` must satisfy the following. First, `G` must satisfy `IsOrdinaryMatrix(G)`, `DimensionsMat(G) = [24,24]`, `TransposedMat(G) = G`, and `ForAll(Flat(G), IsInt)`. If so, then this function verifies that `G` belongs to a unimodular lattice and computes `ShortestVectors(G, 3)`. Provided that no vectors of norm 1, 2, or 3 exist, then by the classification of unimodular lattices in 24 dimensions, Gram matrix `G` must belong to a Leech lattice. For details about the classification see [CS13, chaps. 16-18].

5.1.2 IsGossetLatticeGramMatrix

▷ `IsGossetLatticeGramMatrix(G)` (function)

This function tests whether `G` is a Gram matrix of the Gosset lattice, also known as the E_8 lattice. In order for the function to return `true`, argument `G` must satisfy the following. First, `G` must satisfy `IsOrdinaryMatrix(G)`, `DimensionsMat(G) = [8,8]`, `TransposedMat(G) = G`, and `ForAll(Flat(G), IsInt)`. If so, then this function verifies that `G` belongs to a unimodular lattice and computes `ShortestVectors(G, 1)`. Provided that no vectors of norm 1 exist, then by the classification of unimodular lattices in 8 dimensions, Gram matrix `G` must belong to a Gosset lattice. For details about the classification see [CS13, chaps. 16-18].

5.1.3 Miracle Octad Generator (MOG) Coordinates

▷ `MOGLeechLatticeGeneratorMatrix` (global variable)
▷ `MOGLeechLatticeGramMatrix` (global variable)

The Leech lattice is studied in detail in [CS13] using a specific set of coordinates, known as "Miracle Octad Generator" coordinates. The choice of MOG coordinates exhibits the M_{24} symmetries of the lattice. The ALCO package loads the following variables to construct the Leech lattice in MOG coordinates. The variable `MOGLeechLatticeGeneratorMatrix` stores the 24×24 integer matrix with rows that span a MOG 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> M := MOGLeechLatticeGeneratorMatrix;;
gap> G := M*TransposedMat(M)/8;;
gap> G = MOGLeechLatticeGramMatrix;
true
gap> IsLeechLatticeGramMatrix(G);
true
```

5.1.4 IsOctonionLattice

▷ `IsOctonionLattice` (filter)

As described above, an octonion lattice is a free left $\mathbb{Z}$-module generated by a finite set of octonion vectors and equipped with some octonion vector norm. The category `IsOctonionLattice` is a subcategory of `IsFreeLeftModule` used by the ALCO package to construct octonion lattices as free left modules with the additional structure of a norm on octonion vectors. The simplest available norm is perhaps to set the norm of an octonion vector to be the sum of the norms of the octonion coefficients. The norm for an octonion lattice in the ALCO package is defined (via an inner product) in a more general way, described below.

Let `L` be an object that satisfies `IsOctonionLattice(L)`. Then `L` is equipped with an attribute `G = OctonionGramMatrix(L)`. The matrix `G` is a Hermitian octonion matrix used to define the octonion lattice norm and inner product. For `x`, `y` octonion vectors in `L`, the inner product of `x` and `y` is computed as `Trace(x*G*ComplexConjugate(y))`. The trace (twice the real part) satisfies `Trace((x*y)*z) = Trace(x*(y*z))` for any octonions `x,y,z` [Wil09a, p. 145]. It follows that the inner product defined above for octonion vectors is also well defined despite the non-associativity

of the octonion algebra. If we define the norm as the inner product of a vector with itself, then we can set the vector norm to be the sum of the norms of the octonion coefficients by setting `G = OctonionGramMatrix(L)` to be half the identity matrix.

5.1.5 OctonionLatticeByGenerators

▷ `OctonionLatticeByGenerators(gens[, G[, B]])` (function)

This function returns a free left module that satisfies `IsOctonionLattice` (5.1.4). The attribute `LeftActingDomain` is automatically set to `Integers`. Argument *gens* must be a list of equal length octonion row vectors that satisfies `IsOctonionCollColl(gens)`. The inner product is defined by optional argument *G*, which is an octonion Hermitian matrix that defaults to half the identity matrix. Optional argument *B* is a basis for the octonion algebra used to construct the vectors and defaults to the canonical basis of that algebra. All three arguments must satisfy `IsHomogeneousList(Flat([gens, G, B]))` in order to ensure that the same octonion algebra is being used in each argument.

The input *gens* is stored as the attribute `GeneratorsOfLeftOperatorAdditiveGroup`. The vectors in *gens* need not be linearly independent. The octonion algebra basis *B* is used to convert *gens* to a list of real coefficients using `OctonionToRealVector(B, x)` for each *x* in *gens*. These converted vectors are stored as the attribute `GeneratorsAsCoefficients` and are used to compute a basis for the lattice which is stored as the attribute `LLLReducedBasisCoefficients`.

The argument *G* is stored as the attribute `OctonionGramMatrix` and is used to define `ScalarProduct(L, x, y) = Trace(x*G*ComplexConjugate(y))` for `IsOctonionLattice(L)` and *x,y* octonion vectors in *L*.

In the following example, we construct the octonion lattice consisting of all octavian integer triples where the norm of a vector is the sum of the norms of the coefficients. We also supply the E_8 basis for the octavian integers as the underlying octonion ring basis *B* for the construction.

Example

```
gap> O := OctavianIntegers;;
gap> gens := Concatenation(List(IdentityMat(3), x -> List(Basis(O), y -> x*y)));
[ [ (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7, 0*e1, 0*e1 ],
  [ (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e1+(-1/2)*e3+(1/2)*e4+(1/2)*e5, 0*e1, 0*e1 ],
  [ (-1/2)*e2+(1/2)*e3+(-1/2)*e5+(1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e2+(-1/2)*e4+(1/2)*e5+(-1/2)*e6, 0*e1, 0*e1 ],
  [ (-1/2)*e1+(-1/2)*e3+(1/2)*e4+(-1/2)*e5, 0*e1, 0*e1 ],
  [ (1/2)*e1+(-1/2)*e4+(1/2)*e6+(-1/2)*e8, 0*e1, 0*e1 ],
  [ 0*e1, (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7, 0*e1 ],
  [ 0*e1, (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e1+(-1/2)*e3+(1/2)*e4+(1/2)*e5, 0*e1 ],
  [ 0*e1, (-1/2)*e2+(1/2)*e3+(-1/2)*e5+(1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e2+(-1/2)*e4+(1/2)*e5+(-1/2)*e6, 0*e1 ],
  [ 0*e1, (-1/2)*e1+(-1/2)*e3+(1/2)*e4+(-1/2)*e5, 0*e1 ],
  [ 0*e1, (1/2)*e1+(-1/2)*e4+(1/2)*e6+(-1/2)*e8, 0*e1 ],
  [ 0*e1, 0*e1, (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7 ],
  [ 0*e1, 0*e1, (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7 ],
  [ 0*e1, 0*e1, (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7 ],
  [ 0*e1, 0*e1, (1/2)*e1+(-1/2)*e3+(1/2)*e4+(1/2)*e5 ],
  [ 0*e1, 0*e1, (-1/2)*e2+(1/2)*e3+(-1/2)*e5+(1/2)*e7 ],
```

```
    [ 0*e1, 0*e1, (1/2)*e2+(-1/2)*e4+(1/2)*e5+(-1/2)*e6 ],
    [ 0*e1, 0*e1, (-1/2)*e1+(-1/2)*e3+(1/2)*e4+(-1/2)*e5 ],
    [ 0*e1, 0*e1, (1/2)*e1+(-1/2)*e4+(1/2)*e6+(-1/2)*e8 ] ]
gap> G := IdentityMat(3)*One(O)/2;;
gap> B := CanonicalBasis(O);;
gap> O3 := OctonionLatticeByGenerators(gens, G, B);
<free left module over Integers, with 24 generators>
gap> KnownAttributesOfObject(O3);
[ "LeftActingDomain", "Dimension", "GeneratorsOfLeftOperatorAdditiveGroup",
  "UnderlyingOctonionRing", "UnderlyingOctonionRingBasis",
  "OctonionGramMatrix", "GeneratorsAsCoefficients",
  "LLLReducedBasisCoefficients" ]
```

5.2 Octonion Lattice Attributes

An free left module *L* that satisfies `IsOctonionLattice(`*L*`)` has several attributes in addition to those of a free left module (such as `LeftActingDomain` and `GeneratorsOfLeftOperatorAdditiveGroup`). In the examples of this section, we use the octonion lattice `O3` constructed above to illustrate these attributes.

5.2.1 UnderlyingOctonionRing

▷ `UnderlyingOctonionRing(`*L*`)` (attribute)

When *L* satisfies `IsOctonionLattice(`*L*`)`, this attribute stores the octonion algebra to which the coefficients of the generating octonion vectors *gens* belong. This algebra is determined by `FamilyObj(One(Flat(`*gens*`)))!.fullSCAlgebra` where *gens* is given by `GeneratorsOfLeftOperatorAdditiveGroup(`*L*`)`.

Example

```
gap> UnderlyingOctonionRing(O3);
<algebra-with-one of dimension 8 over Rationals>
```

5.2.2 OctonionGramMatrix

▷ `OctonionGramMatrix(`*L*`)` (attribute)

When *L* satisfies `IsOctonionLattice(`*L*`)`, this attribute stores the optional argument *G* of `OctonionLatticeByGenerators` (5.1.5). When the optional argument is not supplied, the default value is half the appropriate octonion identity matrix. This octonion matrix is used to compute `ScalarProduct` (5.3.2) between lattice vectors.

Example

```
gap> OctonionGramMatrix(O3);; Display(last);
[ [  (1/2)*e8,      0*e1,      0*e1 ],
  [      0*e1,  (1/2)*e8,      0*e1 ],
  [      0*e1,      0*e1,  (1/2)*e8 ] ]
```

5.2.3 GeneratorsAsCoefficients

▷ GeneratorsAsCoefficients(*L*) (attribute)

When *L* satisfies IsOctonionLattice(*L*), this attribute converts the lattice generators, GeneratorsOfLeftOperatorAdditiveGroup(*L*), into a list of coefficients using OctonionToRealVector(*B*, *x*) for each generator *x*. Recall that *B* is an optional argument of OctonionLatticeByGenerators (5.1.5) that defaults to CanonicalBasis(UnderlyingOctonionRing(*L*)).

Example

```
gap> GeneratorsAsCoefficients(O3);
[ [ 1, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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 ] ]
```

5.2.4 LLLReducedBasisCoefficients

▷ LLLReducedBasisCoefficients(*L*) (attribute)

When *L* satisfies IsOctonionLattice(*L*), this attribute stores the LLL reduced basis computed from GeneratorsAsCoefficients(*L*). These vectors define the CanonicalBasis (5.2.7) for *L* and are used to compute the attribute GramMatrix (5.2.6).

Example

```
gap> LLLReducedBasisCoefficients(O3);
[ [ 1, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
```

```
    [ 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 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, 0, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0 ],
    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 ] ]
```

5.2.5 Octonion Lattice Dimension

▷ `Dimension(`*L*`)` (attribute)

▷ `Rank(`*L*`)` (attribute)

When *L* satisfies `IsOctonionLattice(`*L*`)`, these attributes store the lattice dimension, also known as rank. This is calculated by evaluating `Rank( GeneratorsAsCoefficients(`*L*`) )`.

Example

```
gap> Dimension(O3);
24
gap> Rank(O3);
24
```

5.2.6 GramMatrix (Octonion Lattices)

▷ `GramMatrix(`*L*`)` (attribute)

When *L* satisfies `IsOctonionLattice(`*L*`)`, this attribute stores the Gram matrix defined by the basis stored using the `LLLReducedBasisCoefficients` (5.2.4) attribute and the inner product defined on the lattice by `ScalarProduct` (5.3.2).

Example

```
gap> g := 2*GramMatrix(O3);;
gap> DimensionsMat(g);
[ 24, 24 ]
gap> DeterminantMat(g);
1
```

5.2.7 Lattice Basis

▷ `Basis(`*L*`)` (attribute)

▷ `CanonicalBasis(`*L*`)` (attribute)

▷ BasisVectors(*B*) (attribute)
▷ IsOctonionLatticeBasis (filter)

When *L* satisfies IsOctonionLattice(*L*), these attributes are used to store the lattice canonical basis determined by LLLReducedBasisCoefficients (5.2.4). The attributes Basis(*L*) and CanonicalBasis(*L*) are equivalent and satisfy the filter IsOctonionLatticeBasis. The attribute BasisVectors returns the vectors stored in the attribute LLLReducedBasisCoefficients (5.2.4) converted into octonion vectors using RealToOctonionVector(UnderlyingOctonionRingBasis(*L*), *x*) for each *x* in LLLReducedBasisCoefficients(*L*).

Example

```
gap> IsOctonionLatticeBasis(Basis(O3));
true
gap> b := BasisVectors(Basis(O3));
[ [ (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7, 0*e1, 0*e1 ],
  [ (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e1+(1/2)*e2+(1/2)*e4+(-1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7, 0*e1, 0*e1 ],
  [ (1/2)*e1+(1/2)*e3+(1/2)*e4+(-1/2)*e5, 0*e1, 0*e1 ],
  [ (1/2)*e1+(1/2)*e2+(1/2)*e3+(-1/2)*e6, 0*e1, 0*e1 ],
  [ (1/2)*e2+(1/2)*e4+(-1/2)*e5+(-1/2)*e6, 0*e1, 0*e1 ],
  [ (1/2)*e1+(1/2)*e2+(-1/2)*e5+(-1/2)*e8, 0*e1, 0*e1 ],
  [ 0*e1, (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7, 0*e1 ],
  [ 0*e1, (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e1+(1/2)*e2+(1/2)*e4+(-1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7, 0*e1 ],
  [ 0*e1, (1/2)*e1+(1/2)*e3+(1/2)*e4+(-1/2)*e5, 0*e1 ],
  [ 0*e1, (1/2)*e1+(1/2)*e2+(1/2)*e3+(-1/2)*e6, 0*e1 ],
  [ 0*e1, (1/2)*e2+(1/2)*e4+(-1/2)*e5+(-1/2)*e6, 0*e1 ],
  [ 0*e1, (1/2)*e1+(1/2)*e2+(-1/2)*e5+(-1/2)*e8, 0*e1 ],
  [ 0*e1, 0*e1, (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7 ],
  [ 0*e1, 0*e1, (-1/2)*e1+(-1/2)*e2+(-1/2)*e4+(-1/2)*e7 ],
  [ 0*e1, 0*e1, (1/2)*e1+(1/2)*e2+(1/2)*e4+(-1/2)*e7 ],
  [ 0*e1, 0*e1, (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-1/2)*e7 ],
  [ 0*e1, 0*e1, (1/2)*e1+(1/2)*e3+(1/2)*e4+(-1/2)*e5 ],
  [ 0*e1, 0*e1, (1/2)*e1+(1/2)*e2+(1/2)*e3+(-1/2)*e6 ],
  [ 0*e1, 0*e1, (1/2)*e2+(1/2)*e4+(-1/2)*e5+(-1/2)*e6 ],
  [ 0*e1, 0*e1, (1/2)*e1+(1/2)*e2+(-1/2)*e5+(-1/2)*e8 ] ]
gap> GramMatrix(O3) = List(b, x -> List(b, y -> ScalarProduct(O3, x, y)));
true
```

5.3 Octonion Lattice Operations

There are some additional operations available to study octonion lattices. In the examples that follow, we continue to use the lattice O3 defined in the example of OctonionLatticeByGenerators (5.1.5).

5.3.1 \in

▷ \in(*x*, *L*) (operation)

When L satisfies IsOctonionLattice(L) and x is an octonion row vector that satisfies IsOctonionCollection and IsRowVector, this operation test whether vector x is in lattice L. This operation will return fail when x and L do not share the same underlying octonion ring. Note that \in(x,L) and x in L are equivalent expression.

Example

```
gap> x := Sum(BasisVectors(Basis(O3)){[2,3,4]});
[ (1/2)*e2+(1/2)*e3+(-1/2)*e5+(-3/2)*e7, 0*e1, 0*e1 ]
gap> x in O3;
true
gap> \in( x, O3);
true
```

5.3.2 ScalarProduct (Octonion Lattices)

▷ ScalarProduct(L, x, y) (operation)

When L satisfies IsOctonionLattice(L) and x, y satisfy IsOctonionCollection and IsRowVector then this operation returns Trace(x*OctonionGramMatrix(L)*ComplexConjugate(y)). Here the trace is the method Trace (2.2.2) acting on an octonion argument (corresponding to twice the real part, or identity coefficient, of the octonion x*OctonionGramMatrix(L)*ComplexConjugate(y)). When x, y are not octonion vectors but rather coefficient vectors of the appropriate length, withIsHomogeneousList(Flat([x,y,GeneratorsAsCoefficients(L)])), then the operation ScalarProduct converts x and y to suitable octonion vectors to compute the scalar product.

Example

```
gap> b := BasisVectors(Basis(O3));;
gap> b[1];
[ (-1/2)*e1+(1/2)*e5+(1/2)*e6+(1/2)*e7, 0*e1, 0*e1 ]
gap> GramMatrix(O3) = List(b, x -> List(b, y -> ScalarProduct(O3, x, y)));
true
gap> c := LLLReducedBasisCoefficients(O3);;
gap> c[1];
[ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> GramMatrix(O3) = List(c, x -> List(c, y -> ScalarProduct(O3, x, y)));
true
```

5.3.3 Sublattice Identification

▷ IsSublattice(L, M) (operation)
▷ IsSubset(L, M) (operation)
▷ \=(L, M) (operation)

When L and M both satisfy IsOctonionLattice(L), these operations determine whether M is a sublattice of L by checking whether the basis vectors for M are in L. If the lattices L and M do not share the same UnderlyingOctonionRing (5.2.1) then the operation will return fail. The operation \= returns the value of IsSublattice(L, M) and IsSublattice(L, M). In the example below we construct an octonion Leech lattice Leech and verify that it is a sublattice of O3.

Example

```
gap> s := LinearCombination(OctonionE8Basis, [ 1, 2, 1, 2, 2, 2, 2, 1 ]);;
gap> leech_gens := List(Basis(OctavianIntegers), x ->
> x*[[s,s,0],[0,s,s],ComplexConjugate([s,s,s])]);;
gap> leech_gens := Concatenation(leech_gens);;
gap> Leech := OctonionLatticeByGenerators(leech_gens, G, B);
<free left module over Integers, with 24 generators>
gap> IsLeechLatticeGramMatrix(GramMatrix(Leech));
true
gap> IsSublattice(Leech, O3);
false
gap> IsSublattice(O3, Leech);
true
gap> IsSubset(O3, Leech);
true
```

5.3.4 Lattice Vector Coefficients

▷ `Coefficients(B, y)` (operation)

Let B satisfy `IsOctonionLatticeBasis(B)` with basis B belonging to octonion lattice L. For y in L such that `IsOctonionCollection(y)`, the coefficients in y in the lattice basis B can be determined by `Coefficients(B, y)`. For x an integer row vector of suitable length, the linear combination of basis vectors is computed in the usual way as `LinearCombination(B, x)`.

Example

```
gap> rand := Random(O3);;
gap> coeffs := Coefficients(Basis(O3), rand);;
gap> rand = LinearCombination(Basis(O3), coeffs);
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 large or possibly 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 ]
```

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 RandomElementClosure

▷ `RandomElementClosure(`*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. If the size of the expected closed set is known, use of a `repeat-until` loop can permit generating the full set of elements faster than `Closure` (6.1.1) would.

Example

```
gap> start := Basis(QuaternionAlgebra(Rationals)){[2,3]};
[ i, j ]
gap> repeat
> start := RandomElementClosure(start, \*);
> until Length(start) = 8;
gap> start;
[ (-1)*e, (-1)*i, (-1)*j, (-1)*k, k, j, i, e ]
```

6.2.2 RandomOrbitOnSets

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

This function proceeds in a manner very similar to `RandomElementClosure` (6.2.1) with the following differences. This function instead selects a 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 `RandomElementClosure` (6.2.1) apply. Note that while *start* is always a subset of the output, the elements of *gens* are not necessarily included in the output.

Example

```
gap> start := Basis(QuaternionAlgebra(Rationals)){[1,2]};
[ e, i ]
gap> gens := Basis(QuaternionAlgebra(Rationals)){[3]};
[ j ]
gap> repeat
> start := RandomOrbitOnSets(gens, start, {x,y} -> x*y);
> until Length(start) = 8;
```

```
gap> start;
[ (-1)*e, (-1)*i, (-1)*j, (-1)*k, k, j, i, e ]
```

References

[AS72] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of Mathematical Functions*. Dover, New York, 1972. 25

[Bae02] John Baez. The octonions. *Bulletin of the American Mathematical Society*, 39(2):145–205, 2002. 7

[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. 29, 33, 34, 35

[CS03] John H. Conway and Derek A. Smith. *On Quaternions and Octonions: Their Geometry, Arithmetic, and Symmetry*. CRC Press, 2003. 4, 7, 12, 14, 15

[CS13] John Horton Conway and Neil James Alexander Sloane. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013. 12, 39, 40

[CVL91] Peter Jephson Cameron and Jacobus Hendricus Van Lint. *Designs, graphs, codes and their links*, volume 3. Cambridge University Press, Cambridge, 1991. 33, 35

[DGS77] P Delsarte, J M Goethals, and J J Seidel. Spherical codes and designs. *Geometriae Dedicata*, 6:363–388, 1977. 25, 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. 16, 39

[EG01] Noam D Elkies and Benedict H Gross. Cubic Rings and the Exceptional Jordan Algebra. *Duke Mathematical Journal*, 109(2):383–410, 2001. 16

[FK94] Jacques Faraut and Adam Koranyi. *Analysis on Symmetric Cones*. Clarendon Press, 1994. 5, 16, 17, 18, 19, 21

[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. Birkhäuser, Boston, 2000. 18

[Hog82] Stuart G. Hoggar. t-Designs in projective spaces. *European Journal of Combinatorics*, 3(3):233–254, 1982. 25, 26, 31, 36, 37

[Hog92] Stuart G. Hoggar. t-Designs with general angle set. *European Journal of Combinatorics*, 13(4):257–271, 1992. 27, 30, 31, 32, 33

[Lyu09] Yu. I. Lyubich. On tight projective designs. *Designs, Codes and Cryptography*, 51(1):21–31, April 2009. 36

[McC04] Kevin McCrimmon. *A Taste of Jordan Algebras*. Universitext. Springer-Verlag, New York, 2004. 16, 20, 21

[Nas22] Benjamin Nasmith. Octonions and the two strictly projective tight 5-designs. *Algebraic Combinatorics*, 5(3):401–411, 2022. 39

[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, 25, 39

[SV00] Tonny A. Springer and Ferdinand D. Veldkamp. *Octonions, Jordan Algebras and Exceptional Groups*. Springer Monographs in Mathematics. Springer-Verlag, Berlin Heidelberg, 2000. 6, 7

[Wil09a] Robert A. Wilson. *The Finite Simple Groups*. Graduate Texts in Mathematics. Springer-Verlag, London, 2009. 12, 13, 20, 40

[Wil09b] Robert A. Wilson. Octonions and the Leech lattice. *Journal of Algebra*, 322(6):2186–2190, September 2009. 4, 39

[Wil11] Robert A. Wilson. Conway's group and octonions. *Journal of Group Theory*, 14(1):1–8, January 2011. 39

Index