NB.04.F1

July 23, 2024

1 Proposition

If we have a V-configuration of five points $P_1, \dots, P_5$ such that the rank of the matrix $\Phi(P_1, \dots, P_5)$ is 9 and such that $\delta_2(P_1, \dots, P_5) = 0$, then the unique ternary cubic determined by the condition that $P_1, \dots, P_5$ are eigenpoints has also the two other eigenpoints P_6 and P_7 aligned with P_1.

[88]:
```
load("basic_functions.sage")
```

1.1 Case $s_{12} = 0$ and $s_{14} = 0$;

1.1.1 Subcase $P_1 = (1 : 0 : 0)$

[89]:
```
P1 = vector(S, (1, 0, 0))
P2 = vector(S, (0, B2, C2))
P3 = u1*P1 + u2*P2
P4 = vector(S, (0, B4, C4))
P5 = v1*P1 + v2*P4
```

[90]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
```

[91]:
```
assert(M[2] == vector([0 for _ in range(10)]))
```

[92]:
```
M1 = M.matrix_from_rows([0,1,3,5,6,7,9,10,12,13])
```

[93]:
```
M2 = M.matrix_from_rows([0,1,3,5,6,7,9,11,12])
```

[94]:
```
G1 = M2.stack(phi((x,y,z), S)[0])
G2 = M2.stack(phi((x,y,z), S)[1])
G3 = M2.stack(phi((x,y,z), S)[2])

g1 = G1.det()
g2 = G2.det()
g3 = G3.det()
```

[95]:
```
g = P1[2]*g1 - P1[1]*g2 + P1[0]*g3
assert(g == g3)
```

[96]:
```
factors_g = g.factor()
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
↪p[0].degree(z) > 0, factors_g))
candidate_line = xyz_factors[-1][0]
```

[97]:
```
assert(candidate_line.subs(substitution(P1)).is_zero())
```

1.1.2 Subcase $P_1 = (1 : i : 0)$

[98]:
```
P1 = vector(S, (1, ii, 0))
P2 = vector(S, (0, B2, C2))
P3 = u1*P1 + u2*P2
P4 = vector(S, (0, B4, C4))
P5 = v1*P1 + v2*P4
```

[99]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
```

[100]:
```
M1 = M.matrix_from_rows([0,2,3,5,6,7,9,10,12,13])
```

[101]:
```
M2 = M.matrix_from_rows([0,2,3,5,6,7,9,11,12])
```

[102]:
```
G1 = M2.stack(phi((x,y,z), S)[0])
G2 = M2.stack(phi((x,y,z), S)[1])
G3 = M2.stack(phi((x,y,z), S)[2])

g1 = G1.det()
g2 = G2.det()
g3 = G3.det()
```

[103]:
```
g = P1[2]*g1 - P1[1]*g2 + P1[0]*g3
```

[104]:
```
factors_g = g.factor()
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
↪p[0].degree(z) > 0, factors_g))
candidate_line = xyz_factors[-1][0]
```

[105]:
```
assert(candidate_line.subs(substitution(P1)).is_zero())
```

1.2 Case $s_{12} = 0$ and $s_{22} = 0$;

This implies that P_2 is on the isotropic conic, so we suppose $P_1 = (1 : 0 : 0)$ and P_2 one of the two intersections of the isotropic conic with the two tangents from P_1.

[123]:
```
P1 = vector(S, (1, 0, 0))
P2 = vector(S, (0, 1, ii))
P3 = u1*P1 + u2*P2
P4 = vector(S, (A4, B4, C4))
P5 = v1*P1 + v2*P4
```

[124]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
```

[156]:
```
M2 = M.matrix_from_rows([0,1,3,5,6,9,10,12,13])
```

[157]:
```
G1 = M2.stack(phi((x,y,z), S)[0])
G2 = M2.stack(phi((x,y,z), S)[1])
G3 = M2.stack(phi((x,y,z), S)[2])

g1 = G1.det()
g2 = G2.det()
g3 = G3.det()
```

[158]:
```
g = P1[2]*g1 - P1[1]*g2 + P1[0]*g3
```

[159]:
```
factors_g = g.factor()
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
↪p[0].degree(z) > 0, factors_g))
candidate_line = xyz_factors[-1][0]
```

[162]:
```
assert(candidate_line.subs(substitution(P1)).is_zero())
```

1.3 Case $\sigma(P_1, P_2) = 0$ and $\sigma(P_1, P_4) = 0$;

This implies that P_1 cannot be on the isotropic conic, so we can suppose $P_1 = (1 : 0 : 0)$ (indeed, P_1, P_2, P_4 cannot be aligned)

[163]:
```
P1 = vector(S, (1, 0, 0))
Qa = vector(S, (0, 1, ii))
Qb = vector(S, (0, 1, -ii))

P2 = m1*P1 + m2*Qa
P4 = l1*P1 + l2*Qb

assert(sigma(P1, P2) == 0)
assert(sigma(P1, P4) == 0)

P3 = u1*P1 + u2*P2
P5 = v1*P1 + v2*P4
```

[164]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
```

[165]:
```
M1 = M.matrix_from_rows([0,1,3,5,6,7,9,10,12,13])
```

[ ]:
```
M2 = M.matrix_from_rows([0,1,3,5,6,7,9,11,12])
```

In this case the eigenscheme of f is not regular (it contains two lines):

[ ]:
```
cb = M2.stack(vector(S, mon)).det()
Ecb = eig(cb)
assert(Ecb.subs({y: -ii*z}) == vector(S, (0, 0, 0)))
assert(Ecb.subs({y: ii*z}) == vector(S, (0, 0, 0)))
```

1.4 Case $P_3 = (s_{14}s_{15}s_{22} - s_{12}^2 s_{45})P_1 + s_{12}(s_{11}s_{45} - s_{14}s_{15})P_2$

1.4.1 Subcase $P_1 = (1 : 0 : 0)$

[4]:
```
P1 = vector(S, (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
```

[6]:
```
d2 = delta2(P1, P2, P3, P4, P5)
U1 = d2.coefficient(u1)
U2 = d2.coefficient(u2)
```

[7]:
```
p3 = P3.subs({u1: U2, u2: -U1})
```

[8]:
```
M = condition_matrix([P1, P2, p3, P4, P5], S, standard="all")
```

[17]:
```
assert(M[2] == vector([0 for _ in range(10)]))
```

[20]:
```
M1 = M.matrix_from_rows([0,1,3,4,6,7,9,10,12,13])
```

[22]:
```
M2 = M.matrix_from_rows([0,1,3,4,6,7,9,10,12])
```

[28]:
```
G1 = M2.stack(phi((x,y,z), S)[0])
G2 = M2.stack(phi((x,y,z), S)[1])
G3 = M2.stack(phi((x,y,z), S)[2])

g1 = G1.det()
g2 = G2.det()
g3 = G3.det()
```

[33]:
```
g = P1[2]*g1 - P1[1]*g2 + P1[0]*g3
assert(g == g3)
```

[39]:
```
factors_g = g.factor()
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
  ↪p[0].degree(z) > 0, factors_g))
candidate_line = xyz_factors[-1][0]
```

[41]:
```
assert(candidate_line.subs(substitution(P1)).is_zero())
```

1.4.2 Subcase $P_1 = (1 : i : 0)$

[18]:
```
P1 = vector(S, (1, ii, 0))
P2 = vector(S, (A2, B2, C2))
P3 = u1*P1 + u2*P2
P4 = vector(S, (A4, B4, C4))
P5 = v1*P1 + v2*P4
```

[19]:
```
d2 = delta2(P1, P2, P3, P4, P5)
U1 = d2.coefficient(u1)
U2 = d2.coefficient(u2)
```

[20]:
```
p3 = P3.subs({u1: U2, u2: -U1})
```

[21]:
```
M = condition_matrix([P1, P2, P3, P4, P5], S, standard="all")
```

[22]:
```
M1 = M.matrix_from_rows([0,1,3,4,6,7,9,10,12,13])
```

[ ]:
```
M2 = M.matrix_from_rows([0,1,3,4,6,7,9,10,12])
```

[ ]:
```
G1 = M2.stack(phi((x,y,z), S)[0])
G2 = M2.stack(phi((x,y,z), S)[1])
G3 = M2.stack(phi((x,y,z), S)[2])

g1 = G1.det()
g2 = G2.det()
g3 = G3.det()
```

[27]:
```
g = P1[2]*g1 - P1[1]*g2 + P1[0]*g3
```

[47]:
```
gtilde = g.quo_rem(matrix([P1, P2, P4]).det()^2)[0]
gtilde = gtilde.quo_rem(u1^2)[0]
gtilde = gtilde.quo_rem(u2^2)[0]
gtilde = gtilde.quo_rem(v1*v2)[0]
factors_g = gtilde.factor()
```

[48]:
```
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
↪p[0].degree(z) > 0, factors_g))
```

[53]:
```
quadratic_factor = xyz_factors[1][0]
```

[57]:
```
quadratic_factor = quadratic_factor.subs({u1: U2, u2: -U1})
xyz_factors = list(filter(lambda p: p[0].degree(x) > 0 or p[0].degree(y) > 0 or␣
↪p[0].degree(z) > 0, quadratic_factor.factor()))
```

[58]:
```
candidate_line = xyz_factors[-1][0]
```

[59]:
```
assert(candidate_line.subs(substitution(P1)).is_zero())
```