NB.04.F5

July 23, 2024

1 Proposition

If five points $P_1, \dots, P_5$ in a V-configuration satisfy

$$\delta_1(P_1, P_2, P_4) = \bar{\delta}_1(P_1, P_2, P_3) = \bar{\delta}_1(P_1, P_4, P_5) = 0$$

then, in $\Lambda(\Phi(P_1, \dots, P_5))$ there is a cubic curve with 7 eigenpoints with the following three alignments:

$$(P_1, P_2, P_3)\,, \quad (P_1, P_4, P_5)\,, \quad \text{and} \quad (P_1, P_6, P_7)\,.$$

No choices of $P_1, \dots, P_5$ allow one to obtain further alignments of the 7 eigenpoints.

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

We define three points $P_1, \dots, P_5$ so that they form a V-configuration

[2]:
```
P1 = vector((1, 0, 0))
P2 = vector(S, (A2, B2, C2))
P3 = u1*P1+u2*P2
P4 = vector(S, (A4, B4, C4))
P5 = v1*P1+v2*P4
```

We want to impose $\delta_1(P_1, P_2, P_4) = 0$. We check that $\delta_1(P_1, P_2, P_4) = 0$ is $B_2B_4 + C_2C_4$:

[3]:
```
assert(delta1(P1, P2, P4) == B2*B4+C2*C4)
```

It is not possible to have $C_2 = C_4 = 0$.

If $C_2 = 0$, necessarily $B_2 \neq 0$.

If $C_2 \neq 0$ and $B_4 = 0$, we get $C_4 = 0$, but $P_4 \neq P_1$, so if $C_2 \neq 0$, we can assume $B_4 \neq 0$.

Hence, the other situation to consider is $C_2 = B_4 = 0$, in which case the points are $p_1 = (1, 0, 0)$, $p_2 = (A_2, B_2, 0)$, $p_4 = (A_4, 0, C_4)$.

1.1 We assume $C_2 \neq 0$

[4]:
```
st1 = {A4: A4*C2, B4: B4*C2, C4:-B2*B4}
p1 = P1.subs(st1)
p2 = P2.subs(st1)
p4 = P4.subs(st1)
assert(delta1(p1, p2, p4) == 0)
```

We define p_3 and p_5 according to the known formulas:

[5]:
```
p3 = (scalar_product(p1, p2)^2+scalar_product(p1, p1)*scalar_product(p2,␣
 ↪p2))*p1-2*(scalar_product(p1, p1)*scalar_product(p1, p2))*p2
p5 = (scalar_product(p1, p4)^2+scalar_product(p1, p1)*scalar_product(p4,␣
 ↪p4))*p1-2*(scalar_product(p1, p1)*scalar_product(p1, p4))*p4
```

The entries of p_5 can be divided by B_4 (which is, as said, not 0)

[6]:
```
assert(gcd(list(p5)) == B4)
```

[7]:
```
p5 = vector(S, [px.quo_rem(B4)[0] for px in list(p5)])
```

[8]:
```
assert(delta1b(p1, p2, p3) == 0)
assert(delta1b(p1, p4, p5) == 0)
## Incidentally, delta2 is also 0:
assert(delta2(p1, p2, p3, p4, p5) == 0)
```

An example shows that, in general, p_6 and p_7 are not aligned with p_1. Here is an example.

[9]:
```
ss3 = {A2:1, B2:-5, C2:-3, A4:7, B4:-5}
pp1 = p1.subs(ss3)
pp2 = p2.subs(ss3)
pp3 = p3.subs(ss3)
pp4 = p4.subs(ss3)
pp5 = p5.subs(ss3)
cb = cubic_from_matrix(
    condition_matrix(
        [pp1, pp2, pp3, pp4, pp5],
         S,
        standard="all"
    ).stack(
        matrix(
            [
                [2, 3, 4, 5, 6, 7, 8, 9, 1, 2]
            ]
        )
    )
)
```

NOW WE WANT TO SEE WHAT HAPPENS IF WE IMPOSE THE ALIGNMENT p_1, p_6, p_7.

We start with p_1, p_2, p_3, p_4, p_5 as above, such that $\delta_1(p_1, p_2, p_4)$, $\bar{\delta}_1(p_1, p_2, p_3)$ and $\bar{\delta}_1(p_1, p_4, p_5)$ are 0

The matrix $\Phi(p_1, p_2, p_3, p_4, p_5)$ has rank 8.

[10]:
```
M = condition_matrix([p1, p2, p3, p4, p5], S, standard="all")
assert(M.rank() == 8)
```

We select 8 linearly independent rows:

[11]:
```
mm = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 12, 14])
```

[12]:
```
# in general, mm has rank 8
assert(mm.rank() == 8)
```

[13]:
```
# let us see when the rank is not 8:
hj = S.ideal(mm.minors(8))
hj = hj.saturation(S.ideal(matrix([p3, p5]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p2, p3]).minors(2)))[0]
#
# it holds: hj = (1)
assert(hj == S.ideal(1))
# hence the order 8 minor mm has always rank 8.
```

The cubics of $\mathbb{P}^9$ that have $p_1, \dots, p_5$ as eigenpoints are a line L of $\mathbb{P}^9$.

We want to see if, among the points of this line, we can find some which give cubic curves with the eigenpoints p_1, p_6, p_7 collinear. Here is the way in which we procede. We fix two 10-components vectors like: $(1, 2, 5, 6, 0, 2, 3, 4, 9, 11)$ and $(-1, 3, 6, 5, 0, 1, 3, 7, 9, -5)$ and we consider the following matrices: mmA and mmB (we put an index "1" because later we shall repeat the computation):

[14]:
```
mmA_1 = mm.stack(matrix([1, 2, 5, 6, 0, 2, 3, 4, 9, 11]))
mmB_1 = mm.stack(matrix([-1, 3, 6, 5, 0, 1, 3, 7, 9, -5]))
```

mmA and mmB have rank 9 (if not, take two other points!) From mmA we get one cubic curve cbA, from mmB we get one cubic cbB and they are two points of the line L of $\mathbb{P}^9$. Both cbA and cbB have, among the eigenpoints, p_1, p_2, p_3, $_4$, p_5.

Given the generic point $P = (x : y : z)$ of $\mathbb{P}^2$, consider the three matrices obtained from mmA_1 given by * mmA plus the row $\Phi(P)_{(1)}$, * mmA plus the row $\Phi(P)_{(2)}$, and * mmA plus the row $\Phi(P)_{(3)}$.

We get three square matrices with determinant GA1, GA2, GA3 such that $p_{1,z}GA1 - p_{1,y}GA2 + p_{1,x}GA3$ splits, as a polynomial in x, y, z, into three linear factors, one corresponds to the line $p_1 \vee p_2$, the other to the line $p_1 \vee p_4$ and the third to the line passing through the two remaining eigenpoints of cbA. A simplification comes from the fact that p_1 is the point $(1 : 0 : 0)$, hence $p_{1,z}GA1 - p_{1,y}GA2 + p_{1,x}GA3$ is $GA3$. Hence we have to factorize it.

The same construction can be done for mmB and we get a polynomial GB3 which factorizes into three linear factors in x, y, z, the first two correspond again to the lines $p_1 \vee p_2$ and $p_1 \vee p_4$, the third corresponds to the line through the two remaining eigenpoints of cbB.

A point of the line L corresponds to the cubic cb $=$ w1$cbA+w2$cbB, where w1 and w2 are parameters.

The cubic cb has p_1, p_2, p_3, p_4, p_5 as eigenpoints and two other eigenpints, say p6 and p7, that are obtained from the factorization of w1$GA3+w2$GB3.

Now the explicit computations (we repeat the construction twice):

[15]:
```
GA3_1 = mmA_1.stack(matrix([phi((x, y, z), S)[2]])).det()
GB3_1 = mmB_1.stack(matrix([phi((x, y, z), S)[2]])).det()

rr3_1 = list(
    filter(
        lambda uu: w1 in uu[0].variables(),
        list(factor(w1*GA3_1+w2*GB3_1))
    )
)[0][0]
```

rr3_1 is a polynomial of degree 1 in x, y, z which gives the line passing through the eigenpoints p6 and p7 of cb (it depends of w1 and w2).

We want to see if, among the cubics cb = cb(w1, w2), it is possible to find a cubic with p1, p6, p7 aligned. Hence p1 must be a point of rr3_1, so the following polynomial must be zero:

[16]:
```
hh1 = rr3_1.subs({x:1, y:0, z:0})
```

In order to get rid of the choice of the two rows above (i.e. the choice of two points of L), we repeat the construction for two other random rows:

[17]:
```
mmA_2 = mm.stack(matrix([1, -5, 1, 2, 0, 1, -2, 1, 3, 7]))
mmB_2 = mm.stack(matrix([-1, -1, 0, 4, 0, 1, 0, 1, 0, -5]))

GA3_2 = mmA_2.stack(matrix([phi((x, y, z), S)[2]])).det()
GB3_2 = mmB_2.stack(matrix([phi((x, y, z), S)[2]])).det()

rr3_2 = list(
    filter(
        lambda uu: w1 in uu[0].variables(),
        list(factor(w1*GA3_2+w2*GB3_2))
    )
)[0][0]
```

[18]:
```
## if p1, p6, p7 are alligned, also the following polynomial must be zero
hh2 = rr3_2.subs({x:1, y:0, z:0})
```

hh1 (and hh2) are of the form w1*()+w2*(). Hence hh1 (and hh2) is zero iff w1 and w2 are chosen as solution of the equation w1*()+w2*() = 0 or if the coefficients of w1 and w2 are both zero. In the first case, we construct r3_1 and r3_2 as follows:

[23]:
```
r3_1 = (w1*GA3_1+w2*GB3_1).subs(
    {
        w1: hh1.coefficient(w2),
        w2: -hh1.coefficient(w1)
    }
).factor()[-2][0]
r3_2 = (w1*GA3_2+w2*GB3_2).subs(
    {
```

```
        w1: hh2.coefficient(w2),
        w2: -hh2.coefficient(w1)
    }
).factor()[-2][0]
```

(i.e. r3_1 and r3_2 are the line passing through p1, p6, p7. They should be equal, because they should not depend of the two points of the line L chosen. Indeed, they are equal:

[24]:
```
assert(r3_1 == r3_2)
```

Now we have to consider the case in which r3_1 and r3_2 are not defined, i.e. when the coefficients of w1 and w2 in hh1 and hh2 are all zero:

[25]:
```
HH = [hh1, hh2]
JJ = S.ideal([hh.coefficient(w1) for hh in HH] + [hh.coefficient(w2) for hh in␣
 ↪HH])
```

If we have some values of the points p1, ..., p5 such that give a zero of JJ, we have to study that case.

But JJ, after saturation, is (1):

[26]:
```
JJ = JJ.saturation(S.ideal(matrix([p1, p2]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p3, p5]).minors(2)))[0]
```

[27]:
```
assert(JJ == S.ideal(1))
```

This computation shows that there are no exceptions to consider.

On the line L of $\mathbb{P}^9$ there is a point that corresponds to a cubic which has the alignments p_1, p_2, p_3, and p_1, p_4, p_5, and p_1, p_6, p_7 among the eigenpoints.

We can determine the cubic:

[28]:
```
MM_1 = (w1*mmA_1+w2*mmB_1).subs(
    {
        w1: hh1.coefficient(w2),
        w2: -hh1.coefficient(w1)
    }
)

Mcb1 = MM_1.stack(vector(S, mon))

MM_2 = (w1*mmA_2+w2*mmB_2).subs(
    {
        w1: hh2.coefficient(w2),
        w2: -hh2.coefficient(w1)
    }
```

```
)

Mcb2 = MM_2.stack(vector(S, mon))
```

The cubic is the determinant of Mcb1 (or of Mcb2).

The following computations require, respectively, about 3000 and 4000 seconds. We omit them but we define below the cubic cb which is obtained:

[29]:
```
# ttA = cputime()
# cb1 = Mcb1.det()
# print(cputime()-ttA)
#
# ttA = cputime()
# cb2 = Mcb2.det()
# print(cputime()-ttA)
#
# assert(cb1.factor()[-1][0]  == cb2.factor()[-1][0])
```

[30]:
```
cb = (
    z^3*A2*B2^3*C2^2*A4^2 - 3*y*z^2*A2*B2^2*C2^3*A4^2 + 3*y^2*z*A2*B2*C2^4*A4^2
    - y^3*A2*C2^5*A4^2 - y^3*A2^2*B2^3*C2*A4*B4 + 3/2*x*y^2*A2*B2^4*C2*A4*B4
    + 3/2*x*z^2*A2*B2^4*C2*A4*B4 + 1/2*y^3*B2^5*C2*A4*B4 -␣
 ↪3*y^2*z*A2^2*B2^2*C2^2*A4*B4
    + 3/2*y^2*z*B2^4*C2^2*A4*B4 - 3*y*z^2*A2^2*B2*C2^3*A4*B4 +␣
 ↪3*x*y^2*A2*B2^2*C2^3*A4*B4
    + 3*x*z^2*A2*B2^2*C2^3*A4*B4 + 1/2*y^3*B2^3*C2^3*A4*B4 + 3/
 ↪2*y*z^2*B2^3*C2^3*A4*B4
    - z^3*A2^2*C2^4*A4*B4 + 3/2*y^2*z*B2^2*C2^4*A4*B4 + 1/2*z^3*B2^2*C2^4*A4*B4
    + 3/2*x*y^2*A2*C2^5*A4*B4 + 3/2*x*z^2*A2*C2^5*A4*B4 + 3/
 ↪2*y*z^2*B2*C2^5*A4*B4
    + 1/2*z^3*C2^6*A4*B4 - 1/2*z^3*A2*B2^5*B4^2 + 3/2*y*z^2*A2*B2^4*C2*B4^2
    - 3/2*y^2*z*A2*B2^3*C2^2*B4^2 - 1/2*z^3*A2*B2^3*C2^2*B4^2 + 1/
 ↪2*y^3*A2*B2^2*C2^3*B4^2
    + 3/2*y*z^2*A2*B2^2*C2^3*B4^2 - 3/2*y^2*z*A2*B2*C2^4*B4^2 + 1/
 ↪2*y^3*A2*C2^5*B4^2
)
```

We remark however that cb can be computed in 12 seconds using the fact that the rows of Mcb1 have big common factors:

[31]:
```
Ms = []
for i in range(10):
    gd = gcd([Mcb1[i,j] for j in range(10)])
    Ms.append([Mcb1[i,j].quo_rem(gd)[0] for j in range(10)])

cb_alt = matrix(Ms).det().factor()[-1][0]
```

[32]:
```
assert(cb_alt == cb)
```

An example shows that cb, in general, has the following aligned eigenpoints: p1, p2, p3 and p1, p4, p5 and p1, p6, p7. The cubic is irreducible and singular in p1.

[33]:
```
ccb = cb.subs({A2:5, B2:-3, C2:-1, A4:2, B4:-7})
```

HERE WE CONCLUDE THE FIRST PART OF THE COMPUTATION:

In case $C_2 \neq 0$, IT IS POSSIBLE TO HAVE THREE ALIGNMENTS: (1, 2, 3), (1, 4, 5), (1, 6, 7)

Now we want to see if it is possible to have more then three alignments (We continue to assume C2 != 0)

Recall that cb is our cubic.

Recall that r3_1 (= r3_2) is the line through p6 and p7.

Up to a permutation of the indices of the points, if there is another alignment among the eigenpoints, p6 must be on the line p2+p4. Hence we can find it, since is the intersection or p2+p4 and r3.

[34]:
```
r24 = matrix([p2, p4, (x, y, z)]).det()

E1 = matrix(
    [
        [r3_1.coefficient(xx) for xx in [x, y, z]],
        [r24.coefficient(xx) for xx in [x, y, z]]
    ]
).minors(2)
```

All the entries of E1 can be divided by $B_2^2B_4 + C_2^2B_4$ and we know that $B_2^2B_4 + C_2^2B_4 = 0$ implies $p_3 = p_5$, hence we divide with no problems.

[35]:
```
p6 = vector(
    S,
    (
        E1[2]/(B2^2*B4 + C2^2*B4),
        -E1[1]/(B2^2*B4 + C2^2*B4),
        E1[0]/(B2^2*B4 + C2^2*B4)
    )
)
```

Now we compute the ideal kJ of the eigenpoints of cb and we saturate it as much as possible (in particular, we saturate it w.r.t. the ideals of the points p1, p2, p3, p4, p5):

[36]:
```
kJ = S.ideal(
    matrix(
        [
            [x, y, z],
            [cb.derivative(x), cb.derivative(y), cb.derivative(z)]
        ]
    ).minors(2)
```

```
)
```

[37]:
```
kJ = kJ.saturation(B2^2*B4 + C2^2*B4)[0]
kJ = kJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
kJ = kJ.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
kJ = kJ.saturation(S.ideal(z, y))[0]  ##p1
kJ = kJ.saturation(S.ideal(p2[0]*y-p2[1]*x, p2[0]*z-p2[2]*x,␣
 ↪p2[1]*z-p2[2]*y))[0] ## p2
kJ = kJ.saturation(S.ideal(p3[0]*y-p3[1]*x, p3[0]*z-p3[2]*x,␣
 ↪p3[1]*z-p3[2]*y))[0] ## p3
kJ = kJ.saturation(S.ideal(p4[0]*y-p4[1]*x, p4[0]*z-p4[2]*x,␣
 ↪p4[1]*z-p4[2]*y))[0] ## p4
kJ = kJ.saturation(S.ideal(p5[0]*y-p5[1]*x, p5[0]*z-p5[2]*x,␣
 ↪p5[1]*z-p5[2]*y))[0] ## p5
```

After these computations, kJ is the ideal of the two reminining eigenpoints. we want that p1 defined above is an eigenpoint, so the ideal kkJ here defined must be zero:

[39]:
```
kkJ = kJ.subs({x:p6[0], y:p6[1], z:p6[2]}).radical()
```

We saturate kkJ and we get a primary decomposiiton:

[40]:
```
kkJ = kkJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p3, p5]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p2, p6]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p4, p6]).minors(2)))[0]
```

kkJ is (1), so there are no solutions:

[41]:
```
assert(kkJ == S.ideal(1))
```

CONCLUSION (for the case $C_2 \neq 0$) when the three deltas are zero, (hence rank of the matrix of the five points is 8), we have also $\delta_2 = 0$ and we have the collinearities (1, 2, 3) (1, 4, 5). We have a sub-case in which there is the further collinearity (1, 6, 7). No other collinearities among the 7 eigenpoints are possible.

1.2 We assume $C_2 = 0$

[68]:
```
p1 = vector(S, (1, 0, 0))
p2 = vector(S, (A2, B2, 0))
p4 = vector(S, (A4, 0, C4))
```

We are sure that $B_2 \neq 0$ (since $p_2 \neq p_1$) and $C_4 \neq 0$ (since $p_4 \neq p_1$)

[69]:
```
p3 = (
    (scalar_product(p1, p2)^2 + scalar_product(p1, p1)*scalar_product(p2,␣
 ↪p2))*p1
```

```
    -2*(scalar_product(p1, p1)*scalar_product(p1, p2))*p2
)
p5 = (
    (scalar_product(p1, p4)^2 + scalar_product(p1, p1)*scalar_product(p4,␣
 ↪p4))*p1
    -2*(scalar_product(p1, p1)*scalar_product(p1, p4))*p4
)
```

We redefine the points, since B_2 and C_4 are not 0.

[70]:
```
p3, p5 = p3/B2, p5/C4
```

[71]:
```
assert(delta1b(p1, p2, p3) == 0)
assert(delta1b(p1, p4, p5) == 0)
## Incidentally, delta2 is also 0:
assert(delta2(p1, p2, p3, p4, p5) == 0)
```

An example shows that, in general, p_6 and p_7 are not aligned with p_1. Here is an example.

[72]:
```
ss3 = {A2:1, B2:-5, A4:7, C4:-5}
pp1 = p1.subs(ss3)
pp2 = p2.subs(ss3)
pp3 = p3.subs(ss3)
pp4 = p4.subs(ss3)
pp5 = p5.subs(ss3)
cb = cubic_from_matrix(
    condition_matrix(
        [pp1, pp2, pp3, pp4, pp5],
         S,
        standard="all"
    ).stack(
        matrix(
            [
                [2, 3, 4, 5, 6, 7, 8, 9, 1, 2]
            ]
        )
    )
)
```

NOW WE WANT TO SEE WHAT HAPPENS IF WE IMPOSE THE ALIGNMENT p_1, p_6, p_7.

We start with p_1, p_2, p_3, p_4, p_5 as above, such that $\delta_1(p_1, p_2, p_4)$, $\bar{\delta}_1(p_1, p_2, p_3)$ and $\bar{\delta}_1(p_1, p_4, p_5)$ are 0

The matrix $\Phi(p_1, p_2, p_3, p_4, p_5)$ has rank 8.

[73]:
```
M = condition_matrix([p1, p2, p3, p4, p5], S, standard="all")
assert(M.rank() == 8)
```

We select 8 linearly independent rows:

[74]:
```
mm = M.matrix_from_rows([0, 1, 3, 4, 6, 7, 9, 10])
```

[75]:
```
# in general, mm has rank 8
assert(mm.rank() == 8)
```

[76]:
```
# let us see when it is not 8:
hj = S.ideal(mm.minors(8))
```

[77]:
```
hj = hj.saturation(S.ideal(matrix([p1, p2]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
hj = hj.saturation(S.ideal(matrix([p2, p3]).minors(2)))[0]
```

hj is (1), so mm has always rank 8.

[82]:
```
assert(hj == S.ideal(S.one()))
```

Hence the order 8 minor mm has always rank 8. As above, we construct mmA_1 and mmB_1. The construction is the same as above.

[83]:
```
mmA_1 = mm.stack(matrix([1, 2, 5, 6, 0, 2, 3, 4, 9, 11]))
mmB_1 = mm.stack(matrix([-1, 3, 6, 5, 0, 1, 3, 7, 9, -5]))
```

These two matrices have rank 9

[84]:
```
assert(mmA_1.rank() == 9)
assert(mmB_1.rank() == 9)
```

[87]:
```
GA3_1 = mmA_1.stack(matrix([phi((x, y, z), S)[2]])).det()
GB3_1 = mmB_1.stack(matrix([phi((x, y, z), S)[2]])).det()
```

[88]:
```
rr3_1 = list(
    filter(
        lambda uu: w1 in uu[0].variables(),
        list(factor(w1*GA3_1+w2*GB3_1))
    )
)[0][0]

hh1 = rr3_1.subs({x:1, y:0, z:0})
```

[91]:
```
mmA_2 = mm.stack(matrix([1, -5, 1, 2, 0, 1, -2, 1, 3, 7]))
mmB_2 = mm.stack(matrix([-1, -1, 0, 4, 0, 1, 0, 1, 0, -5]))

GA3_2 = mmA_2.stack(matrix([phi((x, y, z), S)[2]])).det()
GB3_2 = mmB_2.stack(matrix([phi((x, y, z), S)[2]])).det()

rr3_2 = list(
    filter(
```

```
        lambda uu: w1 in uu[0].variables(),
        list(factor(w1*GA3_2+w2*GB3_2))
    )
)[0][0]
```

If p_1, p_6, p_7 are aligned, also the following polynomial must be zero

[92]:
```
hh2 = rr3_2.subs({x:1, y:0, z:0})
```

[93]:
```
r3_1 = (w1*GA3_1+w2*GB3_1).subs(
    {
        w1: hh1.coefficient(w2),
        w2: -hh1.coefficient(w1)
    }
).factor()[-1][0]

r3_2 = (w1*GA3_2+w2*GB3_2).subs(
    {
        w1: hh2.coefficient(w2),
        w2: -hh2.coefficient(w1)
    }
).factor()[-1][0]
```

(i.e. r3_1 and r3_2 are the line passing through p1, p6, p7. They should be equal, because they should not depend of the two points of the line L chosen. Indeed, they are equal:

[96]:
```
assert(r3_1 == r3_2)
```

[97]:
```
HH = [hh1, hh2]
JJ = S.ideal([hh.coefficient(w1) for hh in HH]+[hh.coefficient(w2) for hh in␣
 ↪HH])
```

[98]:
```
JJ = JJ.saturation(S.ideal(matrix([p1, p2]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
JJ = JJ.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
```

[100]:
```
assert(JJ == S.ideal(S.one()))
```

This computation shows that there are no exceptions to consider.

[ ]:
```
MM_1 = (w1*mmA_1+w2*mmB_1).subs(
    {
        w1: hh1.coefficient(w2),
        w2: -hh1.coefficient(w1)
    }
)

Mcb1 = MM_1.stack(vector(S, mon))
```

```
MM_2 = (w1*mmA_2+w2*mmB_2).subs(
    {
        w1: hh2.coefficient(w2),
        w2: -hh2.coefficient(w1)
    }
)

Mcb2 = MM_2.stack(vector(S, mon))
```

The cubic is the determinant of Mcb1 (or of Mcb2). one possibility is the following computation (not long)

[101]:
```
cb1 = Mcb1.det().factor()[-1][0]
cb2 = Mcb2.det().factor()[-1][0]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[101], line 1
----> 1 cb1 = Mcb1.det().factor()[-Integer(1)][Integer(0)]
      2 cb2 = Mcb2.det().factor()[-Integer(1)][Integer(0)]

File ~/miniforge3/lib/python3.9/site-packages/sage/matrix/matrix2.pyx:1984, in␣
 ↪sage.matrix.matrix2.Matrix.det (build/cythonized/sage/matrix/matrix2.c:23316)()
   1982         6
   1983     """
-> 1984     return self.determinant(*args, **kwds)
   1985
   1986 def determinant(self, algorithm=None):

File ~/miniforge3/lib/python3.9/site-packages/sage/matrix/
 ↪matrix_mpolynomial_dense.pyx:610, in sage.matrix.matrix_mpolynomial_dense.
 ↪Matrix_mpolynomial_dense.determinant (build/cythonized/sage/matrix/
 ↪matrix_mpolynomial_dense.cpp:12320)()
    608 if isinstance(R, MPolynomialRing_libsingular) and R.base_ring().
 ↪is_field():
    609     singular_det = singular_function("det")
--> 610     d = singular_det(self)
    611
    612 elif can_convert_to_singular(self.base_ring()):

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1298, in sage.libs.singular.function.SingularFunction.__call__ (build/
 ↪cythonized/sage/libs/singular/function.cpp:21339)()
   1296     if not (isinstance(ring, MPolynomialRing_libsingular) or␣
 ↪isinstance(ring, NCPolynomialRing_plural)):
   1297         raise TypeError("cannot call Singular function '%s' with ring␣
 ↪parameter of type '%s'" % (self._name,type(ring)))
-> 1298     return call_function(self, args, ring, interruptible, attributes)
```

```
   1299
   1300 def _instancedoc_(self):

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1477, in sage.libs.singular.function.call_function (build/cythonized/sage/lib
 ↪singular/function.cpp:23326)()
   1475     error_messages.pop()
   1476
-> 1477 with opt_ctx: # we are preserving the global options state here
   1478     if signal_handler:
   1479         sig_on()

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1479, in sage.libs.singular.function.call_function (build/cythonized/sage/lib
 ↪singular/function.cpp:23238)()
   1477 with opt_ctx: # we are preserving the global options state here
   1478     if signal_handler:
-> 1479         sig_on()
   1480         _res = self.call_handler.handle_call(argument_list, si_ring)
   1481         sig_off()

KeyboardInterrupt:
```

[102]:
```
# here is an alternative:
Ms = []
for i in range(10):
    gd = gcd([Mcb1[i,j] for j in range(10)])
    Ms.append([Mcb1[i,j].quo_rem(gd)[0] for j in range(10)])

cb_alt = matrix(Ms).det().factor()[-1][0]
```

We have: cb1 = cb2 and cb2 = cb_alt:

[103]:
```
assert(cb1 == cb2)
assert(cb1 == cb_alt)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[103], line 1
----> 1 assert(cb1 == cb2)
      2 assert(cb1 == cb_alt)

NameError: name 'cb1' is not defined
```

The following example shows that in general there are only the collinearities (1, 2, 3), (1, 4, 5), (1, 6, 7)

[ ]:
```
ccb = cb_alt.subs({A2:5, B2:-3, A4:2, C4:-7})
pp1 = p1.subs({A2:5, B2:-3, A4:2, C4:-7})
pp2 = p2.subs({A2:5, B2:-3, A4:2, C4:-7})
pp3 = p3.subs({A2:5, B2:-3, A4:2, C4:-7})
pp4 = p4.subs({A2:5, B2:-3, A4:2, C4:-7})
pp5 = p5.subs({A2:5, B2:-3, A4:2, C4:-7})
```

HERE WE CONCLUDE THE FIRST PART OF THE COMPUTATION:

In case $C_2 = 0$, IT IS POSSIBLE TO HAVE THREE ALIGNMENTS: (1, 2, 3), (1, 4, 5), (1, 6, 7)

Now we want to see if it is possible to have more then three alignments (We continue to assume $C2 = 0$)

Recall that cb_alt is our cubic.

Recall that r3_1 (= r3_2) is the line through p_6 and p_7.

Up to a permutation of the indices of the points, if there is another alignment among the eigenpoints, p_6 must be on the line $p_2 \vee p_4$. Hence we can find it, since is the intersection of $p_2 \vee p_4$ and r3.

[106]:
```
r24 = det(matrix([p2, p4, (x, y, z)]))
```

[107]:
```
E1 = matrix(
    [
        [r3_1.coefficient(xx) for xx in [x, y, z]],
        [r24.coefficient(xx) for xx in [x, y, z]]
    ]
).minors(2)
```

[108]:
```
p6 = vector(S, (E1[2], -E1[1], E1[0]))
```

[109]:
```
kJ = S.ideal(
    matrix(
        [
            [x, y, z],
            [cb_alt.derivative(x), cb_alt.derivative(y), cb_alt.derivative(z)]
        ]
    ).minors(2)
)
```

[111]:
```
kJ = kJ.saturation(S.ideal(z, y))[0]  ##p1
kJ = kJ.saturation(S.ideal(p2[0]*y-p2[1]*x, p2[0]*z-p2[2]*x,␣
 ↪p2[1]*z-p2[2]*y))[0] ## p2
kJ = kJ.saturation(S.ideal(p3[0]*y-p3[1]*x, p3[0]*z-p3[2]*x,␣
 ↪p3[1]*z-p3[2]*y))[0] ## p3
kJ = kJ.saturation(S.ideal(p4[0]*y-p4[1]*x, p4[0]*z-p4[2]*x,␣
 ↪p4[1]*z-p4[2]*y))[0] ## p4
kJ = kJ.saturation(S.ideal(p5[0]*y-p5[1]*x, p5[0]*z-p5[2]*x,␣
 ↪p5[1]*z-p5[2]*y))[0] ## p5
```

```
kJ = kJ.saturation(S.ideal(matrix([p1, p2]).minors(2)))[0]
kJ = kJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
kJ = kJ.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
kJ = kJ.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
```

```
// ** `sat_with_exp` in use, can not be killed
// ** redefining id (parameter def id; parameter ideal j;  )
elim.lib::sat_with_exp:745
// ** redefining j (parameter def id; parameter ideal j;  )
elim.lib::sat_with_exp:745
// ** redefining ii (   int ii,kk;) elim.lib::sat_with_exp:746
// ** redefining kk (   int ii,kk;) elim.lib::sat_with_exp:746
// ** redefining i (   def i=id;) elim.lib::sat_with_exp:747
// ** redefining p (   int p = printlevel-voice+2;  // p=printlevel (default:
p=0)) elim.lib::sat_with_exp:749
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[111], line 6
      4 kJ = kJ.saturation(S.ideal(p4[Integer(0)]*y-p4[Integer(1)]*x,␣
 ↪p4[Integer(0)]*z-p4[Integer(2)]*x,␣
 ↪p4[Integer(1)]*z-p4[Integer(2)]*y))[Integer(0)] ## p4
      5 kJ = kJ.saturation(S.ideal(p5[Integer(0)]*y-p5[Integer(1)]*x,␣
 ↪p5[Integer(0)]*z-p5[Integer(2)]*x,␣
 ↪p5[Integer(1)]*z-p5[Integer(2)]*y))[Integer(0)] ## p5
----> 6 kJ =␣
 ↪kJ.saturation(S.ideal(matrix([p1, p2]).minors(Integer(2))))[Integer(0)]
      7 kJ = kJ.saturation(S.ideal(matrix([p1, p3]).
 ↪minors(Integer(2))))[Integer(0)]
      8 kJ = kJ.saturation(S.ideal(matrix([p1, p4]).
 ↪minors(Integer(2))))[Integer(0)]

File ~/miniforge3/lib/python3.9/site-packages/sage/rings/qqbar_decorators.py:96␣
 ↪in handle_AA_and_QQbar.<locals>.wrapper(*args, **kwds)
     90 from sage.rings.abc import AlgebraicField_common
     92 if not any(isinstance(a, (Polynomial, MPolynomial, Ideal_generic))
     93             and isinstance(a.base_ring(), AlgebraicField_common)
     94             or is_PolynomialSequence(a)
     95             and isinstance(a.ring().base_ring(), AlgebraicField_common)␣
 ↪for a in args):
---> 96     return func(*args, **kwds)
     98 polynomials = []
    100 for a in flatten(args, ltypes=(list, tuple, set)):

File ~/miniforge3/lib/python3.9/site-packages/sage/rings/polynomial/
 ↪multi_polynomial_ideal.py:2477, in MPolynomialIdeal_singular_repr.
 ↪saturation(self, other)
   2475     sat = ff.elim__lib.sat
```

```
   2476 R = self.ring()
-> 2477 ideal, expo = sat(self, other)
   2478 return (R.ideal(ideal), ZZ(expo))

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1298, in sage.libs.singular.function.SingularFunction.__call__ (build/
 ↪cythonized/sage/libs/singular/function.cpp:21339)()
   1296     if not (isinstance(ring, MPolynomialRing_libsingular) or␣
 ↪isinstance(ring, NCPolynomialRing_plural)):
   1297         raise TypeError("cannot call Singular function '%s' with ring␣
 ↪parameter of type '%s'" % (self._name,type(ring)))
-> 1298     return call_function(self, args, ring, interruptible, attributes)
   1299
   1300 def _instancedoc_(self):

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1477, in sage.libs.singular.function.call_function (build/cythonized/sage/lib:/
 ↪singular/function.cpp:23326)()
   1475     error_messages.pop()
   1476
-> 1477 with opt_ctx: # we are preserving the global options state here
   1478     if signal_handler:
   1479         sig_on()

File ~/miniforge3/lib/python3.9/site-packages/sage/libs/singular/function.pyx:
 ↪1479, in sage.libs.singular.function.call_function (build/cythonized/sage/lib:/
 ↪singular/function.cpp:23238)()
   1477 with opt_ctx: # we are preserving the global options state here
   1478     if signal_handler:
-> 1479         sig_on()
   1480         _res = self.call_handler.handle_call(argument_list, si_ring)
   1481         sig_off()

KeyboardInterrupt:
```

after these computations, kJ is the ideal of the two reminining eigenpoints. we want that p1 defined above is an eigenpoint, so the ideal kkJ here defined must be zero:

[ ]:
```
kkJ = kJ.subs({x:p6[0], y:p6[1], z:p6[2]}).radical()
```

We saturate kkJ

[ ]:
```
kkJ = kkJ.saturation(S.ideal(matrix([p1, p3]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p1, p4]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p1, p5]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p3, p5]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p2, p6]).minors(2)))[0]
kkJ = kkJ.saturation(S.ideal(matrix([p4, p6]).minors(2)))[0]
```

kkJ is (1), so there are no solutions:

[ ]:
```
assert(kkJ == S.ideal(1))
```

CONCLUSION (for the case $C_2 = 0$) when the three deltas are zero, (hence rank of the matrix of the five points is 8), we have also $\delta_2 = 0$ and we have the collinearities $(1, 2, 3)$, $(1, 4, 5)$. We have a sub-case in which there is the further collinearity $(1, 6, 7)$. No other collinearities among the 7 eigenpoints are possible.

[ ]:
```

```