Magma Code Instructions

Colin Ingalls, Vishal Bhatoy

July 2025

1 Introduction

We provide instructions to work with 4-dimensional Koszul Artin-Schelter regular algebras in the computer algebra system `Magma`. All algebras are listed in the file `Algebras.m`. Building on this, we present `Magma` code to compute the **superpotential** and to calculate **discrete invariants** associated with each algebra.

2 Creating Algebras

We define graded algebras in `Magma` by specifying a set of generators and homogeneous relations in the free algebra over a given base field. Let K be a base field, and consider generators $\{x_1, \dots, x_n\}$, each of degree 1, along with a set of homogeneous relations $\{r_1, \dots, r_m\}$. Mathematically, such an algebra is given by:

$$A = \frac{K\langle x_1, \dots, x_n\rangle}{(r_1, \dots, r_m)},$$

where $K\langle x_1, \dots, x_n\rangle$ denotes the free algebra over K, and the ideal of relations is generated by the r_i.

In `Magma`, the construction proceeds by first defining the base field and free algebra:

```
// Define base field
K := <your field>;

// Free algebra on n generators
F<x1, x2, ... , xn> := FreeAlgebra(K, n);
Gens := [x1, x2, ... , xn];

// Then define the list of relations:
Rels := [r1, r2, ..., rm];
```

Let us now consider a specific example: the algebra M. We work over the rational function field $K = \mathbb{Q}(h, f)$, and define a free algebra on generators x, y, z, w, each of degree 1. The algebra M is defined by the following six quadratic relations:

```
// Define the base field and a function field with parameters h and f
QQ := RationalField();
K<h,f> := FunctionField(QQ, 2);

// Define the free algebra with generators x, y, z, w
F<x,y,z,w> := FreeAlgebra(K,4);
Gens := [x,y,z,w];

// Define the six quadratic relations
Rels := [x*w + w*x,
    z*y + y*z,
    y*w + h*(-x*y - w*z ),
    y*x + h*(-f*w*y + x*z ),
    z*w + h*(-w*y + x*z ),
    z*x + h*(x*y + f*w*z ) ];
```

3 `Algebras.m` File

The file `Algebras.m` contains the defining relations for all 4-dimensional Artin-Schelter regular algebras considered in this project. These algebras have been sourced from the literature and represent several known families of Koszul algebras with quadratic relations.

Each algebra is specified by:

- a base field K, which may be a rational function field or number field depending on parameters;
- a free algebra $F = K\langle x, y, z, w\rangle$ on four generators of degree 1;
- a list of six quadratic relations that define the algebra.

For each algebra, the Magma code stores the tuple `<K, F, Rels, Gens>` in a named variable such as `AlgebraA`, `AlgebraB`, and so on. These variables can then be used in further computations (e.g., superpotential and invariants). Parameters like h, f, or p are used symbolically, with different conditions on these parameters depending on the algebra. Some examples are given below:

```
// Algebra A
<Univariate rational function field over Rational Field
Variables: h, Free associative algebra of rank 4 over Univariate rational function
    field over Rational Field
Order: Non-commutative Graded Lexicographical
Variables: x, y, z, w, [
-x*w + w*x,
-y^2 - y*z + z*y,
y*w - h*w*y,
-h*x*y + y*x - h*w*z,
z*w - h*w*z,
2*h*x*y - h*x*z + z*x + h*w*z
], [
x,
y,
z,
w
]>

// Algbera B
<Univariate rational function field over N
Variables: h, Free associative algebra of rank 4 over Univariate rational function
    field over N
Order: Non-commutative Graded Lexicographical
Variables: x, y, z, w, [
x*w - ii*w*x,
-ii*y*z + z*y,
-h*x*z + y*w,
y*x - h*w*z,
h*x*y + z*w,
z*x - h*w*y
], [
x,
y,
z,
w
]>
```

4 `Superpotential.m` File

The `Superpotential.m` code takes as input a basis for the vector space V along with the space of quadratic relations $R \subseteq V^{\otimes 2}$. It then constructs subspaces $S_0, S_1, S_2 \subseteq V^{\otimes 4}$ through tensor operations, and computes their intersection

$$A_4^{!*} = S_0 \cap S_1 \cap S_2,$$

which is known to be one-dimensional. From this intersection, the code extracts a generator $w \in V^{\otimes 4}$, representing the superpotential, unique up to a scalar multiple.

```
// Define the base field and a function field with parameters h and f
QQ := RationalField();
K<h,f> := FunctionField(QQ, 2);

// Define the free algebra with generators x, y, z, w
F<x,y,z,w> := FreeAlgebra(K,4);
Gens := [x,y,z,w];

// Define the six quadratic relations
Rels := [x*w + w*x,
    z*y + y*z,
    y*w + h*(-x*y - w*z ),
    y*x + h*(-f*w*y + x*z ),
    z*w + h*(-w*y + x*z ),
    z*x + h*(x*y + f*w*z ) ];

// Generate monomials of degrees 2, 3, and 4 in the free algebra
m2 := [B*A: A in Gens, B in Gens];
m3 := [B*A: B in m2, A in Gens];
m4 := [B*A: B in m3, A in Gens];

Construct the subspace R tensor V tensor V.
RA1A1 := [i*j*k: i in Rels, j in Gens, k in Gens];
RtensorA1tensorA1 := Matrix([[MonomialCoefficient(l,k): k in m4] : l in RA1A1]);

// Construct the subspace V tensor V tensor R
A1A1R := [i*j*k: i in Gens, j in Gens, k in Rels];
A1tensorA1tensorR := Matrix([[MonomialCoefficient(l,k): k in m4] : l in A1A1R]);

// Construct the subspace V tensor R tensor V
A1RA1 := [i*j*k: i in Gens, j in Rels, k in Gens];
A1tensorRtensorA1 := Matrix([[MonomialCoefficient(l,k): k in m4] : l in A1RA1]);

// Compute the intersection
spb := Basis(Image(RtensorA1tensorA1) meet Image(A1tensorA1tensorR) meet Image(
    A1tensorRtensorA1));

// Calculate superpotential
wsup := &+[sp[i]*m4[i]: i in [1..256]];
MC := Vector([MonomialCoefficient(wsup,k): k in m4]);
wsup;
```

Upon completing the preceding computations for algebra M, we derive the following expression for the superpotential:

$$\begin{aligned}\texttt{wsup} := &(h^2f - h^2)([xyxz]^- + [wyxy]^- + [wzxz]^-)\\ &+ (hf - h)([xyzw]^- + [xzyw]^- + [wzyx]^- + [wyzx]^-)\\ &+ (-h^3f^2 + 2h^3f - h^3)([xw]^+[yz]^+)\\ &- 1/h([xw]^+[yz]^+) + f[zwyw]^-\\ &+ (h^2f^2 - h^2f)[wywz]^- + [zwzx]^- + [ywyx]^- + [zxyx]^-\end{aligned}$$

where $[abcd]^- = abcd - cdab$ and $[ab]^+ = ab - ba$.

5 Idempotents.m File

After obtaining the superpotential, we decompose the tensor space $V^{\otimes 4}$ under the action of the symmetric group S_4. This allows us to extract the component of the superpotential corresponding to each irreducible representation indexed by a partition $\lambda \vdash 4$.
The `Idempotents.m` file constructs all 24 elements of S_4 as permutation matrices acting on $V^{\otimes 4}$, and computes the idempotents $e_\lambda \in K[S_4]$ for each partition λ. These idempotents are used to project the superpotential onto its λ-isotypic components.

```
MC := Vector([MonomialCoefficient(wsup,k): k in m4]);
    A12 := Matrix(K, 16,16, [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
        0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,
    0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
        0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,
        0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]);

I16:= IdentityMatrix(K, 16);
p34:= KroneckerProduct(A12,I16);
p12:= KroneckerProduct(I16,A12);
I4 := Matrix(K, 4,4, [1,0,0,0, 0,1,0,0,  0,0,1,0, 0,0,0,1]);
KP:= KroneckerProduct(I4,A12);
p13:= KroneckerProduct(KP,I4);
p123:= p13*p12;
p1234:= p123*p34;
Z:= p34*p12;
p23:= p123*p13;
p234:= p12*p1234;
p243:= p34*p23;
p24:= p23*p243;
p1243:= p243*p13;
p124:= p34*p1234;
p132:= p12*p13;
p1342:= p123*p1234;
p134:= p23*p1234;
T:= p24*p13;
p1324:= p124*p13;
p1432:= Z*p13;
p142:= p134*T;
p143:= p12*p1432;
p14:= p123*p1432;
p1423:= p34*T;
```

```
Q:= p14*p23;
e := KroneckerProduct(I16, I16);

// Corresponding to partition (4)
eV400 := (1/24) * (e + p12 + p13 + p14 + p23 + p24 + p34 + p123 + p132 + p124 +
    p142 + p134 + p143 + p234 + p243 + p1234 + p1243 + p1342 +  p1324 + p1423 +
    p1432 + Z + T + Q);
eV400w := MC * eV400;

// Corresponding to partition (1,1,1,1)
eV000 := (1/24) * (e + (-1)*(p12 + p13 + p14 + p23 + p24 + p34) + (1)*(p123 + p132
     + p124 + p142 + p134 + p143 + p234 + p243)  + (-1)*(p1234 + p1243 + p1342 +
    p1324 + p1423 + p1432)  + (1)*(Z + T + Q));
eV000w := MC * eV000;

// Corresponding to partition (3,1)
eV210:= (3/24) * (3*e + (1)*(p12 + p13 + p14 + p23 + p24 + p34) + (0)*(p123 + p132
     + p124 + p142 + p134 + p143 + p234 + p243)  + (-1)*(p1234 + p1243 + p1342 +
    p1324 + p1423 + p1432)  + (-1)*(Z + T + Q));
eV210w := MC * eV210;

// Corresponding to partition (2,1,1)
eV101:= (3/24) * (3*e + (-1)*(p12 + p13 + p14 + p23 + p24 + p34) + (0)*(p123 +
    p132 + p124 + p142 + p134 + p143 + p234 + p243)  + (1)*(p1234 + p1243 + p1342 +
      p1324 + p1423 + p1432)  + (-1)*(Z + T + Q));
eV101w:= MC*eV101;

// Corresponding to partition (2,2)
eV020:= (2/24) * (2*e + (0)*(p12 + p13 + p14 + p23 + p24 + p34) + (-1)*(p123 +
    p132 + p124 + p142 + p134 + p143 + p234 + p243)  + (0)*(p1234 + p1243 + p1342 +
      p1324 + p1423 + p1432)  + (2)*(Z + T + Q));
eV020w:= MC*eV020;
```

6 Computation of Discrete Invariants

We now turn to the computation of discrete invariants using the superpotential and idempotents associated with the algebra.

6.1 Invariants for the partition $\lambda = (4)$

As $\lambda = (4)$, the partial flag variety reduces to the projective space $\mathbb{P}^3$. We begin by constructing this projective space over the rational function field $\mathbb{Q}(h, f)$, where the base field is taken as a function field in two variables. The ambient space is the projective space $\mathbb{P}^3$ over the rational function field.

```
QQ := RationalField();
K<h,f> := FunctionField(QQ, 2);
P<x,y,z,w> := ProjectiveSpace(K, 3);
X := Scheme(P, eV400w);
```

We now compute the invariants, listed below:

```
// Primary components of the scheme X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(X)];
> [ [ 2, 1 ], [ 2, 1 ], [ 2, 1 ], [ 2, 1 ] ]

// Primary components of the reduced subscheme of X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(ReducedSubscheme(X))];
```

```
> [ [ 2, 1 ], [ 2, 1 ], [ 2, 1 ], [ 2, 1 ] ]

// Primary components of the singular locus of X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(SingularSubscheme(X))];
> [ [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ] ]

// Primary components of the reduced singular locus
[[Dimension(p), Degree(p)] : p in PrimaryComponents(ReducedSubscheme(
    SingularSubscheme(X)))];
> [ [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ], [ 1, 1 ] ]
```

6.2 Invariants for the partition $\lambda = (2, 2)$

The partial flag variety corresponding to partition $(2, 2)$ reduces to the Grassmannian $\mathrm{Gr}(2, 4)$. The vector $(2 - 2, 2 - 0, 0 - 0) = (0, 2, 0)$ corresponds to the line bundle $\mathcal{O}(2)$ on $\mathrm{Gr}(2, 4)$.
The representation space decomposes as a sum involving

$$\mathtt{S2W2} = \mathtt{Sym}^2(\wedge^2\mathtt{V}) \quad \text{and} \quad \mathtt{S2S2} = \mathtt{Sym}^2(\mathtt{Sym}^2\mathtt{V})$$

The basis for V_{020} consists of the matrices `S2S2` and `S2W2`, each of rank 20, which we construct using the commutator and skew-commutator functions defined below.

```
// Function to calculate commutator
Commutator := function(a, b)
   return a*b - b*a;
end function;

// Function to calculate skew-commutator
SkewCommutator := function(a, b)
   return a*b + b*a;
end function;

OP1:= [[j,i]: i in [j+1..4], j in [1..4]];
prop1:= [[OP1[j], OP1[i]]: i in [1..6], j in [1..6] | i le j];
S2W2L0new:= [sk(comm(F.i,F.j), comm(F.k,F.l)) + sk(comm(F.l,F.i), comm(F.j,F.k)) +
     sk(comm(F.k,F.l), comm(F.i,F.j)) + sk(comm(F.j,F.k), comm(F.l,F.i)) : i in [p
    [1][1]], j in [p[1][2]], k in [p[2][1]], l in [p[2][2]], p in prop1];

OP2:= [[j,i]: i in [j..4], j in [1..4] ];
prop4:= [[OP2[j], OP2[i]]: i in [1..10], j in [1..10] | i lt j and not #(Set(OP2[i
    ]) join Set(OP2[j])) eq 2 and not #(Set(OP2[i]) meet Set(OP2[j])) eq 1 ];
S2S25:= [sk(sk(F.i,F.j), sk(F.k,F.l)) - sk(sk(F.l,F.i), sk(F.j,F.k)): i in [p
    [1][1]], j in [p[1][2]], k in [p[2][1]], l in [p[2][2]], p in prop4 ];

prop7:= [[OP2[j], OP2[i]]: i in [1..10], j in [1..10] | i lt j and #(Set(OP2[i])
    join Set(OP2[j])) eq 2 and  #(Set(OP2[i]) meet Set(OP2[j])) eq 0 ];
S2S28:= [sk(sk(F.i,F.j), sk(F.k,F.l)) - sk(sk(F.l,F.i), sk(F.j,F.k)): i in [p
    [1][1]], j in [p[1][2]], k in [p[2][1]], l in [p[2][2]], p in prop7 ];

S:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in S2W2L0new]);
X:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in S2S25]);
Y:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in S2S28]);
Y1:= VerticalJoin(X, Y);
Y2:= VerticalJoin(Y1, S);
S:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in S2W2L0new]);
reducedS:= Matrix(Basis(Image(S)));
reducedX:= Matrix(Basis(Image(X)));
```

```
// Combining S2S25 and S2S28 into one matrix S2S2
S2W2 := reducedS;
S2S2:= VerticalJoin(reducedX,Y);
```

Next, we define the **Kronecker product** and the **co-product** action for Lie algebra elements acting on tensor powers, as needed to identify highest weight vectors. The function `Kron4(a, b, c, d)` calculates the Kronecker product of the four matrices `a`, `b`, `c`, and `d`. The function `coProduct4` defines how a Lie algebra element acts on the tensor product of multiple vector spaces. Specifically, the element `g` acts on the tensor product space $V_1 \otimes V_2 \otimes V_3 \otimes V_4$ in the following way: The first term `Kron4(g, iden, iden, iden)` represents the action of `g` on the first tensor factor V_1, while the identity matrix acts on the other factors. Similarly, the second, third, and fourth terms correspond to the action of `g` on the second, third, and fourth tensor factors, respectively.

```
// Define Kronecker product
Kron4 := function(a, b, c, d)
    return KroneckerProduct(KroneckerProduct(KroneckerProduct(a, b), c), d);
end function;

// Define co-product map
coProduct4 := function(g)
    iden := ScalarMatrix(4, K!1);
    return Kron4(g, iden, iden, iden) +
           Kron4(iden, g, iden, iden) +
           Kron4(iden, iden, g, iden) +
           Kron4(iden, iden, iden, g);
end function;
```

We specify generators of the Lie algebra $\mathfrak{sl}_4$, including the Cartan elements and the root vectors denoted by e_{ij}. These generators act on tensor representations through the co-product construction.

```
// Define Cartan subalgebra
e1 := Matrix(K, 4, 4, [1,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,-1]);
e2 := DiagonalMatrix(K, 4, [1, -1, 0, 0]);
e3 := DiagonalMatrix(K, 4, [0, 1, -1, 0]);
e4 := DiagonalMatrix(K, 4, [0, 0, 1, -1]);

// Root vectors
e12 := Matrix(SparseMatrix(K, 4, 4, [<1, 2, 1>]));
e23 := Matrix(SparseMatrix(K, 4, 4, [<2, 3, 1>]));
e34 := Matrix(SparseMatrix(K, 4, 4, [<3, 4, 1>]));
e21 := Matrix(SparseMatrix(K, 4, 4, [<2, 1, 1>]));
e32 := Matrix(SparseMatrix(K, 4, 4, [<3, 2, 1>]));
e43 := Matrix(SparseMatrix(K, 4, 4, [<4, 3, 1>]));
e13 := Matrix(SparseMatrix(K, 4, 4, [<1, 3, 1>]));
e31 := Matrix(SparseMatrix(K, 4, 4, [<3, 1, 1>]));
```

We identify the highest weight vector `h0` in the image of `S2W2` by computing the intersection of nullspaces under the co-product action of operators `e12,e23,e34`. Note that `m4` is the list of all degree 4 monomials in the algebra.

```
// Find highest weight vector in S2W2
N1 := [e12, e23, e34];
Interse := &meet [NullSpace(coProduct4(x)) : x in N1];
I := Interse meet Image(S2W2);
h0vec := I.1;
h0 := &+[h0vec[i] * m4[i] : i in [1..256]];
```

Here we search for a second highest weight vector `h1` in the image of `S2S2`.

```
// Find highest weight vector in S2S2
Interse := &meet [NullSpace(coProduct4(x)) : x in N1];
Inew := Interse meet Image(S2S2);
h1vec := Inew.1;
h1 := &+[h1vec[i] * m4[i] : i in [1..256]];
```

However, `h1` is not fully contained in `L0` or `L2`, so we decompose it further. Here, `L0` and `L2` denote the isotypic components corresponding to the trivial and sign representations of the cyclic group C_4, respectively. In the trivial representation `L0`, all elements of C_4 act identically (i.e., as the identity), leaving the vector unchanged. In the sign representation `L2`, the elements of C_4 act by flipping the sign of the vector.
Let $\mathtt{e}, \mathtt{u1}, \mathtt{u2}, \mathtt{u3}$ be the elements of C_4, where `e` is the identity element and `u1`, `u2`, `u3` represent successive cyclic permutations. Using its character table, we construct projection operators onto these components:

$$\mathtt{eL0} = \tfrac{1}{4}(\mathtt{e} + \mathtt{u1} + \mathtt{u2} + \mathtt{u3}), \quad \mathtt{eL2} = \tfrac{1}{4}(\mathtt{e} - \mathtt{u1} + \mathtt{u2} - \mathtt{u3}).$$

```
// Apply projectors to the highest weight vector h1
h1L0 := Matrix(h1vec) * eL0;
h1L2 := Matrix(h1vec) * eL2;

// Reconstruct symbolic expressions
h11 := &+[Vector(h1L0)[i] * m4[i] : i in [1..256]];
h12 := &+[Vector(h1L2)[i] * m4[i] : i in [1..256]];
```

Finally, the highest weight vectors in this case are `h0`, `h11`, and `h12`. We define a list `lieAlg` of Lie algebra elements in $\mathfrak{sl}_4$, including Cartan elements and root vectors as well as their products. These will act via the co-product on the tensor space.

```
lieAlg := [e21, e32, e43, e1, e2, e3, e43*e32, e32*e21, e43*e32*e21, e23*e32,
    ScalarMatrix(K,4,1)];
```

We act on each highest weight vector with quadruple co-products of elements from `lieAlg` to generate the full representation space. Each resulting matrix has rank 20.

```
M0 := Matrix([h1L0 * coProduct4(X) * coProduct4(Y) * coProduct4(Z) * coProduct4(W)
     : X in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);
M2 := Matrix([h1L2 * coProduct4(X) * coProduct4(Y) * coProduct4(Z) * coProduct4(W)
     : X in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);
M1 := Matrix([h0vec * coProduct4(X) * coProduct4(Y) * coProduct4(Z) * coProduct4(W
    ) : X in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);
```

We apply the projection operators `eL0` and `eL2` to the vector `eV020w:= MC * eV020` , yielding `w0` and `w2` in their respective components.

```
w0 := eV020w * eL0;
w2 := eV020w * eL2;
```

We solve for the transformation matrices `S01` and `S21`.

```
S01 := Transpose(Solution(Transpose(M0), Transpose(M1)));
S21 := Transpose(Solution(Transpose(M2), Transpose(M1)));
```

The final vectors `f0` and `f2` in each representation are constructed by applying the transformations to the projected generic vector.

```
f0 := w0 * S01;
f2 := w2 * S21;
```

We now define a polynomial ring $k[\mathbf{b}]$ in six variables and a 4×4 skew-symmetric matrix `E` whose entries are linear forms in the variables.

```
PR6<[b]> := PolynomialRing(K,6);
E := Matrix(PR6, [ [0,     b[1],  b[2],  b[3]],
    [-b[1], 0,     b[4],  b[5]],
    [-b[2], -b[4], 0,     b[6]],
    [-b[3], -b[5], -b[6], 0] ]);
```

Using symmetrized combinations of matrix entries, we compute a basis `bas` for the space of quadratic forms in the entries of `E`. The Grassmannian $\mathrm{Gr}(2,4)$ parametrizes 2-dimensional subspaces of a 4-dimensional vector space V. The Grassmannian $\mathrm{Gr}(r,n)$ is a projective subvariety of the projective space $\mathbb{P}(\wedge^r k^n)$. Applying this general result to our case, the Plücker embedding realizes $\mathrm{Gr}(2,4)$ as a projective algebraic variety:

$$\mathrm{Gr}(2,4) \subseteq \mathbb{P}(\wedge^2 k^4) \cong \mathbb{P}^5$$

where k is our base field and $\wedge^2 k^4$ is the second exterior power of dimension $\binom{4}{2} = 6$.
The Plücker embedding is realized through the symmetric combinations of the matrix entries $E[i,j]$, which encode the 2-planes of the Grassmannian in $\mathbb{P}(\wedge^2 V)$.
We define `OP1` as pairs (i,j) with $1 \leq j < i \leq 4$, and `prop1` selects pairs of pairs from `OP1` with $i \leq j$.

```
bas := [E[i,j]*E[k,l] + E[l,i]*E[j,k] + E[k,l]*E[i,j] + E[j,k]*E[l,i] :
        i in [p[1][1]], j in [p[1][2]], k in [p[2][1]], l in [p[2][2]], p in prop1
            ];
prop1:= [[OP1[j], OP1[i]]: i in [1..6], j in [1..6] | i le j];
OP1:= [[j,i]: i in [j+1..4], j in [1..4]];
```

We solve the linear systems to express `f0` and `f2` as linear combinations of the basis quadratics.

```
S:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in S2W2]);
beta1 := Solution(S, f0);
beta2 := Solution(S, f2);
```

The ideal defining the variety is generated by the classical Plücker relation

$$x_{12}x_{34} - x_{13}x_{24} + x_{14}x_{23} = 0,$$

together with the two quadratic equations `quadraticbeta1` and `quadraticbeta2` derived from the representation.

```
quadraticbeta1 := &+ [beta1[i] * bas[i] : i in [1..21]];
quadraticbeta2 := &+ [beta2[i] * bas[i] : i in [1..21]];
I := ideal<PR6 | [E[1,2]*E[3,4] - E[1,3]*E[2,4] + E[1,4]*E[2,3], quadraticbeta1,
    quadraticbeta2]>;
X := Proj(PR6 / I);
```

Lastly, we compute the invariants listed below:

```
// Primary components of the scheme X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(X)];
> [ [ 3, 4 ] ]

// Primary components of the reduced subscheme of X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(ReducedSubscheme(X))];
```

```
> [ [ 3, 4 ] ]

// Primary components of the singular locus of X
[[Dimension(p), Degree(p)] : p in PrimaryComponents(SingularSubscheme(X))];
> [ [ 0, 1 ], [ 0, 1 ], [ 0, 1 ],
    [ 0, 1 ], [ 0, 1 ], [ 0, 1 ] ]

// Primary components of the reduced singular locus
[[Dimension(p), Degree(p)] : p in PrimaryComponents(ReducedSubscheme(
    SingularSubscheme(X)))];
> [ [ 0, 1 ], [ 0, 1 ], [ 0, 1 ],
    [ 0, 1 ], [ 0, 1 ], [ 0, 1 ] ]
```

6.3 Invariants for the Partition $\lambda = (2,1,1)$

For the partition $\lambda = (2,1,1)$, the partial flag variety corresponds to $\mathbb{P}^3 \times \mathbb{P}^{3*}$. We start building the basis for eV101, which consists of the components $\wedge^2(\wedge^2 V)$, $V \otimes \wedge^3(V)$, and $\wedge^3(V) \otimes V$, defined respectively as:

$$\texttt{W2W21} = \wedge^2(\wedge^2\texttt{V}), \quad \texttt{VtensorW3V1} = \texttt{V} \otimes \wedge^3(\texttt{V}), \quad \text{and} \quad \texttt{W3VtensorV1} = \wedge^3(\texttt{V}) \otimes \texttt{V}$$

```
comm:= function(x,y)
return x*y-y*x;
end function;
sk:= function(x,y)
return x*y+y*x;
end function;

// Start Basis V101

// Basis W2W2
W2W2def:= [comm(comm(F.i,F.j), comm(F.k,F.l)): i in [1..4], j in [1..4], k in
    [1..4], l in [1..4] ];
Z7:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in W2W2def]);

OP9:= [[j,i]: i in [j+1..4], j in [1..4]];
prop9:= [[OP9[j], OP9[i]]: i in [1..6], j in [1..6] | i lt j];
W2W21:= [comm(comm(F.i,F.j), comm(F.k,F.l)): i in [p[1][1]], j in [p[1][2]], k in
    [p[2][1]], l in [p[2][2]], p in prop9];
Z1:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in W2W21]);
// End Basis W2W2



// Basis V tensor Wedge^3(V)
wedge3 := function(a, b, c)
    return a*b*c - b*a*c - a*c*b - c*b*a + c*a*b + b*c*a;
end function;
VtensorW3V := [F.i * wedge3(F.j, F.k, F.l): i in [1..4], j in [1..4], k in [1..4],
     l in [1..4]];
Z8:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in VtensorW3V]);

prop10 := [[i,j,k,l]: i in [1..4], j in [1..k-1], k in [1..l-1], l in [1..4]];
VtensorW3V1 := [ F.p[1] * wedge3(F.p[2], F.p[3], F.p[4]): p in prop10 ];
Z2:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in VtensorW3V1]);
// End Basis V tensor Wedge^3(V)

// Basis Wedge^3(V) tensor V
```

```
W3VtensorV1 := [ wedge3(F.p[2], F.p[3], F.p[4]) * F.p[1]: p in prop10 ];
Z3:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in W3VtensorV1]);
// End Basis Wedge^3(V) tensor V

// Combine all matrices to check Rank
J1:= VerticalJoin(Z1,Z2);
J2:= VerticalJoin(J1,Z3);

//End Basis V101
```

We now proceed to find the highest weight vector using the functions `Kron4` and `coProduct4` as we did previously.

```
Kron4:= function(a,b,c,d);
return KroneckerProduct(KroneckerProduct(KroneckerProduct(a,b), c),d);
end function;
coProduct4:= function(g);
iden:= ScalarMatrix(4,K!1);
return Kron4( g, iden, iden, iden) + Kron4( iden, g, iden, iden) + Kron4( iden,
    iden, g, iden) + Kron4( iden, iden, iden, g);
end function;

e1 := Matrix(K, 4,4, [1,0,0,0, 0,0,0,0,  0,0,0,0, 0,0,0,-1]);
e2:= DiagonalMatrix(K, 4, [1,-1,0,0]);
e3:= DiagonalMatrix(K, 4, [0,1,-1,0]);
e4:= DiagonalMatrix(K, 4, [0,0,1,-1]);
e12 := Matrix(SparseMatrix(K,4,4, [<1,2,1>]));
e23 := Matrix(SparseMatrix(K,4,4, [<2,3,1>]));
e34 := Matrix(SparseMatrix(K,4,4, [<3,4,1>]));
e21 := Matrix(SparseMatrix(K,4,4, [<2,1,1>]));
e32:=Matrix(SparseMatrix(K,4,4,[<3,2,1>]));
e43:=Matrix(SparseMatrix(K,4,4,[<4,3,1>]));
e13:=Matrix(SparseMatrix(K,4,4,[<1,3,1>]));
e31:=Matrix(SparseMatrix(K,4,4,[<3,1,1>]));

N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z1);   // where Z1 corresponds to W2W2
s1vec:= I.1;
// first highest weight vector
s1:= &+[s1vec[i]*m4[i]:  i in [1..256]];
// s1 is in W2W2.

N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z2);      //  where Z2 corresponds to V tensor Wedge^3(V)
s2vec1:= I.1;
s2vec2:= I.2;
s2vec := I.1 + I.2;
s2 := &+[s2vec[i]*m4[i]:  i in [1..256]];

// Two highest weight vectors from V tensor Wedge^3(V)
s21:= &+[s2vec1[i]*m4[i]:  i in [1..256]];
s22:= &+[s2vec2[i]*m4[i]:  i in [1..256]];


N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z3);      //  where Z3 corresponds to Wedge^3(V) tensor V
```

```
s3vec1:= I.1;
s3vec2:= I.2;
// Two highest weight vectors from Wedge^3(V) tensor V
s31:= &+[s3vec1[i]*m4[i]:  i in [1..256]];
s32:= &+[s3vec2[i]*m4[i]:  i in [1..256]];

// We have 4 highest weight vectors now in total, s1(from W2W2), s21 & s22 (from V
     tensor Wedge^3(V)), and s31(= s21) & s32 (from Wedge^3(V) tensor V)
```

From the character table of C_4, specifically, L_1 corresponds to the character of g, which acts as i (a fourth root of unity), and L_3 corresponds to the character of g^3, which acts as $-i$ (the conjugate of i). The corresponding projection operators are:

$$\mathsf{eL1} = \tfrac{1}{4}(\mathsf{e} + i\,\mathsf{u1} - \mathsf{u2} - i\,\mathsf{u3}), \quad \mathsf{eL3} = \tfrac{1}{4}(\mathsf{e} - i\,\mathsf{u1} - \mathsf{u2} + i\,\mathsf{u3}).$$

```
u1:= Z*p13;
u2:= Z*p13*Z*p13;
u3:= Z*p13*Z*p13*Z*p13;
eL0 := (1/4)*(e + u1 + u2 + u3);
eL1 := (1/4)*(e + i*u1 - u2 - i*u3);
eL3 := (1/4)*(e - i*u1 - u2 + i*u3);
eL2 := (1/4)*(e - u1 + u2 - u3);
// where i = RootOfUnity(4)


// There are 9 possible combinations between s1,s22,s32  and L0,L1 & L3.
s1L0:= Matrix(s1vec)*eL0;
s1L1:= Matrix(s1vec)*eL1;
s1L3:= Matrix(s1vec)*eL3;

s22L0:= Matrix(s2vec2)*eL0;
s22L1:= Matrix(s2vec2)*eL1;
s22L3:= Matrix(s2vec2)*eL3;

s32L0:= Matrix(s3vec2)*eL0;
s32L1:= Matrix(s3vec2)*eL1;
s32L3:= Matrix(s3vec2)*eL3;

s10:= &+[Vector(s1L0)[i]*m4[i]:  i in [1..256]];
s11:= &+[Vector(s1L1)[i]*m4[i]:  i in [1..256]];
s13:= &+[Vector(s1L3)[i]*m4[i]:  i in [1..256]];

s220:= &+[Vector(s22L0)[i]*m4[i]:  i in [1..256]];
s221:= &+[Vector(s22L1)[i]*m4[i]:  i in [1..256]];
s223:= &+[Vector(s22L3)[i]*m4[i]:  i in [1..256]];

s320:= &+[Vector(s32L0)[i]*m4[i]:  i in [1..256]];
s321:= &+[Vector(s32L1)[i]*m4[i]:  i in [1..256]];
s323:= &+[Vector(s32L3)[i]*m4[i]:  i in [1..256]];
```

We act on each highest weight vector with quadruple co-products of elements from `lieAlg` to generate the full representation space. Each resulting matrix has rank 15.

```
lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
P2:= Matrix([s2vec2*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X in
     lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);
```

```
lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
P11:= Matrix([s1L1*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X in
    lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);

lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
P220:= Matrix([s22L0*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);

lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
P323:= Matrix([s32L3*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);

Join1 := VerticalJoin(P11, P220);
Join2 := VerticalJoin(Join1, P323);
```

We apply the projection operators `eL0`, `eL1`, and `eL3` to the vector `eV101w := MC * eV101`, resulting in `w1010`, `w1011`, and `w1013` in their respective components. The final vectors `f0`, `f1`, and `f3` are constructed by applying the transformations to the projected generic vector, using the solutions of the corresponding equations `P11`, `P220`, and `P323`. Specifically, the vectors are obtained by solving the transpose systems and multiplying them with the projected vectors.

```
w1010:= eV101w*eL0;
w1011:= eV101w*eL1;
w1013:= eV101w*eL3;

S112:= Transpose(Solution(Transpose(P11),  Transpose(P2)));
S2202:= Transpose(Solution(Transpose(P220), Transpose(P2)));
S3232:= Transpose(Solution(Transpose(P323), Transpose(P2)));

f0 := w1010*S2202;
f1 := w1011*S112;
f3 := w1013*S3232;

gamma1:= Solution(Z2, f0);
gamma2:= Solution(Z2, f1);
gamma3:= Solution(Z2, f3);
```

We begin by defining the variety $\mathbb{P}^3 \times \mathbb{P}^{3*}$, the direct product of two projective spaces over the function field K, using the `ProjectiveSpace` and `DirectProduct` functions.
Next, we constructed the flag variety `flag101` by summing the products of coordinates x_1, x_2, x_3, x_4 from the first $\mathbb{P}^3$ and x_5, x_6, x_7, x_8 from the second $\mathbb{P}^{3*}$. We then defined the `base101`, consisting of 16 terms formed by multiplying coordinates from both projective spaces.
Finally, we computed three linear combinations `z1, z2, and z3`, by summing the products of the solutions `gamma1, gamma2, gamma3` with the elements of `base101`. These combinations represent components of the variety $\mathbb{P}^3 \times \mathbb{P}^{3*}$. We define the variety `X101withoutsegrenew` as the scheme over $\mathbb{P}^3 \times \mathbb{P}^{3*}$ generated by the equations `flag101, z1, z2, z3`, and compute it as:

```
P3new:= ProjectiveSpace(K,3);
P3P3new<x1,x2,x3,x4,x5,x6,x7,x8>:= DirectProduct(P3new,P3new);
flag101:= &+[P3P3new.i*P3P3new.(i+4): i in [1..4]];
base101 := [x1*x5, x2*x5, x3*x5, x4*x5, x1*x6, x2*x6, x3*x6, x4*x6, x1*x7, x2*x7,
    x3*x7, x4*x7, x1*x8, x2*x8, x3*x8, x4*x8];
```

```
z1:= &+[gamma1[i]*base101[i] : i in [1..16]];
z2:= &+[gamma2[i]*base101[i] : i in [1..16]];
z3:= &+[gamma3[i]*base101[i] : i in [1..16]];

X101withoutsegrenew:= Scheme(P3P3new, [flag101, z1, z2, z3] );
```

Finally, we compute the invariants:

```
// Primary components of the scheme X101withoutsegrenew
[[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents(
    X101withoutsegrenew)];
> [ [ 3, 4 ], [ 3, 6 ], [ 3, 6 ], [ 3, 4 ] ]

// Primary components of the reduced subscheme of X101withoutsegrenew
[[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents(
    ReducedSubscheme(X101withoutsegrenew))];
> [ [ 3, 4 ], [ 3, 6 ], [ 3, 6 ], [ 3, 4 ] ]

// Primary components of the singular locus of X101withoutsegrenew
[[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents(
    SingularSubscheme(X101withoutsegrenew))];
> [ [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 0, 1 ], [ 0, 1 ] ]

// Primary components of the reduced singular locus of X101withoutsegrenew
[[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents(
    ReducedSubscheme(SingularSubscheme(X101withoutsegrenew)))];
> [ [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 0, 1 ], [ 0, 1 ] ]
```

6.4 Invariants for the Partition $\lambda = (3, 1)$

For the partition $\lambda = (3, 1)$, the partial flag variety corresponds to $\mathbb{P}^3 \times \mathrm{Gr}(2, 4)$. We start building the basis for V_{210}, which consists of the components `W2S22`, `Wedge2VtensorSym2V1`, and `Sym2VtensorWedge2V1`, defined respectively as

$$\texttt{W2S22} = \wedge^2(\mathrm{Sym}^2\mathtt{V});\ \texttt{Wedge2VtensorSym2V1} = \wedge^2\mathtt{V} \otimes \mathrm{Sym}^2\mathtt{V};\ \text{and } \texttt{Sym2VtensorWedge2V1} = \mathrm{Sym}^2\mathtt{V} \otimes \wedge^2\mathtt{V}$$

```
comm:= function(x,y)
return x*y-y*x;
end function;
sk:= function(x,y)
return x*y+y*x;
end function;

//Start Basis V210
// Basis W2S2

W2S21:= [comm(sk(F.i,F.j), sk(F.k,F.l)): i in [1..4], j in [1..4], k in [1..4], l
    in [1..4] ];
Z4:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in W2S21]);
OP11:= [[j,i]: i in [j..4], j in [1..4]];
prop11:= [[OP11[j], OP11[i]]: i in [1..10], j in [1..10] | i lt j];
W2S22:= [comm(sk(F.i,F.j), sk(F.k,F.l)) : i in [p[1][1]], j in [p[1][2]], k in [p
    [2][1]], l in [p[2][2]], p in prop11 ];
Z5:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in W2S22]);
// End Basis W2S2
```

```
sym2 := function(a, b)
    return  a*b + b*a ;
end function;

wedge2 := function(a, b)
    return  a*b - b*a ;
end function;


// Basis sym2Vtensorwedge2V

sym2Vtensorwedge2V := [ sym2(F.i, F.j) * wedge2(F.k, F.l): i in [1..4], j in
    [1..4], k in [1..4], l in [1..4] ];
Z19:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in sym2Vtensorwedge2V ]);

sym2Vtensorwedge2V1 := [sym2(F.i, F.j) * wedge2(F.k, F.l): i in [j..4], j in
    [1..4], k in [l+1..4], l in [1..4]];
Z20:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in sym2Vtensorwedge2V1  ]);


// Basis wedge2Vtensorsym2V

wedge2Vtensorsym2V := [  wedge2(F.k, F.l) * sym2(F.i, F.j)  : i in [1..4], j in
    [1..4], k in [1..4], l in [1..4] ];
Z21:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in wedge2Vtensorsym2V ]);


wedge2Vtensorsym2V1 := [  wedge2(F.k, F.l) * sym2(F.i, F.j)  : i in [j..4], j in
    [1..4], k in [l+1..4], l in [1..4] ];
Z22:= Matrix([[MonomialCoefficient(l,k): k in m4] : l in wedge2Vtensorsym2V1 ]);

// Combine all matrices to check Rank

J6:= VerticalJoin(Z5,Z20);
J7:= VerticalJoin(J6,Z22);
```

We now proceed to find the highest weight vector using the functions Kron4 and coProduct4 as we did previously.

```
Kron4:= function(a,b,c,d);
return KroneckerProduct(KroneckerProduct(KroneckerProduct(a,b), c),d);
end function;
coProduct4:= function(e);
iden:= ScalarMatrix(4,K!1);
return Kron4( e, iden, iden, iden) + Kron4( iden, e, iden, iden) + Kron4( iden,
   iden, e, iden) + Kron4( iden, iden, iden, e);
end function;

e1 := Matrix(K, 4,4, [1,0,0,0, 0,0,0,0,  0,0,0,0, 0,0,0,-1]);
e2:= DiagonalMatrix(K, 4, [1,-1,0,0]);
e3:= DiagonalMatrix(K, 4, [0,1,-1,0]);
e4:= DiagonalMatrix(K, 4, [0,0,1,-1]);
e12 := Matrix(SparseMatrix(K,4,4, [<1,2,1>]));
e23 := Matrix(SparseMatrix(K,4,4, [<2,3,1>]));
e34 := Matrix(SparseMatrix(K,4,4, [<3,4,1>]));
e21 := Matrix(SparseMatrix(K,4,4, [<2,1,1>]));
e32:=Matrix(SparseMatrix(K,4,4,[<3,2,1>]));
e43:=Matrix(SparseMatrix(K,4,4,[<4,3,1>]));
```

```
e13:=Matrix(SparseMatrix(K,4,4,[<1,3,1>]));
e31:=Matrix(SparseMatrix(K,4,4,[<3,1,1>]));


N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z5);   // Z5 corresponds to W2S2
u1vec:= I.1;
// first highest weight vector
u1:= &+[u1vec[i]*m4[i]:  i in [1..256]];


N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z20);   // Z20 corresponds to sym2Vtensorwedge2V
u21vec:= I.1;
// first highest weight vector
u21:= &+[u21vec[i]*m4[i]:  i in [1..256]];
u21;


N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z20);   // Z20 corresponds to sym2Vtensorwedge2V
u22vec:= I.2;
// first highest weight vector
u22:= &+[u22vec[i]*m4[i]:  i in [1..256]];



N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z22);   // Z22 corresponds to wedge2Vtensorsym2V
u31vec:= I.1;
// first highest weight vector
u31:= &+[u31vec[i]*m4[i]:  i in [1..256]];


N1:= [e12,e23,e34];
Interse:= &meet [NullSpace((coProduct4(x))) : x in N1] ;
I:= Interse meet Image(Z22);   // Z22 corresponds to wedge2Vtensorsym2V
u32vec:= I.2;
// first highest weight vector
u32:= &+[u32vec[i]*m4[i]:  i in [1..256]];
```

We now use the previously defined idempotents `eL0`, `eL1`, `eL2`, `eL3` which correspond to the irreducible representations of the cyclic group C_4. Applying each idempotent `eL1`, `eL2`, and `eL3` to the structured vectors `u1vec`, `u21vec`, `u22vec`, `u31vec`, and `u32vec` produces all linear combinations lying in the corresponding isotypic components associated to the irreducible representations of C_4.

```
u1:= Z*p13;
u2:= Z*p13*Z*p13;
u3:= Z*p13*Z*p13*Z*p13;
eL0 := (1/4)*(e + u1 + u2 + u3);
eL1 := (1/4)*(e + i*u1 - u2 - i*u3);
eL3 := (1/4)*(e - i*u1 - u2 + i*u3);
eL2 := (1/4)*(e - u1 + u2 - u3);
```

```
// where i = RootOfUnity(4)

u1L1:= Matrix(u1vec)*eL1;
u1L2:= Matrix(u1vec)*eL2;
u1L3:= Matrix(u1vec)*eL3;

u21L1:= Matrix(u21vec)*eL1;
u21L2:= Matrix(u21vec)*eL2;
u21L3:= Matrix(u21vec)*eL3;


u22L1:= Matrix(u22vec)*eL1;
u22L2:= Matrix(u22vec)*eL2;
u22L3:= Matrix(u22vec)*eL3;

u31L1:= Matrix(u31vec)*eL1;
u31L2:= Matrix(u31vec)*eL2;
u31L3:= Matrix(u31vec)*eL3;

u32L1:= Matrix(u32vec)*eL1;
u32L2:= Matrix(u32vec)*eL2;
u32L3:= Matrix(u32vec)*eL3;

u11:= &+[Vector(u1L1)[i]*m4[i]:  i in [1..256]];
u12:= &+[Vector(u1L2)[i]*m4[i]:  i in [1..256]];
u13:= &+[Vector(u1L3)[i]*m4[i]:  i in [1..256]];


u211:= &+[Vector(u21L1)[i]*m4[i]:  i in [1..256]];
u212:= &+[Vector(u21L2)[i]*m4[i]:  i in [1..256]];
u213:= &+[Vector(u21L3)[i]*m4[i]:  i in [1..256]];


u221:= &+[Vector(u22L1)[i]*m4[i]:  i in [1..256]];
u222:= &+[Vector(u22L2)[i]*m4[i]:  i in [1..256]];
u223:= &+[Vector(u22L3)[i]*m4[i]:  i in [1..256]];


u311:= &+[Vector(u31L1)[i]*m4[i]:  i in [1..256]];
u312:= &+[Vector(u31L2)[i]*m4[i]:  i in [1..256]];
u313:= &+[Vector(u31L3)[i]*m4[i]:  i in [1..256]];

u321:= &+[Vector(u32L1)[i]*m4[i]:  i in [1..256]];
u322:= &+[Vector(u32L2)[i]*m4[i]:  i in [1..256]];
u323:= &+[Vector(u32L3)[i]*m4[i]:  i in [1..256]];
```

We act on each highest weight vector with quadruple co-products of elements from `lieAlg` to generate the full representation space. Each resulting matrix has rank 45.

```
lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S11:= Matrix([u1L1*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X in
    lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);


lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S21:= Matrix([u21vec*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);


lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S22:= Matrix([u22vec*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);



lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S31:= Matrix([u31vec*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);


lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S32:= Matrix([u32vec*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);



lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S222:= Matrix([u22L2*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);



lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
S323:= Matrix([u32L3*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
    in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);


Join8 := VerticalJoin(S11, S222);
Join9 := VerticalJoin(Join8, S323);


lieAlg:=[e21,e32,e43,e1,e2,e3,e43*e32,e32*e21,e43*e32*e21,e23*e32,ScalarMatrix(K
    ,4,1)];
SV210:= Matrix([u22vec*coProduct4(X)*coProduct4(Y)*coProduct4(Z)*coProduct4(W) : X
     in lieAlg, Y in lieAlg, Z in lieAlg, W in lieAlg]);
```

We apply the projection operators eL1, eL2, and eL3 to the vector eV210w := MC * eV210, resulting in w2101, w2102, and w2103 in their respective components. The final vectors g1, g2, and g3 are constructed by applying the transformations to the projected generic vector, using the solutions of the corresponding equations S11, S222, and S323. Specifically, the vectors are obtained by solving the transpose systems and multiplying them with the projected vectors.

```
w2101:= eV210w*eL1;
w2102:= eV210w*eL2;
w2103:= eV210w*eL3;



S11210:= Transpose( Solution(  Transpose(S11),  Transpose(SV210) ) );
S222210:= Transpose( Solution(  Transpose(S222),  Transpose(SV210) ) );
```

```
S323210:= Transpose( Solution(  Transpose(S323),  Transpose(SV210) ) );


g1  := w2101* S11210;
g2  := w2102* S222210;
g3  := w2103* S323210;


delta1:= Solution(Z20, g1);
delta2:= Solution(Z20, g2);
delta3:= Solution(Z20, g3);
```

We begin by defining the variety $\mathbb{P}^3 \times \mathrm{Gr}(2,4)$ through a sequence of steps. First, we define the projective space $\mathbb{P}^3$ over the field K with coordinates p_1, p_2, p_3, p_4, and then create a polynomial ring `PR6V210` with six variables $c_{12}, c_{13}, c_{14}, c_{23}, c_{24}, c_{34}$. This polynomial ring is used to define the projective space $\mathbb{P}^5$. The direct product of $\mathbb{P}^3$ and $\mathbb{P}^5$ is then taken, resulting in the variety `P3P5`, with the corresponding coordinate ring `P10V210` consisting of the coordinates $p_1, p_2, p_3, p_4, c_{12}, c_{13}, c_{14}, c_{23}, c_{24}, c_{34}$.
Next, the matrix `E210` is constructed using the c_{ij} variables, and from this we define the equation `Eq24`, which encodes the Plücker relations for the Grassmannian $\mathrm{Gr}(2,4)$. These relations cut out $\mathrm{Gr}(2,4)$ as a subvariety of $\mathbb{P}^5$.
In addition, the equation `Eqflag` imposes a flag condition, relating the projective coordinates p_i to the c_{ij} variables. The basis `bas210new` consists of 60 monomials formed by products of p_i with c_{jk}, providing a coordinate basis for the ambient space. Finally, the elements `delta11, delta12, delta13` are defined as linear combinations of the basis elements with coefficients `delta1, delta2, delta3`, and are used in subsequent computations involving the geometry of $\mathbb{P}^3 \times \mathrm{Gr}(2,4)$.

```
// Without Segre Embedding
// Defining Variety P^3 x Gr(2,4)

P3<p1,p2,p3,p4> := ProjectiveSpace(K,3);
PR6V210<c12, c13, c14, c23, c24, c34> := PolynomialRing(K, 6);
P5:= Proj(PR6V210 );

P3P5:= DirectProduct(P3,P5);
P10V210<p1,p2,p3,p4,c12,c13,c14,c23,c24,c34> := CoordinateRing(P3P5);


E210:= Matrix(P10V210, [[0,c12,c13,c14], [-c12,0,c23,c24], [-c13, -c23,0, c34], [-
    c14, -c24, -c34, 0 ]] );
Eq24 := E210[1,2]*E210[3,4] - E210[1,3]*E210[2,4] + E210[1,4]*E210[2,3];

Eqflag:= [p1*c23 - p2*c13 + p3*c12, p1*c24 - p2*c14 + p4*c12, p1*c34 - p3*c14 + p4
    *c13, p2*c34 - p3*c24 + p4*c23 ];

bas210new:= [p1*p1*c12, p2*p1*c12, p3*p1*c12, p4*p1*c12, p2*p2*c12, p3*p2*c12, p4*
    p2*c12, p3*p3*c12, p4*p3*c12, p4*p4*c12,
p1*p1*c13, p2*p1*c13, p3*p1*c13, p4*p1*c13, p2*p2*c13, p3*p2*c13, p4*p2*c13, p3*p3
    *c13, p4*p3*c13, p4*p4*c13,
p1*p1*c14, p2*p1*c14, p3*p1*c14, p4*p1*c14, p2*p2*c14, p3*p2*c14, p4*p2*c14, p3*p3
    *c14, p4*p3*c14, p4*p4*c14,
p1*p1*c23, p2*p1*c23, p3*p1*c23, p4*p1*c23, p2*p2*c23, p3*p2*c23, p4*p2*c23, p3*p3
    *c23, p4*p3*c23, p4*p4*c23,
p1*p1*c24, p2*p1*c24, p3*p1*c24, p4*p1*c24, p2*p2*c24, p3*p2*c24, p4*p2*c24, p3*p3
    *c24, p4*p3*c24, p4*p4*c24,
p1*p1*c34, p2*p1*c34, p3*p1*c34, p4*p1*c34, p2*p2*c34, p3*p2*c34, p4*p2*c34, p3*p3
    *c34, p4*p3*c34, p4*p4*c34 ];
```

```
delta11:= &+[delta1[i]*bas210new[i] : i in [1..60]];
delta12:= &+[delta2[i]*bas210new[i] : i in [1..60]];
delta13:= &+[delta3[i]*bas210new[i] : i in [1..60]];
```

Finally, we define the variety `X210withoutsegre` as the scheme over $\mathbb{P}^3 \times \mathrm{Gr}(2, 4)$ generated by the equations `Eqflag, Eq24, delta11, delta12, delta13`, and compute it as:

```
X210withoutsegre:= Scheme( P3P5, Eqflag cat [Eq24, delta11, delta12, delta13] );
X210withoutsegre;

// Due to computational complexity and the non-termination of code in Magma,
    certain entries could not be computed and are marked as *****.

// Primary components of the scheme X210withoutsegre
DimDegPCX210:= [[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents
    (X210withoutsegre)];
> *****

// Primary components of the reduced subscheme of X210withoutsegre
DimDegPCRX210:= [[Dimension(p), Degree(SegreEmbedding(p))] : p in
    PrimaryComponents(ReducedSubscheme(X210withoutsegre))];
> [ [ 3, 67 ], [ 3, 5 ], [ 3, 5 ] ]

// Primary components of the singular locus of X210withoutsegre
[[Dimension(p), Degree(SegreEmbedding(p))] : p in PrimaryComponents(
    SingularSubscheme(X210withoutsegre))];
> *****

// Primary components of the reduced singular locus of X210withoutsegre
DimDegPCRSX210:= [[Dimension(p), Degree(SegreEmbedding(p))] : p in
    PrimaryComponents(ReducedSubscheme(SingularSubscheme(X210withoutsegre)))];
> [[ 2, 2 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ], [ 2, 3 ],
  [ 2, 1 ], [ 2, 1 ], [ 2, 1 ], [ 2, 1 ], [ 2, 1 ], [ 2, 1 ]]
```

7 Absolutely Irreducible Components

When working with schemes defined over a field K, the decomposition into primary components over K sometimes does not reflect the true geometric structure of the scheme. In particular, these components may fail to capture irreducibility over the algebraic closure $\overline{K}$, leading to incorrect conclusions about the scheme's irreducible components. To address this, we consider base extensions to finite Galois extensions of K, where the decomposition into primary components corresponds to the absolutely irreducible components.

We intersect a primary component `Y1` with a randomly chosen hyperplane to obtain a curve C, and then use `IsAbsolutelyIrreducible(C)` to determine whether C is irreducible over $\overline{K}$.
The command `FieldOfGeometricIrreducibility(C)` returns the minimal field extension over which the curve becomes geometrically irreducible, giving further insight into the behavior of the component under base change. This process can be repeated for all primary components of X to verify which are absolutely irreducible.

```
pc := PrimaryComponents(X);
Y1 := pc[1]; // First primary component
C := Curve(Intersection(Y1, Scheme(P1, [49*x - 17*y + 6*z])));
FieldOfGeometricIrreducibility(C);
IsAbsolutelyIrreducible(C);
```