NB.06.F5

July 23, 2024

1 Theorem

Suppose that Γ is a conic and let C be a cubic such that $\Gamma \subseteq \mathrm{Eig}(C)$. Then we have three possible cases: * $\Gamma = \mathcal{Q}_{\text{iso}}$. This is true if and only if $C = \ell\mathcal{Q}_{\text{iso}}$ where ℓ is any line of the plane; * Γ is bitangent to $\mathcal{Q}_{\text{iso}}$ in two distinct points P_1 and P_2. This is true iff $C = r(\lambda\mathcal{Q}_{\text{iso}} + \mu r^2)$, where $\lambda, \mu \in \mathbb{C}$, $r = P_1 \vee P_2$. In this case $\Gamma = \lambda\mathcal{Q}_{\text{iso}} + 3\mu r^2$; * Γ is iperosculating $\mathcal{Q}_{\text{iso}}$ in a point P. This is true iff $C = r(\mathcal{Q}_{\text{iso}} - r^2)$, where r is the tangent to $\mathcal{Q}_{\text{iso}}$ in P. In this case $\Gamma = \mathcal{Q}_{\text{iso}} - 3r^2$.

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

If `do_long_computations` is `True`, we perform all the computations, if it is `False`, we load the computations done in a previous session.

[2]:
```
do_long_computations = True
```

1.1 Four distinct eigenpoints $P_1, \dots, P_4$ on the isotropic conic

We assume $P_1 = (1 : i : 0)$ and we define the generic point on the isotropic conic, which depends on two parameteres: l_1 and l_2. We can assume $l_2 \neq 0$:

[3]:
```
P1 = vector(S, (1, ii, 0))
Pg = vector(S, ((-ii)*l1^2 + (-ii)*l2^2, l1^2 - l2^2, 2*l1*l2))
```

[4]:
```
assert(matrix([P1, Pg.subs(l2=0)]).rank() == 1)
```

Now we define three points on the isotropic conic and the matrix of conditions of the four eigenpoints

[5]:
```
P2 = Pg.subs({l1:u1, l2:1})
P3 = Pg.subs({l1:v1, l2:1})
P4 = Pg.subs({l1:w1, l2:1})

M = condition_matrix([P1, P2, P3, P4], S, standard="all")
```

1.1.1 First, we assume that none of the four points is $(1 : -i : 0)$.

Since the point $(1, -i, 0)$ is NOT one of the points P_2, P_3, P_4, we can assume that u_1, v_1, w_1 are not zero. The consequence is that the third coordinate of P_2, P_3, P_4 is not zero. First we see that if $(1 : -i : 0)$ is one of the points P_2, P_3, P_4 then we have u_1 or v_1 or w_1 zero.

```
[6]: assert(S.ideal(matrix([(1, -ii, 0), P2]).minors(2)).groebner_basis()==[u1])
     assert(S.ideal(matrix([(1, -ii, 0), P3]).minors(2)).groebner_basis()==[v1])
     assert(S.ideal(matrix([(1, -ii, 0), P4]).minors(2)).groebner_basis()==[w1])

     assert(P2[2] == 2*u1)
     assert(P3[2] == 2*v1)
     assert(P4[2] == 2*w1)
```

Since

$$P_{2,z}M_{(4)} - P_{2,y}M_{(5)} + P_{2,x}M_{(6)} = 0$$
$$P_{3,z}M_{(7)} - P_{3,y}M_{(8)} + P_{3,x}M_{(9)}] = 0$$
$$P_{4,z}M_{(10)} - P_{4,y}M_{(11)} + P_{4,x}M_{(12)} = 0$$

the rows $M_{(4)}, M_{(7)}, M_{(10)}$ are unnecessary in our hypothesis; similarly, the row $M_{(2)}$ is also unnecessary. We construct the matrix M_1 eraising the rows $2, 3, 6, 9$ from M:

```
[7]: M1 = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11])
```

M_1 has always rank 7: it cannot have rank 8, since all the order 8 minors are zero:

```
[8]: assert(Set(M1.minors(8)) == Set([S.zero()]))
```

If we impose that all the 7-minors of M_1 are zero, we get that (if the points P_1, P_2, P_3, P_4 are distinct) there are no solutions:

```
[9]: mm7 = S.ideal(M1.minors(7))
     mm7 = mm7.saturation(S.ideal(u1*v1*w1))[0]
     mm7 = mm7.saturation(S.ideal((u1-v1)*(u1-w1)*(v1-w1)))[0]
     assert(mm7 == S.ideal(S(1)))
```

All the cubics of the form $\ell * \mathcal{Q}_{\text{iso}}$ have $\mathcal{Q}_{\text{iso}}$ in the eigenscheme:

```
[10]: assert(gcd(eig((x*u2+y*v2+z*w2)*Ciso)).quo_rem(Ciso)[1]== 0)
```

Conclusion 1: If four distinc points of $\mathcal{Q}_{\text{iso}}$ are eigenpoints of a cubic (and if one of the points is not $(1, -i, 0)$), then the cubic splits into $\mathcal{Q}_{\text{iso}}$ and a line.

Proof: The matrix M_1 has always rank 7, hence all the cubics with P_1, P_2, P_3, P_4 as eigenpoints are a linear variety in $\mathbb{P}^9$ of dimension 2. However, this family contains all the cubics which split into $\mathcal{Q}_{\text{iso}}$ and a line of the plane, which is of dimension 2, hence the two families coincide.

1.1.2 Now we assume that P_2 is $(1 : -i, 0)$

```
[11]: P2 = vector(S, (1, -ii, 0))
      P3 = Pg.subs({l1:v1, l2:1})
      P4 = Pg.subs({l1:w1, l2:1})

      M = condition_matrix([P1, P2, P3, P4], S, standard="all")
```

As above, we have that $M_{(3)}$, $M_{(6)}$, $M_{(7)}$, $M_{(10)}$ are unnecessary so we erase these rows from M:

[12]:
```
M1 = M.matrix_from_rows([0, 1, 3, 4, 7, 8, 10, 11])
```

M_1 has rank 7. It cannot have rank 8:

[13]:
```
assert(Set(M1.minors(8)) == Set([S.zero()]))
```

[14]:
```
mm7 = S.ideal(M1.minors(7))
mm7 = mm7.saturation(S.ideal(v1*w1))[0]
mm7 = mm7.saturation(S.ideal((v1-w1)))[0]
assert(mm7 == S.ideal(S.one()))
```

Conclusion 2: Also in case one of the point $(1, -i, 0)$ we obtain the same conclusion: if four eigenpoints of a cubic are on $\mathcal{Q}_{\text{iso}}$, then $\mathcal{Q}_{\text{iso}}$ is contained in the eigenscheme of the cubic.

1.2 Case Γ tangent to $\mathcal{Q}_{\text{iso}}$ in P_1 and passing through P_2 and P_3

We redefine $P_1 = (1 : i : 0)$, here we define the generic point P_g on $\mathcal{Q}_{\text{iso}}$ and two points P_2 and P_3 on $\mathcal{Q}_{\text{iso}}$:

[15]:
```
P1 = vector(S, (1, ii, 0))
Pg = vector(S, ((-ii)*l1^2 + (-ii)*l2^2, l1^2 - l2^2, 2*l1*l2))
assert(matrix([P1, Pg.subs(l2=0)]).rank() == 1)

P2 = Pg.subs({l1:u1, l2:1})
P3 = Pg.subs({l1:v1, l2:1})
```

Also here we consider two cases: * $(1 : -i : 0)$ is not one of the points P_2, P_3; * $P_3 = (1 : -i : 0)$.

1.2.1 Case P_2 and P_3 not $(1 : -i : 0)$.

Hence, as above, we can assume that u_1 and v_1 are not zero.

We compute the pencil of conics C_g passing through P_1, P_2, P_3 and tangent to $\mathcal{Q}_{\text{iso}}$ in P_1:

c_2 is the conic given by the two lines $P_1 \vee P_2$ and $P_1 \vee P_3$

[16]:
```
c2 = matrix([P1, P2, [x, y, z]]).det()*matrix([P1, P3, [x, y, z]]).det()
Cg = Ciso+l1*c2
```

Now we construct a generic point (different from P_1) on C_g, depending on the parameter w_1 (P_{gn} is the generic point on C_g)

[17]:
```
foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]

Pgn = vector(
    S,
    (
        foo.coefficient(z),
        ii*(foo.coefficient(z)) + w1*(-foo.coefficient(x)),
        -foo.coefficient(x)
    )
)

## Pg is the following:
assert(
    Pgn ==
    vector(
        S,
        (
            u1*v1*w1^2*l1 + u1*w1*l1 + v1*w1*l1 + 1/4*w1^2 + l1 + 1/4,
            ii*u1*v1*w1^2*l1 + ii*u1*w1*l1 + ii*v1*w1*l1 + (-1/4*ii)*w1^2 +␣
 ↪ii*l1 + (1/4*ii),
            (-1/2*ii)*w1
        )
    )
)

Pg1, Pg2 = Pgn.subs({w1:w1}), Pgn.subs({w1:w2})
```

Pg_1, Pg_2 are two points on C_g. We can assume l_1, u_1, v_1, w_1, w_2, $(w_1 - w_2)$, $(u_1 - v_1)$, $(v_1w_2 + 1)$, $(u_1w_2 + 1)$, $(v_1w_1 + 1)$, $(u_1w_1 + 1)$ all different from 0.

Indeed, $l_1 = 0$ gives $C_g = \mathcal{Q}_{\text{iso}}$ (condition already considered above) $u_1 = 0$ or $v_1 = 0$ gives that P_2 or P_3 is the point $(1, -i, 0)$, and we are in case 1, so this is not possible;

$w_1 = 0$ gives $Pg_1 = P_1$:

$w_2 = 0$ gives $Pg_2 = P1$:

$w_1 - w_2 = 0$ gives $Pg_1 = Pg_2$

$u_1 - v_1 = 0$ gives $P_2 = P_3$

$v_1w_2 + 1 = 0$ gives $P_3 = Pg_2$:

[18]:
```
assert(Set(matrix([Pg1.subs(w1=0), P1]).minors(2)) == Set([S(0)]))
assert(Set(matrix([Pg2.subs(w2=0), P1]).minors(2)) == Set([S(0)]))
assert(
    S.ideal(
        matrix([P3, Pg2]).minors(2)
    ).saturation(w2)[0] == S.ideal(v1*w2+1)
)
```

Similarly, $(u_1w_2 + 1)$, $(v_1w_1 + 1)$, $(u_1w_1 + 1)$ give, respectively: $P_2 = Pg_2$, $P_3 = Pg_1$, $P_2 = Pg_1$. Hence we define a polynomial `degenerate_cases`, which contains all these degenerate cases and we can saturate our computations w.r.t. this polynomial. So we construct the polynomial of degenerate cases, in order to compute saturations.

[19]:
```
degenerate_cases =␣
 ↪l1*v1*u1*w1*w2*(w1-w2)*(u1-v1)*(v1*w2+1)*(u1*w2+1)*(v1*w1+1)*(u1*w1+1)
```

The points P_1, P_2, P_3, Pg_1, Pg_2 are 5 points on the conic C_g. If they are eigenpoints, the following matrix must have rank 9 or less:

```
[20]: M = condition_matrix([P1, P2, P3, Pg1, Pg2], S, standard="all")
```

We have $P_{1,z}M_{(1)} - P_{1,y}M_{(2)} + P_{1,x}M_{(3)} = 0$ and $P_{1,x} = 1$ so $M_{(3)}$ is linearly dependent on $M_{(1)}$ and $M_{(2)}$, and can be omitted.

```
[21]: assert(P1[2]*M[0]-P1[1]*M[1]+P1[0]*M[2] == 0)
      assert(P1[0] == 1)
```

We have $P_{2,z}M_{(4)} - P_{2,y}M_{(4)} + P_{2,x}M_{(6)} = 0$ $P_{2,z} = 2u_1$ which, under our hypothesis, is always not zero, so $M_{(4)}$ can be omitted.

```
[22]: assert(P2[2]*M[3]-P2[1]*M[4]+P2[0]*M[5] == 0)
      assert(P2[2] == 2*u1)
```

In a similar way, $M_{(7)}$, $M_{(10)}$, $M_{(13)}$, $_{(16)}$ can be omitted. Hence we construct the square matrix of order 10:

```
[23]: MM1 = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11, 13, 14])
```

The first row of MM1 is $(-3i, 3, 3i, -3, 0, 0, 0, 0, 0, 0)$. The second row is $(0, 0, 0, 0, 1, i, -1, 0, 0, 0)$

```
[24]: assert(MM1[0] == vector(S, ((-3*ii), 3, (3*ii), -3, 0, 0, 0, 0, 0, 0)))
      assert(MM1[1] == vector(S, (0, 0, 0, 0, 1, ii, -1, 0, 0, 0)))
```

So with elementary row operations we can simplify MM1:

```
[25]: MM1.rescale_row(0, 1/3*ii)
      for i in range(2, 10):
          MM1.add_multiple_of_row(i, 0, -MM1[i][0])

      for i in range(2, 10):
          MM1.add_multiple_of_row(i, 1, -MM1[i][4])
```

Now the 0-th column of MM1 is $(1, 0, \dots, 0)$

and the 4-th column of MM1 is $(0, 1, 0, \dots, 0)$

```
[26]: assert([MM1[i, 0] for i in range(10)] == [1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
      assert([MM1[i, 4] for i in range(10)] == [0, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

We extract from MM1 an order 8 square matrix, extracting the last 8 rows and the columns of position 1, 2, 3, 5, 6, 7, 8, 9.

We get a new matrix MM1.

This new matrix MM1 has rank n iff the original MM1 has rank $n + 2$ (iff M has rank $n + 2$) In particular, we want to see if MM1 can have rank ≤ 6.

```
[27]: MM1 = MM1.matrix_from_rows_and_columns(
          [2, 3, 4, 5, 6, 7, 8, 9],
          [1, 2, 3, 5, 6, 7, 8, 9]
      )
```

```
[28]: if do_long_computations:
          ttA = cputime()
          mm1_7 = MM1.minors(7)
          print("Computation of the 64 order 7 minors:")
          print("time: "+str(cputime()-ttA))
          save(mm1_7, "NB.06.F5-mm1_7.sobj")
      else:
          mm1_7 = load("NB.06.F5-mm1_7.sobj")
```

```
Computation of the 64 order 7 minors:
time: 295.01199800000006
```

```
[29]: J7 = S.ideal(mm1_7)
      J7 = J7.saturation(degenerate_cases)[0]
```

The above ideal is (1), so it is not possible to have that the matrix MM1 has rank ≤ 6 (hence it is not possible that M has rank ≤ 8)

```
[30]: assert(J7 == S.ideal(S(1)))
```

In order to have P_1, P_2, Pg_1, Pg_2, Pg_3 eigenpoints, M must have zero determinant. But $\det(M) = \det(MM1)$. So we compute $\det(MM1)$ and we saturate it w.r.t. `degenerate_cases`.

```
[32]: if do_long_computations:
          ttA = cputime()
          ddt = MM1.det()
          ddt = S.ideal(ddt).saturation(degenerate_cases)[0].gens()[0]
          print(cputime()-ttA)
          sleep(1)
      else:
          ddt = (16) * (l1 + 1/4)^2 * (u1*w1*w2 + v1*w1*w2 + w1 + w2)

      assert(ddt == (16) * (l1 + 1/4)^2 * (u1*w1*w2 + v1*w1*w2 + w1 + w2))
```

```
0.020036000000004606
```

Hence we have two possibilities: * $l_1 + 1/4 = 0$, or * $u_1w_1w_2 + v_1w_1w_2 + w_1 + w_2 = 0$.

This second condition must be satisfied for every point Pg_2 of the conic C_g, i.e. for every w_2, therefore this condition is impossible.

Hence the only possible case is $l_1 = -1/4$. In this case C_g splits into two lines: * the line $x + iy$ * the line $xu_1v_1 + iyu_1v_1 + izu_1 + izv_1 + x - iy$

The first line is the tangent line to $\mathcal{Q}_{\text{iso}}$ in the point P_1 and the second line is the line passing through the points P_2 and P_3:

```
[33]: assert(S.ideal(Ciso, x+ii*y).groebner_basis() == [z^2, x + ii*y])

      assert(
          Cg.subs(l1=-1/4) ==
          (x+ii*y)*S(matrix([P2, P3, (x, y, z)]).det()/(2*(u1-v1)))
      )
```

The matrix `M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11, 13, 14])` i.e., the original matrix extracted from M, with the condition $l_1 = -1/4$ has rank 9:

[34]:
```
MM2 = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11, 13, 14]).subs(l1 = -1/4)
assert(MM2.rank() == 9)
```

and the last row of MM2 is linearly dependent w.r.t. the other rows of MM2. Indeed, the rank of MM2 without the last row is still 9:

[35]:
```
MM3 = MM2.matrix_from_rows([0, 1, 2, 3, 4, 5, 6, 7, 8])
assert(MM3.rank() == 9)
```

hence the cubic corresponding to the matrix MM2 is:

[39]:
```
if do_long_computations:
    ttA = cputime()
    cbb = det(MM3.stack(matrix([mon])))
    print(str(cputime()-ttA) + " seconds of computation")
    sleep(1)
    ttA = cputime()
    sleep(1)
    cbb = S.ideal(cbb).saturation(degenerate_cases)[0].gens()[0]
    print(str(cputime()-ttA) + " seconds of computation")
else:
    cbb = (x + ii*y) * (u1*v1 - 1) * (x*u1*v1 + ii*y*u1*v1 + ii*z*u1 + ii*z*v1␣
 ↪+ x + (-ii)*y)^2

assert(cbb == (x + ii*y) * (u1*v1 - 1) * (x*u1*v1 + ii*y*u1*v1 + ii*z*u1 +␣
 ↪ii*z*v1 + x + (-ii)*y)^2)
```

```
40.211033999999984 seconds of computation
10.736583999999937 seconds of computation
```

`cbb` is the cubic obtained from the matrix MM3, hence is the unique cubic that has $P_1, P_2, P_3, Pg_1, Pg_2$ as eigenpoints and `cbb` splits into the line $(P_1 \vee P_2)^2$ and the tangent to $\mathcal{Q}_{\text{iso}}$ in the point P_1. The eigenpoints are the line $P_1 \vee P_2$ plus other points and is not a conic.

This conclude the case Γ tangent to $\mathcal{Q}_{\text{iso}}$ in one point in case P_2 and P_3 are different from the point $(1 : -i : 0)$.

1.2.2 Case $P_3 = (1 : -i : 0)$

[40]:
```
P3 = ii*P3.subs(v1=0)
assert(P3 == vector(S, (1, -ii, 0)))
```

We construct the conic given by two lines $P_1 \vee P_2$ and $P_1 \vee P_3$ and the generic conic tangent to $\mathcal{Q}_{\text{iso}}$ in P_1 and passing through P_2 and P_3 (it depends on the parameter l_1):

```
[41]: c2 = matrix([P1, P2, [x, y, z]]).det()*matrix([P1, P3, [x, y, z]]).det()

      Cg = Ciso+l1*c2
```

Construction of a generic point (different from $(1, i, 0)$) on C_g and generic point of C_g (depends on the parameter w_1) and construction of Pg_1 and Pg_2 (two points on C_g)

```
[42]: foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]

      Pg = vector(
          S,
          (
              foo.coefficient(z),
              ii*(foo.coefficient(z)) + w1*(-foo.coefficient(x)),
              -foo.coefficient(x)
          )
      )
```

```
[43]: Pg1, Pg2 = Pg.subs({w1:w1}), Pg.subs({w1:w2})
```

Matrix of conditions of P_1, P_2, P_3, Pg_1, Pg_2:

```
[44]: M = condition_matrix([P1, P2, P3, Pg1, Pg2], S, standard="all")
```

We have P1[2]*M[0]-P1[1]*M[1]+P1[0]*M[2] = 0 and P1[0] = 1 so M[2] is lin dep of M[0] and M[1] and can be omitted.

We have P2[2]*M[3]-P2[1]*M[4]+P2[0]*M[5] = 0 and P2[2] = 2*u1 which, under our hypothesis, is always not zero, so M[3] can be omitted.

```
[45]: assert(P1[2]*M[0]-P1[1]*M[1]+P1[0]*M[2] == 0)
      assert(P1[0] == 1)
      assert( P2[2]*M[3]-P2[1]*M[4]+P2[0]*M[5] == 0)
      assert(P2[2] == 2*u1)
```

In a similar way, M[9], M[12], M[15] can be omitted.

P3[2]*M[6]-P3[1]*M[7]+P3[0]*M[8] = 0 and P3[0] = 1, hence M[8] can be omitted.

Hence we construct the square matrix of order 10:

```
[46]: MM1 = M.matrix_from_rows([0, 1, 4, 5, 6, 7, 10, 11, 13, 14])
```

The first row of MM1 is ((-3*ii), 3, (3*ii), -3, 0, 0, 0, 0, 0, 0)

The second row is (0, 0, 0, 0, 1, ii, -1, 0, 0, 0)

```
[47]: assert(MM1[0] == vector(S, ((-3*ii), 3, (3*ii), -3, 0, 0, 0, 0, 0, 0)))
      assert(MM1[1] == vector(S, (0, 0, 0, 0, 1, ii, -1, 0, 0, 0)))
```

so with elementary row operations we can simplify MM1:

[48]:
```
MM1.rescale_row(0, 1/3*ii)
for i in range(2, 10):
    MM1.add_multiple_of_row(i, 0, -MM1[i][0])

for i in range(2, 10):
    MM1.add_multiple_of_row(i, 1, -MM1[i][4])
```

Now the 0-th column of MM1 is (1, 0, ..., 0) and the 4-th column of MM1 is (0, 1, 0, ..., 0)

[49]:
```
assert([MM1[i, 0] for i in range(10)] == [1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
assert([MM1[i, 4] for i in range(10)] == [0, 1, 0, 0, 0, 0, 0, 0, 0, 0])
```

We extract from MM1 an order 8 square matrix, extracting the last 8 rows and the columns of position 1, 2, 3, 5, 6, 7, 8, 9.

[50]:
```
MM1 = MM1.matrix_from_rows_and_columns(
    [2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 5, 6, 7, 8, 9]
)
```

This new matrix MM1 has rank n iff the original MM1 has rank n+2 (iff M has rank n+2) In particular, we want to see if MM1 can have rank <= 6.

[51]:
```
if do_long_computations:
   ttA = cputime()
   mm1_7b = MM1.minors(7)
   print("Computation of the 64 order 7 minors:")
   sleep(1)
   print("time: " + str(cputime()-ttA))
   save(mm1_7b, "NB.06.F5-mm1_7b.sobj")
else:
   mm1_7b = load("NB.06.F5-mm1_7b.sobj")
```

```
Computation of the 64 order 7 minors:
time: 14.641388000000006
```

[52]:
```
dgnCs = l1*u1*w1*w2*(w1-w2)*(u1-v1)*(v1*w2+1)*(u1*w2+1)*(v1*w1+1)*(u1*w1+1)
```

[53]:
```
J7 = S.ideal(mm1_7b)
J7 = J7.saturation(dgnCs)[0]
```

The above ideal is (1), so it is not possible to have that the matrix MM1 has rank <= 6 (hence it is not possible that M has rank <= 8)

[54]:
```
assert(J7 == S.ideal(S(1)))
```

We want now to compute the determinant of the original MM1 (the order 10 matrix) which is equal to the determiant of the order 8 matrix MM1:

[55]:
```
dt2 = MM1.det()
dt2 = S.ideal(dt2).saturation(dgnCs)[0].gens()[0]
```

we get that dt2 is:

(16) • (l1 + (-1/4*ii))^2* (u1*w1*w2 + w1 + w2)

[56]:
```
assert(dt2 == (16) * (l1 + (-1/4*ii))^2 * (u1*w1*w2 + w1 + w2))
```

We have u1*w1*w2+w1+w2 = 0 and, as in the previous case, since w2 is generic, does not give solutions.

The case l1 = 1/4*ii In this case Cg splits into the line x+ii*y (tangent line to Cg in P1) and the line P2+P3:

[57]:
```
assert(
    (-2)*u1*Cg.subs(l1=1/4*ii)
    == ii*(x+ii*y)*matrix([P2, P3, (x, y, z)]).det()
)
```

We consider the substitution of l1=1/4*ii in the original MM, i.e. in MM1 = M.matrix_from_rows([0, 1, 4, 5, 6, 7, 10, 11, 13, 14])

[58]:
```
MM2 = M.matrix_from_rows([0, 1, 4, 5, 6, 7, 10, 11, 13, 14]).subs(l1=1/4*ii)
```

Also here the last row of MM2 is unnecessary:

[59]:
```
assert(MM2.rank() == MM2.matrix_from_rows(range(9)).rank())
```

Hence we compute the cubic:

[60]:
```
M3 = (MM2.matrix_from_rows(range(9))).stack(matrix([mon]))
cb2 = M3.det()
```

The cubic splits into the line tangent to $\mathcal{Q}_{\text{iso}}$ in P_1 and the reducible conic given by $(P_2 \vee P_3)^2$

[61]:
```
assert(
    S.ideal(cb2).saturation(dgnCs)[0].gens()[0]
    == (x + ii*y) * (z*u1 + (-ii)*x - y)^2
)
```

also in this case, we do not get a cubic which has a conic among its eigenpoints.

This conclude the case of Γ tangent to $\mathcal{Q}_{\text{iso}}$

1.3 Γ bitangent to $\mathcal{Q}_{\text{iso}}$

Now we consider the case in which among the eigenpoints of a cubic, we have a concic Γ which is bitangent to $\mathcal{Q}_{\text{iso}}$ in two points P_1 and P_2. We restart the computations:

[62]:
```
do_long_computations = True
```

Construction of a generic point Pg on the isotropic conic:

[63]:
```
P1 = vector(S, (1, ii, 0))
rt1 = l1*(y-ii*x)+l2*z

rt1.subs({x:P1[0], y:P1[1], z:P1[2]})

scndP = S.ideal(Ciso, rt1).radical().primary_decomposition()[1]
aux = scndP.gens()[:2]
mm2 = matrix(
    [
        [aux[0].coefficient(x), aux[0].coefficient(y), aux[0].coefficient(z)],
        [aux[1].coefficient(x), aux[1].coefficient(y), aux[1].coefficient(z)]
    ]
).minors(2)
```

[64]:
```
Pg = vector(S, (mm2[2], -mm2[1], mm2[0]))
```

[65]:
```
assert(scndP.subs({x:Pg[0], y:Pg[1], z:Pg[2]}) == S.ideal(S(0)))
assert(Ciso.subs({x:Pg[0], y:Pg[1], z:Pg[2]}) == S(0))
```

We can always assume that l2 != 0, since l2 = 0 gives that Pg = P1

[66]:
```
assert(matrix([P1, Pg.subs(l2=0)]).rank() == 1)
```

We construct P2, a generic point on $\mathcal{Q}_{\text{iso}}$ (it depends on u1)

[67]:
```
P2 = Pg.subs({l1:u1, l2:1})
```

And now we construct the tangent line to $\mathcal{Q}_{\text{iso}}$ in P1, the tangent line to $\mathcal{Q}_{\text{iso}}$ in P2, the pencil of conics tangent to $\mathcal{Q}_{\text{iso}}$ in P1 and P2 (it depends on l1)

[68]:
```
rtg1 = scalar_product(P1, vector((x, y, z))-P1)
rtg2 = scalar_product(P2, vector((x, y, z))-P2)*ii

Cg = Ciso + l1*rtg1*rtg2
```

If l1 = -1, Cg is the conic given by (P1+P2)^2, so we can assume l1+1 != 0

[69]:
```
assert(4*Cg.subs(l1=-1) == det(matrix([P1, P2, (x, y, z)]))^2)
```

construction of a generic point (different from (1, ii, 0)) on Cg:

[70]:
```
foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]
```

Generic point of Cg (depends on the parameter w1):

[71]:
```
Pg2 = vector(
    S,
    (
        foo.coefficient(z),
        ii*(foo.coefficient(z)) + w1*(-foo.coefficient(x)),
        -foo.coefficient(x)
    )
)
```

the last coordinate of Pg2 is ((2*ii*)) (l1 + 1) * w1. If w1 = 0, then Pg2 = P1, hence we can assume w1 != 0.

[72]:
```
assert(matrix([P1, Pg2.subs(w1=0)]).rank() == 1)
```

Now we define three points on Cg:

[73]:
```
Pa1, Pa2 = Pg2.subs({w1:w1}), Pg2.subs({w1:w2})
Pa3 = Pg2.subs({w1:m1})
```

and we can assume w1, w2, m1 != 0.

the following matrix must have rank <= 9:

[74]:
```
M = condition_matrix([P1, P2, Pa1, Pa2, Pa3], S, standard="all")
```

1.3.1 We assume that $u_1 \neq 0$, so P_2 is not the point $(1 : -i : 0)$.

Under this hypothesis, we have that we can extract from M the rows: 0, 1; 4, 5; 7, 8; 10, 11; 13, 14. (Remember that w1, w2, m1 are not zero).

[75]:
```
MM1 = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11, 13, 14])
```

[76]:
```
# we make a copy of MM1
MM2 = M.matrix_from_rows([0, 1, 4, 5, 7, 8, 10, 11, 13, 14])
```

Using the fact that the first two rows are good (MM2[0,0] is a non zero constant, and MM2[1, 4] is 1) we can reduce MM2 with elementary rows and columns operations.

[77]:
```
MM2.rescale_row(0, ii/3)
for i in range(2, 10):
    MM2.add_multiple_of_row(i, 0, -MM2[i][0])

for i in range(2, 10):
    MM2.add_multiple_of_row(i, 1, -MM2[i][4])
```

We extract from MM2 an order 8 square matrix, extracting the last 8 rows and the columns of position 1, 2, 3, 5, 6, 7, 8, 9.

[78]:
```
MM2 = MM2.matrix_from_rows_and_columns(
    [2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 5, 6, 7, 8, 9]
)
```

The computation of det(MM2) gives 0 (time of computation: 3' 20' ')

[79]:
```
if do_long_computations:
   ttA = cputime()
   dtMM2 = MM2.det()
   print("Computation of the determinant of MM2:")
   print("time: "+str(cputime()-ttA))
   sleep(1)
else:
   dtMM2 = S.zero()

assert(dtMM2 == S.zero())
```

```
Computation of the determinant of MM2:
time: 103.08253500000006
```

Hence M and MM1 have rank <= 9. We want to see when M has rank <=8, i.e. when MM2 has rank <=6 hence we compute the ideal of the order 7-minors of MM2. Time of computation: 35'

If doLongComputations=false, it takes some seconds to load the file

[80]:
```
if do_long_computations:
   ttA = cputime()
   mm2_7 = MM2.minors(7)
   print("Computation of the order 7 minors:")
   print("time: "+str(cputime()-ttA))
   save(mm2_7, "NB.06.F5-mm2_7.sobj")
   sleep(1)
else:
   mm2_7 = load("NB.06.F5-mm2_7.sobj")
```

```
Computation of the order 7 minors:
time: 1000.0280200000001
```

here is a list of factors that cannot be zero:

[81]:
```
nonZeroFt = [
    l1,v1,u1,w1,w2,m1,l1+1,
    (w1-w2),
    (u1-v1),
    (w2-m1),
    (w1-m1),
    (v1*w2+1),
    (u1*w2+1),
    (v1*w1+1),
    (u1*w1+1),
    (u1*m1+1)
]
```

[82]:
```
# an auxiliary function:
# input: a polynomial pol and a list of polynomials listFt:
# output: a list of polynomials obtained deleting from the factors of pol
# those factors which are contained in the list listFt. Each factor is taken
# with exponent 1.
# example:
# erase_superfluous_factors(x^3*(x+1)^4*(z+y)^3*(x+y+z)^5, [x, z+y, z])
# answer: [x+1, x+y+z]
```

[83]:
```
def erase_superfluous_factors(pol, list_factors):
    if pol == S.zero():
        return(pol)
    ftOK = []
    for ft in list(pol.factor()):
        if not ft[0] in list_factors:
            ftOK.append(ft[0])
    return(ftOK)
```

here from each order 7-minor of MM2 (i.e. every element of mm2_7) we clear off the superfluous factors and we construct the ideal J7 of these polynomials (7’’ of computation)

[84]:
```
J7 = []
for ff in mm2_7:
    J7.append(prod(erase_superfluous_factors(ff, nonZeroFt)))
```

since the ideal J7 (after suitable saturation) is (1), we have that the matrix MM2 cannot have rank 6 or smaller, hence M cannot have rank 8 or smaller:

[85]:
```
assert(S.ideal(J7).saturation((l1+1)*(w2-m1))[0] == S.ideal(S(1)))
```

The above computations show that the matrix M has rank 9, there is thereofre only one cubic given by M and is obtained from the determinant of the matrix Mc below, given when we stack to MM1 the row ginven by the list mon.

The next block constructs the cubic Cbc, which is the unique cubic given by the matrix M.

[86]:
```
Mc = MM1.matrix_from_rows(range(9))
Mc = Mc.stack(matrix([mon]))
```

[87]:
```
if do_long_computations:
   ttA = cputime()
   dtMc = Mc.det()
   print("Computation of the cubic:")
   print("time: "+str(cputime()-ttA))
   save(dtMc, "NB.06.F5-dtMc.sobj")
   sleep(1)
else:
   dtMc = load("NB.06.F5-dtMc.sobj")
```

```
Computation of the cubic:
time: 1077.124312
```

[88]:
```
## from dtMc we erase useless factors:

dt1 = erase_superfluous_factors(dtMc, nonZeroFt)
```

We get that dt1 is a list of 4 elements, which are:

[89]:
```
Ls = [
    x*u1 + ii*y*u1 + ii*z,
    u1*w1*w2 + 1/2*w1 + 1/2*w2,
    u1^2*l1*m1^2 - l1*m1^2 - 2*u1*m1 - m1^2 - 1,
    x^2*u1^2*l1 + (2*ii)*x*y*u1^2*l1 - y^2*u1^2*l1 + (2*ii)*x*z*u1*l1
    - 2*y*z*u1*l1 + 3*x^2*l1 + 3*y^2*l1 + 2*z^2*l1 + 3*x^2 + 3*y^2 + 3*z^2
]
```

[90]:
```
assert(dt1 == Ls)
```

We select the two factors which are a line and a conic and whose product is the desired cubic.

The factor dt1[0] is the line r12 given by P1+P2:

[91]:
```
r12 = det(matrix([P1, P2, (x, y, z)]))/(2*ii)
```

[92]:
```
assert(dt1[0] == r12)
```

The factor Lt[3] is the l1*r12ˆ2+3*(l1+1)*Ciso:

[93]:
```
assert(dt1[3] == l1*r12^2+3*(l1+1)*Ciso)
```

The cubic is therefore:

[94]:
```
Cbc = dt1[0]*dt1[3]
```

The cubic Cbc is of the form r12*(u*r12ˆ2+v*Ciso) for suitable u, v

[95]:
```
assert(Cbc == r12*(l1*r12^2+3*(l1+1)*Ciso))
```

now we compute the eigenpoints of Cbc:

[96]:
```
Je = S.ideal(
    matrix(
        [
            [
                Cbc.derivative(x),
                Cbc.derivative(y),
                Cbc.derivative(z)
            ],
            [x, y, z]
        ]
    ).minors(2)
)
```

we get two components, which are: the ideal generated by rtg1, rtg2 (hence is a point) and the principal ideal generated by Ciso+l1*rtg1*rtg2

[97]:
```
PDe = Je.primary_decomposition()
```

[98]:
```
assert(len(PDe) == 2)
assert(PDe[0] == S.ideal(rtg1, rtg2))
assert(PDe[1] == S.ideal(Ciso+l1*rtg1*rtg2))
```

Conversely, we assume here that the cubic C is of the form C1 = (l1*Ciso+l2*(u1*x+v1*y+w1*z)^2)*(u1*x+v1*y+w1*z) where u1*x+v1*y+w1*z is a generic line of the plane and l1 and l2 are parameters.

[99]:
```
C1 = (l1*Ciso+l2*(u1*x+v1*y+w1*z)^2)*(u1*x+v1*y+w1*z)
```

We compute the eigenpoints of C1: and the primary decomposition of the ideal of the eigenpoints. We get two components:

The first is:

Ideal (*zv1* - *y*w1, *zu1* - *x*w1, *yu1* - *x*v1)

i.e. the point (u1, v1, w1)

and the second component is the conic

l1*Ciso+3*l2*(u1*x+v1*y+w1*z)^2

[100]:
```
JJ = S.ideal(
    matrix(
        [
            [
                C1.derivative(x),
                C1.derivative(y),
                C1.derivative(z)
            ],
            [x, y, z]
        ]
    ).minors(2)
)
```

[101]:
```
pdJJ = JJ.primary_decomposition()
```

[102]:
```
assert(pdJJ[0] == S.ideal (z*v1 - y*w1, z*u1 - x*w1, y*u1 - x*v1))
assert(pdJJ[1] == S.ideal(l1*Ciso+3*l2*(u1*x+v1*y+w1*z)^2))
```

In case $P_2 \neq (1 : -i : 0)$ we have:

A cubic C of the plane has in its eigenpoints a conic bitanget to $\mathcal{Q}_{\text{iso}}$ if and only if C is of the form (l1*Ciso+l2*(u1*x+v1*y+w1*z)^2)*(u1*x+v1*y+w1*z)

Moreover (for any P_2), if a cubic is of the above form, the conic l1*Ciso+3*l2*(u1*x+v1*y+w1*z)^2 is contained in its eigenpoints.

1.3.2 Case $P_2 = (1 : -i : 0)$.

We start as before, so we define the points P_1, P_2, the tangent lines to $\mathcal{Q}_{\text{iso}}$ in P_1 and P_2, the pencil of conics Cg bitangent to $\mathcal{Q}_{\text{iso}}$ in P_1 and P_2, we construct three points, here called Pg1, Pg2 and Pg3 on Cg. We observe that we can assume that some used parameters are not 0

[103]:
```
P1 = vector(S, (1, ii, 0))
P2 = vector(S, (1, -ii, 0))
```

[104]:
```
## Tangent line to Ciso in P1:
rtg1 = scalar_product(P1, vector((x, y, z))-P1)



## Tangent line to Ciso in P2:
rtg2 = scalar_product(P2, vector((x, y, z))-P2)

## Pencil of conics tangent to P1 and P2 to Ciso:
Cg = Ciso + l1*rtg1*rtg2

## If l1 = -1, Cg is the conic given by (P1+P2)^2, so we can assume
## l1+1 != 0
assert(-4*Cg.subs(l1=-1) == det(matrix([P1, P2, (x, y, z)]))^2)

## construction of a generic point (different from (1, ii, 0)) on Cg:

foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]

## generic point of Cg (depends on the parameter w1):
Pg = vector(S, (foo.coefficient(z), ii*(foo.coefficient(z))+\
                     w1*(-foo.coefficient(x)), -foo.coefficient(x)))

## the last coordinate of Pg is ((-2*ii)) * (l1 + 1) * w1.
## If w1 = 0, then Pg = P1, hence we can assume w1 != 0.
assert(matrix([P1, Pg.subs(w1=0)]).rank() == 1)

## Now we define three points on Cg:

Pg1, Pg2 = Pg.subs({w1:w1}), Pg.subs({w1:w2})
Pg3 = Pg.subs({w1:m1})

## and we can assume w1, w2, m1 != 0.
```

Now we construct the matrix of conditions of the points P1, P2, Pg1, Pg2, Pg3 and we study its rank. We see that the matrix M has rank <=9.

[105]:
```
## the following matrix must have rank <= 9:

M = condition_matrix([P1, P2, Pg1, Pg2, Pg3], S, standard="all")
```

```
## Under this hypothesis, we have that we can extract from M
## the rows: 0, 1; 3, 4; 7, 8; 10, 11; 13, 14.
## (Remember that w1, w2, m1 are not zero).

MM1 = M.matrix_from_rows([0, 1, 3, 4, 7, 8, 10, 11, 13, 14])

## we make a copy of MM1
MM2 = M.matrix_from_rows([0, 1, 3, 4, 7, 8, 10, 11, 13, 14])

## Using the fact that the first two rows are good
## (MM2[0,0] is a non zero constant, and MM2[1, 4] is 1)
## we can reduce MM2 with elementary rows and columns operations.

MM2.rescale_row(0, ii/3)
for i in range(2, 10):
    MM2.add_multiple_of_row(i, 0, -MM2[i][0])


for i in range(2, 10):
    MM2.add_multiple_of_row(i, 1, -MM2[i][4])

## We extract from MM2 an order 8 square matrix,
## extracting the last 8 rows and the columns of position
## 1, 2, 3, 5, 6, 7, 8, 9.
MM2 = MM2.matrix_from_rows_and_columns([2, 3, 4, 5, 6, 7, 8, 9], \
                                       [1, 2, 3, 5, 6, 7, 8, 9])

## The computation of det(MM2) gives 0
dtMM2 = MM2.det()
assert(dtMM2 == S(0))

## Hence M and MM1 have rank <= 9.
```

Now we want to see if M can have rank < 9. We will see that M cannot have rank 8 or smaller:

[106]:
```
## We want to see when M has rank <=8, i.e. when MM2 has rank <=6
## hence we compute the ideal of the order 7-minors of MM2.
## Time of computation: 15''

mm2_7 = MM2.minors(7)

## here is a list of factors that cannot be zero:
nonZeroFt = [l1, l1+1, w1, w2, m1, w1-w2, w1-m1, w2-m1]



## here from each order 7-minor of MM2 (i.e. every element
## of mm2_7) we clear off the superfluous factors and
## we construct the ideal J7 of these polynomials (7'' of computation)

J7 = []
for ff in mm2_7:
    J7.append(prod(erase_superfluous_factors(ff, nonZeroFt)))

## since the ideal J7 is (1), we have
## that the matrix MM2 cannot have rank 6 or smaller, hence
## M cannot have rank 8 or smaller:
assert(S.ideal(J7) == S.ideal(S(1)))
```

The above computation show that the matrix M has rank 9. Therefore there is only one cubic given by M and is obtained from the determinant of the matrix Mc below. We get therefore the cubic Cbc below.

[107]:
```
Mc = MM1.matrix_from_rows(range(9))
Mc = Mc.stack(matrix([mon]))

dtMc = Mc.det()

## from dtMc we erase useless factors:

dt1 = erase_superfluous_factors(dtMc, nonZeroFt)

## We get that dt1 is a list of 4 elements, which are:
Ls = [w1 + w2, z, l1*m1^2 + m1^2 + 1, \
      x^2*l1 + y^2*l1 + 2/3*z^2*l1 + x^2 + y^2 + z^2]

assert(dt1 == Ls)

## we select the two factors which are a line and a conic and whose
## product is the desired cubic.

## The factor dt1[1] is the line r12 given by P1+P2:

r12 = -det(matrix([P1, P2, (x, y, z)]))/(2*ii)

assert(dt1[1] == r12)

## The factor Lt[3] is the l1*r12^2+3*(l1+1)*Ciso:

assert(dt1[3] == (l1+1)*Ciso-1/3*l1*r12^2)

## The cubic is therefore:

Cbc = dt1[1]*dt1[3]
```

```
assert(Cbc == r12*((l1+1)*Ciso-1/3*l1*r12^2))
```

The above cubic is of the form desired, i.e. is of the form r*(u*r^2+v*Ciso) for suitable u, v in K. Now we compute the eigenpoints of Cbc and we see that they are of the desired form:

[108]:
```
Je = S.ideal(
    matrix(
        [
            [Cbc.derivative(x), Cbc.derivative(y), Cbc.derivative(z)],
            [x, y, z]
        ]
    ).minors(2)
)

## we get two components:
PDe = Je.primary_decomposition()

assert(len(PDe) == 2)

assert(PDe[0] == S.ideal(rtg1, rtg2))

assert(PDe[1] == S.ideal(Ciso+l1*rtg1*rtg2))
```

This concludes the case $P_2 = (1 : -i : 0)$ and there are no differences w.r.t. the case $P_2 \neq (1 : -i : 0)$

1.4 Γ osculating the isotropic conic in $P_1 = (1 : i : 0)$.

Time of computations: about 800 seconds.

Here we want to see the case in which a cubic has among its eigenpoints, a conic which is osculating $\mathcal{Q}_{\text{iso}}$ in the point $P_1 = (1 : i : 0)$. We define P_2 a generic point on $\mathcal{Q}_{\text{iso}}$ different from P_1 (P_2 depends on the parameter u_1). We define the tangent lines to $\mathcal{Q}_{\text{iso}}$ in P_1 and P_2, we define the pencil of conics Cg that pass through P_2 and are osculating $\mathcal{Q}_{\text{iso}}$ in P_1. Then we define 6 “random” points $P_3, \dots, P_8$ on Cg

[109]:
```
P1 = vector(S, (1, ii, 0))
P2 = vector(S, ((-ii)*u1^2 + (-ii), u1^2 - 1, 2*u1))

## Tangent line to Ciso in P1:
rtg = scalar_product(P1, vector((x, y, z))-P1)

## line P1 + P2
r12 = matrix([P1, P2, (x, y, z)]).det()

## Pencil of conics osculating Ciso in P1 and passing through P2:
Cg = Ciso + l1*rtg*r12

## If l1 = 0, Cg is Ciso, so we can assume
## l1 != 0
assert(Cg.subs(l1=0) == Ciso)

## construction of a generic point (different from (1, ii, 0)) on Cg:

foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]

## generic point of Cg (it depends on the parameter w1):
Pg = vector(
    S,
    (
        foo.coefficient(z),
        ii*(foo.coefficient(z))+w1*(-foo.coefficient(x)),
        -foo.coefficient(x)
    )
)

## Pg is on Cg:
assert(Cg.subs(substitution(Pg)) == 0)


## If w1 = 0, then Pg = P1, hence we can assume w1 != 0.
assert(matrix([P1, Pg.subs(w1=0)]).rank() == 1)

## Now we define six specific points on Cg:

P3 = Pg.subs(w1=-1)
P4 = Pg.subs(w1=-2)
P5 = Pg.subs(w1=-3)
P6 = Pg.subs(w1=4)
P7 = Pg.subs(w1=-4)
P8 = Pg.subs(w1=-6)
```

Then we construct two matrices, MT1 and MT2 of conditions that must have rank ≤ 9. We manipulate these matrices and at the end of the block below, we have two matrices (called again MT1 and MT2) that, if $P_1, \ldots, P_8$ are eigenpoints, must have rank ≤ 8.

[110]:
```
MT1 = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
MT2 = condition_matrix([P1, P2, P6, P7, P8], S, standard="all")

## Since the 0-row of MT1 and MT2 is numeric and also the 1-row
## of MT1 and MT2 is numeric, we can use them to simplify MT1 and MT2.

assert(MT1[0] == vector(S, ((-3*ii), 3, (3*ii), -3, 0, 0, 0, 0, 0, 0)))
assert(MT2[0] == vector(S, ((-3*ii), 3, (3*ii), -3, 0, 0, 0, 0, 0, 0)))
assert(MT1[1] == vector(S, (0, 0, 0, 0, 1, ii, -1, 0, 0, 0)))
assert(MT2[1] == vector(S, (0, 0, 0, 0, 1, ii, -1, 0, 0, 0)))
```

```
MT1.rescale_row(0, ii/3)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 0, -MT1[i][0])


MT1.rescale_row(1, 1)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 1, -MT1[i][4])


MT1 = MT1.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT1 = MT1.matrix_from_rows(range(3, 15))

## same for MT2:

MT2.rescale_row(0, ii/3)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 0, -MT2[i][0])


MT2.rescale_row(1, 1)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 1, -MT2[i][4])


MT2 = MT2.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT2 = MT2.matrix_from_rows(range(3, 15))

## If P1, P2, P3, P4, P5, P6, P7, P8 are eigenpoints,
## the order 8 minors of MT1 and of MT2 must all be zero.
```

Now we compute the two ideals of the minors of order 8 of MT1 and MT2. The next computations require about 200 seconds. At the end of the computations below, we have two ideals, j1 and j2 which are such that when the parameters are a zero of j1+j2, then MT1 and MT2 have rank <=8

[111]:
```
## this computation requires about 63":
print("About 63 seconds of computation...")
sleep(1)
ttA = cputime()
mt1_8 = MT1.minors(8)
print(cputime()-ttA)


## this computation requires about 80"
print("About 80 seconds of computation...")
sleep(1)
```

```
ttA = cputime()
mt2_8 = MT2.minors(8)
print(cputime()-ttA)


## To speed up the computations, we keep separated the two ideals

## The following block requires 36 seconds.
print("Now 36 seconds of computations")
sleep(1)
JJ1 = S.ideal(mt1_8)
JJ2 = S.ideal(mt2_8)


ttA = cputime()
j1 = S.ideal(JJ1).saturation(l1)[0]

## we saturate w.r.t. the conditions that P2 and P3, P4, P5
## are different:

plSat = prod(
    [
        S.ideal(matrix([P2, pp]).minors(2)).groebner_basis()[0]
        for pp in [P3, P4, P5]
    ]
)

j1 = j1.saturation(plSat)[0]

## similarly for JJ2.
## we saturate w.r.t. the conditions that P2 and P6, P7, P8
## are different:

j2 = S.ideal(JJ2).saturation(l1)[0]

plSat = prod(
    [
        S.ideal(matrix([P2, pp]).minors(2)).groebner_basis()[0]
        for pp in [P6, P7, P8]
    ]
)

j2 = j2.saturation(plSat)[0]

print(cputime()-ttA)
```

```
About 63 seconds of computation…
41.41182299999991
```

```
About 80 seconds of computation…
46.731866999999966
Now 36 seconds of computations
25.96072499999991
```

In order to study the zeros of j1 and j2, we compute their radical, then we sum the two radicals and we get the ideal (1). This means that the matrices MT1 and MT2 cannot have rank 8 Time of computations: about 146 seconds.

[112]:
```
## We compute the radical of j1 and j2:
print("Computation of a radical: 54 seconds")
sleep(1)
ttA = cputime()
rJ1 = j1.radical()
print(cputime()-ttA)

print("computation of the second radical: 73 seconds")
sleep(1)
ttA = cputime()
rJ2 = j2.radical()
print(cputime()-ttA)


## now we sum the two ideals and we get
## the ideal (1).

print("about 15 seconds")
sleep(1)
ttA = cputime()
assert(rJ1+rJ2 == S.ideal(1))
print(cputime()-ttA)
```

```
Computation of a radical: 54 seconds
39.453751999999895
computation of the second radical: 73 seconds
52.07700100000011
about 15 seconds
10.753017999999884
```

The above computations show that it is almost sure that it is not possible to have $P_1, \ldots, P_8$ eigenpoints. In order to be sure, we have to repeat the computation for other points and to see if we get again that also these points on Cs cannot be eigenpoint for a cubic. The next block repeat therefore the computations for other points. The time of computation is 420 seconds.

[113]:
```
print("Half of the computation. Now the same computations with other points")

P3 = Pg.subs(w1=1)
P4 = Pg.subs(w1=2)
P5 = Pg.subs(w1=3)
```

```
P6 = Pg.subs(w1=5)
P7 = Pg.subs(w1=-5)
P8 = Pg.subs(w1=6)




## the following two matrices must have rank <= 9:

MT1 = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
MT2 = condition_matrix([P1, P2, P6, P7, P8], S, standard="all")

## Since the 0-row of MT1 and MT2 is numeric and also the 1-row
## of MT1 and MT2 is numeric, we can use them to simplify MT1 and MT2.

## assert(MT1[0] ==
MT1.rescale_row(0, ii/3)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 0, -MT1[i][0])


MT1.rescale_row(1, 1)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 1, -MT1[i][4])


MT1 = MT1.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT1 = MT1.matrix_from_rows(range(3, 15))

## same for MT2:
##assert(MT1[0] == MT2[0])
##assert(MT1[1] == MT2[1])

MT2.rescale_row(0, ii/3)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 0, -MT2[i][0])


MT2.rescale_row(1, 1)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 1, -MT2[i][4])


MT2 = MT2.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT2 = MT2.matrix_from_rows(range(3, 15))
```

Half of the computation. Now the same computations with other points

[114]:
```
## If P1, P2, P3, P4, P5, P6, P7, P8 are eigenpoints,
## the order 8 minors of MT1 and of MT2 must all be zero.

## this computation requires about 62":
print("About 62 seconds of computation...")
sleep(1)
ttA = cputime()
mt1_8 = MT1.minors(8)
print(cputime()-ttA)

## this computation requires about 97"
print("About 97 seconds of computation...")
sleep(1)
ttA = cputime()
mt2_8 = MT2.minors(8)
print(cputime()-ttA)
```

```
About 62 seconds of computation…
35.11327500000016
About 97 seconds of computation…
51.91341700000021
```

[115]:
```
## To speed up the computations, we keep separated the two ideals

## The following block requires 46 seconds.
print("Now 46 seconds of computations")
sleep(1)
JJ1 = S.ideal(mt1_8)
JJ2 = S.ideal(mt2_8)

ttA = cputime()
j1 = S.ideal(JJ1).saturation(l1)[0]
```

```
Now 46 seconds of computations
```

[116]:
```
## we saturate w.r.t. the conditions that P2 and P3, P4, P5
## are different:

plSat = prod(
    [
        S.ideal(matrix([P2, pp]).minors(2)).groebner_basis()[0]
        for pp in [P3, P4, P5]
    ]
)

j1 = j1.saturation(plSat)[0]
```

[117]:
```
## similarly for JJ2.
## we saturate w.r.t. the conditions that P2 and P6, P7, P8
## are different:

j2 = S.ideal(JJ2).saturation(l1)[0]

plSat = prod(
    [
        S.ideal(
            matrix([P2, pp]).minors(2)
        ).groebner_basis()[0]
        for pp in [P6, P7, P8]
    ]
)

j2 = j2.saturation(plSat)[0]

print(cputime()-ttA)
```

```
32.323909000000185
```

[118]:
```
## We compute the radical of j1 and j2:
print("Computation of a radical: 55 seconds")
sleep(1)
ttA = cputime()
rJ1 = j1.radical()
print(cputime()-ttA)

print("computation of the second radical: 116 seconds")
sleep(1)

ttA = cputime()
rJ2 = j2.radical()
print(cputime()-ttA)
```

```
Computation of a radical: 55 seconds
38.51320700000042
computation of the second radical: 116 seconds
79.28385499999968
```

[119]:
```
## now we sum the two ideals and we get
## the ideal (1).

print("about 20 seconds")
sleep(1)
ttA = cputime()
assert(rJ1+rJ2 == S.ideal(1))
print(cputime()-ttA)
```

```
about 20 seconds
13.807926000000407
```

Conclusion of the above comptations: it is not possible to have a cubic whose eigenpoints contain a conic which is osculating the isotropic conic in a point.

1.5 Γ iperosculating the isotropic conic $\mathcal{Q}_{\mathrm{iso}}$ in $P_1 = (1 : i : 0)$

We want to see now if there are cubics such that among their eigenpoints have a conic which is iperosculating $\mathcal{Q}_{\mathrm{iso}}$ in a point that, as usual, can be the point $P_1 = (1 : i : 0)$. The computations are similar to the computations in the previous case:

We construct the pencil of conics Cg iperosculating $\mathcal{Q}_{\mathrm{iso}}$ in P_1, we define 8 further points "random" on Cg and we construct two matrices MT1 and MT2 whose rank must be ≤ 9. We study the ideal obtained from these conditions.

```
[120]: P1 = vector(S, (1, ii, 0))
```

```
[121]: ## Tangent line to Ciso in P1:
rtg = scalar_product(P1, vector((x, y, z))-P1)
```

```
[122]: ## Pencil of conics iperosculating Ciso in P1
Cg = Ciso + l1*rtg^2
```

```
[123]: ## If l1 = 0, Cg is Ciso, so we can assume
## l1 != 0
assert(Cg.subs(l1=0) == Ciso)
```

```
[124]: ## construction of a generic point (different from (1, ii, 0)) on Cg:

foo = Cg.subs(y=ii*x+w1*z).factor()[-1][0]
```

```
[125]: ## generic point of Cg (it depends on the parameter w1):
Pg = vector(
    S,
    (
        foo.coefficient(z),
        ii*(foo.coefficient(z)) + w1*(-foo.coefficient(x)),
        -foo.coefficient(x)
    )
)
```

```
[126]: ## Pg is on Cg:
assert(Cg.subs(substitution(Pg)) == 0)
```

```
[127]: ## If w1 = 0, then Pg = P1, hence we can assume w1 != 0.
assert(matrix([P1, Pg.subs(w1=0)]).rank() == 1)
```

```
[128]: ## Now we define eight specific points on Cg:
```

[129]:
```
P2 = Pg.subs(w1=-1)
P3 = Pg.subs(w1=-2)
P4 = Pg.subs(w1=-3)
P5 = Pg.subs(w1=3)
P6 = Pg.subs(w1=1)
P7 = Pg.subs(w1=2)
P8 = Pg.subs(w1=5)
P9 = Pg.subs(w1=7)
```

[130]:
```
## the following two matrices must have rank <= 9:

MT1 = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
MT2 = condition_matrix([P1, P6, P7, P8, P9], S, standard="all")
```

We manipulate MT1 and MT2 in order to see if it is possible that their rank is ≤ 9. At the end of these computations, we see that we have to study the condition given by the groebner basis gb which is l1*(l1 + 3) (about 10 seconds of computations)

[131]:
```
MT1.rescale_row(0, ii/3)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 0, -MT1[i][0])


MT1.rescale_row(1, 1)
for i in range(2, 10):
    MT1.add_multiple_of_row(i, 1, -MT1[i][4])

MT1 = MT1.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT1 = MT1.matrix_from_rows(range(3, 15))

m8 = MT1.minors(8)

I = S.ideal(m8)

MT2.rescale_row(0, ii/3)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 0, -MT2[i][0])


MT2.rescale_row(1, 1)
for i in range(2, 10):
    MT2.add_multiple_of_row(i, 1, -MT2[i][4])

MT2 = MT2.matrix_from_columns([1, 2, 3, 5, 6, 7, 8, 9])
MT2 = MT2.matrix_from_rows(range(3, 15))

m8 = MT2.minors(8)
```

```
J = S.ideal(m8)

gb = (I+J).groebner_basis()

assert(gb == [l1*(l1 + 3)])
```

Here we see that we have to study the case l1 = -3. We study the cubic we obtain and we see it is of the form $t(\mathcal{Q}_{iso} - t^2)$, while the eigenpoints are given by $\mathcal{Q}_{iso} - 3t^2$ (t is the tangent line)

[132]:
```
# we particularize Cg  and we define Cs:

Cs = Cg.subs({l1:-3})
## we define the matrix of conditions for the case l1 = -3:
M = condition_matrix([P1, P2, P3, P4, P5], S, standard = 'all').subs({l1:-3})
assert(M.rank() == 9)

# select a suitable 9x10 submatrix of rank 9
N = M.matrix_from_rows([0,1,3,4,6,7,9,10,12])
assert(N.rank() == 9)

# determine the cubic from M (and N)
C = N.stack(vector(S, mon)).det()

pd = (S.ideal(list(eig(C)))).primary_decomposition()

## we get two ideals. One is the double point P_1

assert(len(pd) == 2)

assert(pd[1] == S.ideal(z, x^2 + (2*ii)*x*y - y^2))

## the other ideal is the conic Cs:
assert(pd[0] == S.ideal(Cs))

## The cubic C is of the form rtg*(Ciso-rtg^2)
## the conic Cs is of the form Ciso-3*rtg^2:
assert(S.ideal(C) == S.ideal(rtg*(Ciso-rtg^2)))
assert(S.ideal(Cs) == S.ideal(Ciso-3*rtg^2))

## Moreover, rtg*(1/2*Cs + Ciso)) and rtg*(Ciso - rtg^2)
## are the same cubic
assert(S.ideal(rtg*(1/2*Cs + Ciso)) == S.ideal(rtg*(Ciso - rtg^2)))
```

The conclusion of this computation is that if a cubic C has among the eigenpoints a conic which is iperosculating $\mathcal{Q}_{\rm iso}$ in a point P_1, then the cubic is of the form $t(\mathcal{Q}_{iso} - t^2)$ and the conic of eigenpoints is $\mathcal{Q}_{iso} - 3t^2$. The converse is also true and is immediately verified, because is given by the computation of the ideal pd above, which is the ideal of the eigenpoints of the cubic $t(\mathcal{Q}_{iso} - t^2)$.

This concludes the proof of the theorem.