NB.04.F7

July 23, 2024

1 Proposition

The generic cubic of the family of cubics satisfying

$$\sigma(P_1, P_2) = \sigma(P_1, P_4) = 0 \quad \text{and} \quad s_{22} = s_{44} = 0$$

has seven eigenpoints with the alignments:

$$(P_1, P_2, P_3),\ (P_1, P_4, P_5),\ (P_1, P_6, P_7)$$

Among these points we have the relation $\langle P_1 \times P_6, P_3 \times P_5 \rangle = 0$ (i.e., the lines $P_1 \vee P_6$ and $P_3 \vee P_5$ are orthogonal). In the family there is a sub-family of cubics whose eigenpoints have the following alignments:

$$(P_1, P_2, P_3),\ (P_1, P_4, P_5),\ (P_1, P_6, P_7),\ (P_2, P_4, P_6).$$

In this case the points P_6 and P_7 are given by the formulas:

$$P_6 = s_{15}s_{34}\, P_2 + s_{13}s_{25}\, P_4, \quad P_7 = s_{15}(s_{26}s_{46} + s_{24}s_{66})\, P_1 + s_{11}s_{24}s_{56}\, P_6$$

and a sub-family whose eigenpoints have the following alignments:

$$(P_1, P_2, P_3),\ (P_1, P_4, P_5),\ (P_1, P_6, P_7),\ (P_2, P_5, P_6),\ (P_3, P_4, P_6),\ (P_3, P_5, P_7)$$

In this case, the point~P_6 (given, for instance, by the formula $P_6 = s_{15}\, P_3 + s_{13}\, P_5$) is obtained as the intersection $(P_2 \vee P_5) \cap (P_3 \vee P_4)$ and consequently $P_7 = (P_1 \vee P_6) \cap (P_3 \vee P_5)$.

No other collinearities among the eigenpoints are possible.

[1]:
```
load("basic_functions.sage")
```

[2]:
```
P1 = vector(S, (1, 0, 0))
P2 = vector(S, (0, ii, 1))
P4 = vector(S, (0, -ii, 1))
P3 = u1*P1 + u2*P2
P5 = v1*P1 + v2*P4
```

a remark on δ_1 and δ_2: $\delta_1(P_1, P_2, P_4)$ is not zero, while $\delta_2(P_1, P_2, P_3, P_4, P_5)$ is 0.

[3]:
```
assert(delta1(P1, P2, P4) != 0)
assert(delta2(P1, P2, P3, P4, P5) == 0)
```

In this configuration, the line $P_1 \vee P_2$ is tangent to the isotropic conic in P_2 and the line $P_1 \vee P_4$ is tangent to the isotropic conic in P_4 (for all P_3 and P_5)

[4]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
assert(M.rank() == 8)
```

The next computation requires about 90 seconds:

[5]:
```
ttA = cputime()
m8 = M.minors(8)
print(cputime()-ttA)
```

```
52.118775
```

The linear system $\Lambda(M)$ is the same of the linear system $\Lambda(M_e)$, where M_e is the row echelon form of M

In the matrix M_e the rows 8, 9, 10, 11, 14 are zero, the row 7 is a multiple of the row 6 and the row 13 is a multiple of the row 12.

[6]:
```
Me = M.echelon_form()
assert(Me[8] == vector(S, [0 for _ in range(10)]))
assert(Me[9] == vector(S, [0 for _ in range(10)]))
assert(Me[10] == vector(S, [0 for _ in range(10)]))
assert(Me[11] == vector(S, [0 for _ in range(10)]))
assert(Me[14] == vector(S, [0 for _ in range(10)]))
assert(Me[7]+ii*Me[6] == vector(S, [0 for _ in range(10)]))
assert(Me[13]-ii*Me[12] == vector(S, [0 for _ in range(10)]))
```

Hence, the family of cubics which have eigenpoints $P_1, \dots, P_5$ is obtained from the matrix M1 below:

[7]:
```
M1 = Me.matrix_from_rows([0, 1, 2, 3, 4, 5, 6, 13])
```

M_1 has rank 8, as it should be:

[8]:
```
assert(M1.rank() == 8)
```

1.0.1 Now we construct the pencil of cubics obtained from M_1

first we compute two random cubics with $P_1, \dots, P_5$ eigenpoints:

[9]:
```
ff1 = M1.stack(vector(S, [0, 1, 0, 2, -1, 3, 1, 4, -2, 3])).stack(vector(S,␣
↪mon)).det()
ff2 = M1.stack(vector(S, [1, 2, 0, 1, 5, 0, 1, 3, -1, 1])).stack(vector(S,␣
↪mon)).det()
```

we erase non zero factors from the two polynomials:

[10]:
```
ff1 = S.ideal(ff1).saturation(u1*u2*v1*v2)[0].gens()[0]
ff2 = S.ideal(ff2).saturation(u1*u2*v1*v2)[0].gens()[0]
```

Now we verify that the pencil of cubics $\langle ff_1, ff_2\rangle$ is also generated by the two other simpler polynomials f_1 and f_2 below.

[11]:
```
f1 = x*(x^2+3/2*y^2+3/2*z^2)
f2 = (y + ii*z) * (y + (-ii)*z) * (y*u2*v1 + (-ii)*z*u2*v1 - y*u1*v2 +
 ↪(-ii)*z*u1*v2 + (2*ii)*x*u2*v2)
```

The computations below show that $\langle ff_1, ff_2 \rangle = \langle f_1, f_2 \rangle$ (we convert the polynomials in 10-components vectors and we verify that the two vector associated to ff_1 is a linear combination of the two vectors associated to f_1 and f_2 (same for ff_2)

[12]:
```
assert(matrix([[f1.coefficient(mm) for mm in mon], [f2.coefficient(mm) for mm
 ↪in mon]]).rank() == 2)
assert(matrix([[f1.coefficient(mm) for mm in mon], [f2.coefficient(mm) for mm
 ↪in mon], [ff1.coefficient(mm) for mm in mon]]).rank() == 2)
assert(matrix([[f1.coefficient(mm) for mm in mon], [f2.coefficient(mm) for mm
 ↪in mon], [ff2.coefficient(mm) for mm in mon]]).rank() == 2)
```

hence the pencil of cubics which have eigenpoints $P_1, \ldots, P_5$ is:

[13]:
```
cb = l1*f1+l2*f2
```

The above is the equation of ALL the cubics with eigenpoints P_1, P_2, P_3, P_4, P_5 (it depends on u_1, u_2, v_1, v_2 and l_1, l_2)

the polynomial f_1 has the lines of equation $x + iy = 0$ and $x - iy = 0$ of eigenpoints, hence f_1 does not have a finite number of eigenpoints. This means that in the study of all the cubics given by cb we can assume $l_2 \neq 0$

[14]:
```
assert(S.ideal(list(eig(f1))).subs(y = -ii*z) == 0)
assert(S.ideal(list(eig(f1))).subs(y = ii*z) == 0)
```

We compute now the iegenpoints of *cb*

[15]:
```
ej = S.ideal(list(eig(cb)))
```

we erase the known eigenpoints from ej:

[16]:
```
for pp in [P1, P2, P3, P4, P5]:
    ej = ej.saturation(S.ideal(matrix([pp, (x, y, z)]).minors(2)))[0]
```

[17]:
```
ej = ej.saturation(u1*u2*v1*v2*l2)[0]
```

[18]:
```
ej = ej.radical()
assert(ej.is_prime())
```

Now ej is a prime ideal with two generators: a polynomial rt of degree 1 in x, y, z and a polynomial of degree 2 in x, y, z. Since ej gives the remaining eigenpoints of cb, we see that they are the intersection of a line rt and a conic qn

[19]:
```
rt = y*u2*v1 + (-ii)*z*u2*v1 + y*u1*v2 + ii*z*u1*v2

qn = 12*x*y*u1*v2*l2 + (12*ii)*x*z*u1*v2*l2 + (-8*ii)*x^2*u2*v2*l2 +
 ↪(4*ii)*y^2*u2*v2*l2 + (4*ii)*z^2*u2*v2*l2 + 3*y^2*l1 + 3*z^2*l1
```

```
assert(ej == S.ideal(rt, qn).saturation(u1*u2*v1*v2)[0])
```

The line rt contains therefore the points P_6 and P_7 and it passes through P_1:

[20]:
```
assert(rt.subs(substitution(P1))== 0)
```

The line rt is orthogonal to the line $P_3 \vee P_5$, as follows from these computations:

[21]:
```
r35 = matrix([P3, P5, (x, y, z)]).det()
```

r35 and rt are orthogonal:

[22]:
```
assert(
    scalar_product(
        [r35.coefficient(xx) for xx in (x, y, z)],
        [rt.coefficient(xx) for xx in (x, y, z)]
    ).is_zero()
)
```

The points P_6 and P_7, which are the itersecton point of rt and conic, do not have coordinates in $\mathbb{Q}[i][A, B, C, u, v]$, since the ideal ej is prime.

1.1 Subcases:

We want to see if there are other alignments among the points. We have to consider three cases:

- Case 1: P_2, P_4, P_6 aligned
- Case 2: P_2, P_5, P_6 aligned
- Case 3: P_3, P_5, P_7 (or P_3, P_5, P_6) aligned

1.2 Case 1: P_2, P_4, P_6 aligned.

The line $P_2 \vee P_4$ is $r24 = x$:

[23]:
```
r24 = x
assert(matrix([P2, P4, (x, y, z)]).det() == 2*ii*x)
```

Construction of the point P_6 as intersection of the lines rt and r24:

[24]:
```
P6 = vector(S, (0, rt.coefficient(z), -rt.coefficient(y)))
```

[25]:
```
assert(rt.subs(substitution(P6)) == 0)
assert(r24.subs(substitution(P6)) == 0)
```

P_6 always exists:

[26]:
```
assert(S.ideal(list(P6)).saturation(u1*u2*v1*v2)[0] == S.ideal(1))
```

If P_6 is an eigenpoint, it must be a point of the conic qn. This condition gives that $l_1 = u_2 v_2$, $l_2 = 3/4i$:

[27]:
```
assert(qn.subs(substitution(P6)) == ((16*ii))*v2*v1*u2*u1*(u2*v2*l2 + (-3/
↪4*ii)*l1))
```

the family of cubics in which P_6 is aligned with P_2 and P_4 is therefore obtained from cb as follows:

[28]:
```
cb1 = S(cb.subs({l1: u2*v2, l2: 3/4*ii}))
```

cb1 is smooth:

[29]:
```
assert(S.ideal(list(gdn(cb1))).saturation(u1*u2*v1*v2)[0].radical() == S.
↪ideal(x, y, z))
```

Construction of P_7. It is the second intersection of rt and conic, and these two polynomials give the ideal ej: first we evaluate ej on $l_1 = u_2 v_2$, $l_2 = 3/4i$, then we saturate the ideal, we divide it by the ideal of P_6, we get the ideal pp7 which gives the coordinates of the point P_7

[30]:
```
ej1 = ej.subs({l1: u2*v2, l2: 3/4*ii})
```

[31]:
```
ej1 = ej1.saturation(u1*u2*v1*v2)[0]
```

[32]:
```
pp7 = ej1.saturation(S.ideal(matrix([P6, (x, y, z)]).minors(2)))[0]
```

[33]:
```
p7coord = matrix(
    [
        [
            pp7.gens()[0].coefficient(xx) for xx in [x, y, z]
        ],
        [
            pp7.gens()[1].coefficient(xx) for xx in [x, y, z]
        ]
    ]
).minors(2)
```

[34]:
```
P7 = vector(S, (p7coord[2], -p7coord[1], p7coord[0]))
```

P7 is a common point of rt and conic (i.e. a zero of ej1) and P_7 always exists:

[35]:
```
assert(ej1.subs(substitution(P7)) == S.ideal(0))
```

[36]:
```
assert(S.ideal(list(P7)).saturation(u1*u2*v1*v2)[0] == S.ideal(1))
```

P_6 is obtained from the formula: $P_6 = s_{11}s_{15}P_3 - 2s_{13}s_{15}P_1 + s_{11}s_{15}P_5$

[37]:
```
P6a = (
    scalar_product(P1, P1)*scalar_product(P1, P5)*P3
    - 2*scalar_product(P1, P3)*scalar_product(P1, P5)*P1
    + scalar_product(P1, P1)*scalar_product(P1, P3)*P5
)

assert(S.ideal(matrix([P6, P6a]).minors(2)) == S.ideal(0))
```

P_7 is obtained from the formula: $P_7 = s_{11}s_{15}P_3 + s_{13}s_{15}P_1 + s_{11}s_{15}P_5$

[38]:
```
P7a = (
    scalar_product(P1, P1)*scalar_product(P1, P5)*P3
    + scalar_product(P1, P3)*scalar_product(P1, P5)*P1
    + scalar_product(P1, P1)*scalar_product(P1, P3)*P5
)

assert(S.ideal(matrix([P7, P7a]).minors(2)) == S.ideal(0))
```

The eigenpoints $P_1, \dots, P_7$ have the following alignments: $[(1, 2, 3), (1, 4, 5), (1, 6, 7), (2, 4, 6)]$

[39]:
```
assert(alignments([P1, P2, P3, P4, P5, P6, P7]) == [(1, 2, 3), (1, 4, 5), (1,␣
↪6, 7), (2, 4, 6)])
```

There are no subcases in which the seven eigenpints have other alignments:

[40]:
```
lp = [P1, P2, P3, P4, P5, P6, P7]

for i in range(5):
    for j in range(i+1, 6):
        for k in range(j+1, 7):
            if not ((i+1, j+1, k+1) in [(1, 2, 3), (1, 4, 5), (1, 6, 7), (2, 4,␣
↪6)]):
                assert(S.ideal(matrix([lp[i], lp[j], lp[k]]).det()).
↪saturation(u1*u2*v1*v2)[0] == S.ideal(S.one()))
```

We have the following orthogonalities: $(P_1 \vee P_2$ ort $P_2 \vee P_4)$, $(P_1 \vee P_6$ ort $P_2 \vee P_4)$, $(P_1 \vee P_4$ ort $P_2 \vee P_4)$, $(P_1 \vee P_6$ ort $P_3 \vee P_5)$

[41]:
```
assert(scalar_product(wedge_product(P1, P2), wedge_product(P2, P4)) == 0)
assert(scalar_product(wedge_product(P1, P6), wedge_product(P2, P4)) == 0)
assert(scalar_product(wedge_product(P1, P4), wedge_product(P2, P4)) == 0)
assert(scalar_product(wedge_product(P1, P6), wedge_product(P3, P5)) == 0)
```

In particular, we have that $P_1 = P_2 \times P_4$ and the configuration is (C_5)

The formulas for P_6 and P_7 will be considered later.

This concludes case 1

[42]:
```
assert(condition_matrix([P1, P2, P3, P4, P5], S, standard = "all").rank() == 8)
```

1.3 Case 2: P_2, P_5, P_6 aligned

equation line $P_2 \vee P_5$: $r25 = yv_1 - izv_1 + 2ixv_2$

[43]:
```
r25 = matrix([P2, P5, (x, y, z)]).det()
assert(r25 == y*v1 + (-ii)*z*v1 + (2*ii)*x*v2)
```

construction of the point P_6 intersection of the lines rt and r25:

[44]:
```
slz6 = matrix(
    [
        [r25.coefficient(xx) for xx in (x, y, z)],
        [rt.coefficient(xx) for xx in (x, y, z)]
    ]
).minors(2)

P6 = vector(S, (slz6[2], -slz6[1], slz6[0]))
```

check that P_6 is $rt \cap r25$:

[45]:
```
assert(r25.subs(substitution(P6)).is_zero() and  rt.subs(substitution(P6)).
 ↪is_zero())
```

P_6 always exists:

[46]:
```
assert(S.ideal(list(P6)).saturation(u1*u2*v1*v2)[0] == S.ideal(S.one()))
```

if P_6 is an eigenpoint, it must be a point of the conic. This condiiton gives that $u_1v_1l_2 + u_2v_2l_2 - 3/4il_1 = 0$:

[47]:
```
assert(
    qn.subs(substitution(P6))
    == ((-64*ii)) * v1 * u2 * u1 * v2^3 * (u1*v1*l2 + u2*v2*l2 + (-3/4*ii)*l1)
)
```

Hence the values for l_1 and l_2 is: $l_1 : u_1v_1 + u_2v_2 : l_2 : 3/4i$. This gives a cubic cb1

[48]:
```
cb1 = S(cb.subs({l1: u1*v1+u2*v2, l2: 3/4*ii}))
```

If we assume $(2u_1v_1 + 3u_2v_2)(u_1v_1 + u_2v_2)(u_1v_1 + 3u_2v_2) \neq 0$, then cb1 is smooth:

[49]:
```
irr = S.ideal(list(gdn(cb1))).saturation(u1*u2*v1*v2)[0]
irr = irr.saturation(2*u1*v1+3*u2*v2)[0]
irr = irr.saturation(u1*v1+u2*v2)[0]
irr = irr.saturation(u1*v1+3*u2*v2)[0]
assert(irr.radical() == S.ideal(x, y, z))
```

In particular we see that the cubic cb1 is not necessarily smooth.

Now we construct the point P_7. We compute the ideal e_cb1 of the iegenpoints of cb1 and we saturate it w.r.t. the eigenpoints $P_1, \ldots, P_6$:

[50]:
```
e_cb1 = S.ideal(list(eig(cb1))).saturation(u1*u2*v1*v2)[0]

for pp in [P1, P2, P3, P4, P5, P6]:
    e_cb1 = e_cb1.saturation(S.ideal(matrix([pp, (x, y, z)]).minors(2)))[0]
```

Now the ideal e-cb1 is the ideal of the point P_7. It is given by three polynomials, but it is generated by the first two polynoials:

```
[51]: assert(len(e_cb1.gens()) == 3)
pl1, pl2 = tuple(e_cb1.gens()[:2])
assert(e_cb1 == S.ideal(pl1, pl2))
```

We use the two polynomials pl1 and pl2, which are linear in x, y, z, to construct P_7:

```
[52]: slz = matrix(
    [
        [pl1.coefficient(xx) for xx in (x, y, z)],
        [pl2.coefficient(xx) for xx in (x, y, z)]
    ]
).minors(2)

P7 = vector(S, [slz[2], -slz[1], slz[0]])
```

P_7 is an eigenpoint of cb1 and it always exists:

```
[53]: assert(eig(cb1).subs(substitution(P7))==vector(S, (0, 0, 0)))

assert(S.ideal(list(P7)).saturation(u1*u2*v1*v2)[0] == S.ideal(S.one()))
```

The seven eigenpoints have the following alignments:

$$[(1,2,3),(1,4,5),(1,6,7),(2,5,6),(3,4,6),(3,5,7)]$$

Hence we get the configuration (C_8)

```
[54]: assert(alignments([P1, P2, P3, P4, P5, P6, P7]) == [(1, 2, 3), (1, 4, 5), (1,
↪6, 7), (2, 5, 6), (3, 4, 6), (3, 5, 7)])
```

We also have the relation $P_3 = Q_3$, where Q_3 is given by the formula above (is the orthocenter):

```
[55]: Q3 = (
    wedge_product(P1, P5)*scalar_product(P1, P6)*scalar_product(P5, P6)
    - scalar_product(P1, P5)*wedge_product(P1, P6)*scalar_product(P5, P6)
    + scalar_product(P1, P5)*scalar_product(P1, P6)*wedge_product(P5, P6)
)

assert(Q3 != vector(S, (0, 0, 0)))
assert(matrix([P3, Q3]).minors(2) == [0, 0, 0])
```

and we have some orthogonalities of the lines joining the eigenpoints:

```
[56]: assert(scalar_product(wedge_product(P1, P4), wedge_product(P3, P4))==0)
assert(scalar_product(wedge_product(P1, P2), wedge_product(P2, P5))==0)
assert(scalar_product(wedge_product(P3, P5), wedge_product(P1, P6))==0)
```

1.4 Case 3: P_3, P_5, P_7 aligned

Here we call P7 the point of rt which is aligned with P_3 and P_5

equation line r35 $= P_3 \vee P_5$. It turns out to be:

$yu_2v_1 + (-i)zu_2v_1 - yu_1v_2 + (-i)zu_1v_2 + (2i)xu_2v_2$

[57]:
```
r35 = matrix([P3, P5, (x, y, z)]).det()
assert(r35 == y*u2*v1 + (-ii)*z*u2*v1 - y*u1*v2 + (-ii)*z*u1*v2 +
 ↪(2*ii)*x*u2*v2)
```

Construction of the point P_7 intersection of the lines rt and r35:

[58]:
```
slz7 = matrix(
    [
        [r35.coefficient(xx) for xx in (x, y, z)],
        [rt.coefficient(xx) for xx in (x, y, z)]
    ]
).minors(2)
P7 = vector(S, (slz7[2], -slz7[1], slz7[0]))

assert(r35.subs(substitution(P7)).is_zero() and  rt.subs(substitution(P7)).
 ↪is_zero())
```

P_7 always exists:

[59]:
```
assert(S.ideal(list(P7)).saturation(u1*u2*v1*v2)[0] == S.ideal(S.one()))
```

If P_7 is an eigenpoint, it must be a point of the conic. Hence

u1*v1*l2 + u2*v2*l2 + (-3/4*ii)*l1

indeed:

[60]:
```
assert(
    qn.subs(substitution(P7))
    == ((-64*ii)) * v1 * u1 * v2^3 * u2^3 * (u1*v1*l2 + u2*v2*l2 + (-3/4*ii)*l1)
)
```

Hence the substitution into cb is: {l1: u1*v1+u2*v2, l2: 3/4*ii}

[61]:
```
cb1bis = S(cb.subs({l1: u1*v1+u2*v2, l2: 3/4*ii}))
```

We obtain that the new cubic is the same the cubic cb1 of the previous case, so this case is already studied above.

[62]:
```
assert(cb1bis == cb1)
```

In conclusion, we have two possible configurations: (C_5) and (C_8) (and a case of two lines of eigenpoints) the line $P_3 \vee P_5$ and the line $P_1 \vee P_6 \vee P_7$ are always orthogonal.

1.5 Final computations: formulas for P_6 and P_7

We want to find the formulas for P_6 and P_7 given above (in Case 1: P_2, P_6, P_6 collinear)

This part, if doLongComputations is true, requires about 30 minutes.

We define 5 generic points such that: P_2 and P_4 are on the isotropic conic, $P_1 \vee P_2$ and $P_1 \vee P_4$ are tangent to the isotropic conic and P_3 is a point on $P_1 \vee P_2$ and P_5 is a point on $P_1 \vee P_4$ (parameteres: u1, u2, l1, l2, v1, v2, m1, m2):

[63]:
```
P1 = vector(S, (2*u1*v1 + 2*u2*v2, (2*ii)*u1*v1 + (-2*ii)*u2*v2, (2*ii)*u2*v1 +
    (2*ii)*u1*v2))
P2 = vector(S, ((-ii)*u1^2 + (-ii)*u2^2, u1^2 - u2^2, 2*u1*u2))
P3 = vector(
    S,
    (
        2*u1*v1*l1 + 2*u2*v2*l1 + (-ii)*u1^2*l2 + (-ii)*u2^2*l2,
        (2*ii)*u1*v1*l1 + (-2*ii)*u2*v2*l1 + u1^2*l2 - u2^2*l2,
        (2*ii)*u2*v1*l1 + (2*ii)*u1*v2*l1 + 2*u1*u2*l2
    )
)
P4 = vector(S, ((-ii)*v1^2 + (-ii)*v2^2, v1^2 - v2^2, 2*v1*v2))
P5 = vector(
    S,
    (
        2*u1*v1*m1 + 2*u2*v2*m1 + (-ii)*v1^2*m2 + (-ii)*v2^2*m2,
        (2*ii)*u1*v1*m1 + (-2*ii)*u2*v2*m1 + v1^2*m2 - v2^2*m2,
        (2*ii)*u2*v1*m1 + (2*ii)*u1*v2*m1 + 2*v1*v2*m2
    )
)
```

[64]:
```
assert(Ciso.subs(substitution(P2)) == 0)
assert(Ciso.subs(substitution(P4)) == 0)
assert(alignments([P1, P2, P3, P4, P5]) == [(1, 2, 3), (1, 4, 5)])
```

It holds:

$$P_1 = P_2 \times P_4$$

[65]:
```
assert(matrix([P1, wedge_product(P2, P4)]).minors(2) == [0, 0, 0])
```

We know that $\sigma(P_1, P_2) = 0$, $\sigma(P_1, P_4) = 0$ and P_2, P_4 are on Ciso, hence we have:

$$s_{12} = 0, s_{14} = 0, s_{22} = 0, s_{44} = 0, s_{23} = 0, s_{45} = 0$$

[66]:
```
assert(scalar_product(P1, P2) == 0)
assert(scalar_product(P1, P4) == 0)
assert(scalar_product(P2, P2) == 0)
assert(scalar_product(P4, P4) == 0)
assert(scalar_product(P2, P3) == 0)
assert(scalar_product(P4, P5) == 0)
```

We know that the V - configuration $P_1, \dots, P_5$ has rank 8

[67]:
```
# The following computation if executed, requires about 1' 30''
# assert(matrixEigenpoints([P1, P2, P3, P4, P5]).rank() == 8)
```

We want to see when a point P_6 on the line $P_2 \vee P_4$ is an eigenpoint. Therefore we define P_6 and we try to see when it is an eigenpoint, i.e. when $\Phi(P_1, \dots, P_6)$ has rank ≤ 9.

[68]:
```
P6 = w1*P2+w2*P4
```

As a consequence of its definition, we have that $s_{16} = 0$.

Now we construct the matrix of conditions:

[69]:
```
MM = condition_matrix([P1, P2, P3, P4, P5, P6], S, standard = "all")
```

The next computations show that P_6 is an eigenpoint iff w2*l2*m1 - w1*l1*m2 $= 0$. This result requires 25 minutes.

We select some order 10-minors of MM in such a way that the ideal they generate gives precisely the conditions for which P_6 is an eigenpoint. Remember that we can assume (u2*v1-u1*v2) != 0, since P_2 and P_4 are distinct. It turns out that all the determinants of the order 10-minors of MM can be divided by (u2*v1-u1*v2)^24.

The next block contains these computations:

[72]:
```
doLongComputations = False
```

[73]:
```
if doLongComputations:
    Lrw = [
        [0, 1, 3, 4, 6, 9, 10, 12, 15, 16],
        [0, 1, 3, 4, 6, 9, 10, 12, 15, 17],
        [0, 2, 3, 4, 6, 9, 10, 12, 15, 16],
        [0, 1, 3, 4, 7, 9, 10, 12, 15, 16],
        [0, 1, 3, 4, 6, 9, 10, 13, 15, 16],
        [0, 1, 3, 4, 6, 9, 11, 12, 15, 16],
        [0, 1, 3, 5, 6, 9, 10, 12, 15, 16],
        [1, 2, 3, 4, 7, 9, 10, 13, 15, 16],
        [0, 2, 3, 4, 6, 9, 10, 12, 15, 17],
        [0, 1, 3, 4, 7, 9, 11, 12, 15, 16],
        [0, 1, 3, 5, 6, 9, 11, 12, 15, 17],
        [0, 1, 3, 5, 6, 9, 10, 13, 15, 16],
        [0, 1, 3, 4, 7, 9, 10, 12, 15, 17],
        [0, 1, 3, 4, 6, 9, 10, 13, 15, 17],
        [1, 2, 3, 4, 7, 9, 10, 13, 15, 17],
        [0, 1, 3, 4, 6, 9, 10, 12, 16, 17],
        [0, 1, 3, 5, 6, 9, 11, 12, 16, 17],
        [0, 2, 3, 4, 6, 9, 10, 12, 16, 17]
    ]

    Jb = S.ideal(0)
    flag = 1
    for ll in Lrw:
        print(ll)
        sleep(1)
```

```
        ttA = cputime()
        Mx = MM.matrix_from_rows(ll)
        ddt = Mx.det()
        print("computed long determinant")
        sleep(1)
        ddtDiv = ddt.quo_rem((u2*v1-u1*v2)^24) ## it turns out this happens
        if ddtDiv[1] == 0:
            ff = ddtDiv[0]
        else:
            ff = ddt
            print("just in case...") ## In practise, this never happens
        print(cputime()-ttA)
        sleep(1)
        ffId = S.ideal(ff)
        ## we saturate w.r.t. conditions that are surely satisfied.
        ffId = ffId.saturation((u2*v1-u1*v2)*m1*m2*l1*l2*w1*w2)[0]
        Jb = Jb + ffId
        Jb = Jb.saturation((u2*v1-u1*v2)*m1*m2*l1*l2*w1*w2)[0]
        print("computation n. : "+str(flag)+" over "+ str(len(Lrw)))
        print("")
        flag += 1
        sleep(1)
else:
    Jb = S.ideal(w2*l2*m1 - w1*l1*m2)
```

The only condition we get is: w2*l2*m1 - w1*l1*m2 = 0

[74]:
```
assert(Jb == S.ideal(w2*l2*m1 - w1*l1*m2))
```

We construct a matrix whose determinant is the cubic which has P_1, P_2, P_3, P_4, P_5 as eigenpoints and a row of $\phi(P_6)$:

We construct P_6 with this condition:

[75]:
```
PP6 = P6.subs({w1:l2*m1, w2: l1*m2})
```

It holds:

$$P_6 = s_{11}s_{15}P_3 - 2s_{13}s_{15}P_1 + s_{11}s_{13}P_5$$

and also:

$$P_6 = s_{15}s_{34}P_2 + s_{13}s_{25}P_4$$

[76]:
```
PP6a = (
    scalar_product(P1, P1)*scalar_product(P1, P5)*P3
    - 2*scalar_product(P1, P3)*scalar_product(P1, P5)*P1
    + scalar_product(P1, P1)*scalar_product(P1, P3)*P5
)

assert(S.ideal(matrix([PP6, PP6a]).minors(2)) == S.ideal(0))
```

```
PP6b = scalar_product(P1, P5)*scalar_product(P3, P4)*P2+scalar_product(P1,␣
 ↪P3)*scalar_product(P2, P5)*P4
assert(S.ideal(matrix([PP6, PP6b]).minors(2)) == S.ideal(0))
```

We construct the cubic whose eigenpoints are $P_1, \ldots, P_6$:

[77]:
```
MM1 = condition_matrix([P1, P2, P3, P4, P5, PP6], S, standard = "all").
 ↪matrix_from_rows([0, 1, 3, 4, 6, 9, 10, 12, 16])
MM1 = MM1.stack(vector(mon))
```

[78]:
```
## the following computation requires 146 seconds:
cbc = MM1.det()
```

We can find that P_7 is given by the formula:

$$s_{11}s_{15}P_3 + s_{13}s_{15}P_1 + s_{11}s_{13}P_5$$

and also by:

$$s_{15}(s_{26}s_{46} + s_{24}s_{66})P_1 + s_{11}s_{24}s_{56}P_6$$

[79]:
```
PP7b = (
    scalar_product(P1, P5)
    * (scalar_product(P2, PP6)*scalar_product(P4, PP6)+ scalar_product(P2,␣
 ↪P4)*scalar_product(PP6, PP6))*P1
    + scalar_product(P1, P1)*scalar_product(P2, P4)*scalar_product(P5, PP6)*PP6
)

PP7 = (
    scalar_product(P1, P1)*scalar_product(P1, P5)*P3
    + scalar_product(P1, P3)*scalar_product(P1, P5)*P1
    + scalar_product(P1, P1)*scalar_product(P1, P3)*P5
)
```

PP7 and PP7b are the same point:

[80]:
```
assert(matrix([PP7, PP7b]).minors(2) == [0, 0, 0])
```

P_7 is an eigenpoint (about 40 seconds of computations):

[81]:
```
assert(S.ideal(list(eig(cbc))).subs(substitution(PP7)) == S.ideal(S.zero()))
```