April 2021 version 3

Note on Intersection of rational curves

Barry H Dayton

(https : // barryhdayton.space)

One application of implicitization not mentioned in Section 3.1 of my *Space Curve Book* or *Degree vs Dimension* Mathematical Journal article is the intersection of rational curves. Overall there seems to be a lack of material on this topic in the literature. This note corrects that, however it will not be incorpo - rated into the *Space Curve Book* because I am viewing this not in the context of affine real curves but as complex projective curves and applications lie elsewhere. I am still working numerically of course.

For a simple example let

In[ ]:= $L1 = \left\{\frac{2\,t}{1+t^2}, \frac{1-t^2}{1+t^2}\right\};$

```
L2 = {2 + t, 2 t};
```

Then these curves meet at points

```
p1 = {1.5999999999999996` + 0.6633249580710802` ⅈ,
    -0.8000000000000002` + 1.3266499161421594` ⅈ};
p2 = {1.5999999999999999` - 0.66332495807108` ⅈ,
    -0.8000000000000002` - 1.32664991614216` ⅈ};
```

Let

```
In[ ]:= α1 = -0.4000000000000001` + 0.66332495807108` ⅈ;
α2 = -0.4000000000000001` - 0.66332495807108` ⅈ;
β1 = 0.6666666666666669` - 1.1055415967851334` ⅈ;
β2 = 0.6666666666666669` + 1.1055415967851334` ⅈ;
```

```
In[ ]:= L1 /. {t → β1}
L2 /. {t → α1}
Out[ ]= {1.6 + 0.663325 ⅈ, -0.8 + 1.32665 ⅈ}
Out[ ]= {1.6 + 0.663325 ⅈ, -0.8 + 1.32665 ⅈ}
```

```
In[ ]:= L1 /. {t → β2}
L2 /. {t → α2}
Out[ ]= {1.6 - 0.663325 ⅈ, -0.8 - 1.32665 ⅈ}
Out[ ]= {1.6 - 0.663325 ⅈ, -0.8 - 1.32665 ⅈ}
```

Notice that the parameter values for the two curves differ at the intersection points. By intersection we mean point wise, that is the two parametric curves have a common point in their range not that the parameterized curves have the same value for a given t.

The fact that these curves above, normally considered real curves, have a common point is partly a consequence of Bézout's theorem. It is well known that parameterized curves are algebraic, that is satisfy an implicit equation, we will show this explicitly later (this is also in my space curve book and Mathematica Journal paper). In the example above one is a circle and the other is a line, both plane algebraic curves one of degree 1 and one of degree 2. So Bézout's theorem requires these implicit curves to have two complex projective points in common. A caveat is that the range of a parameterized curve need not be the complete implicit curve. In the case above both parameterized curves are missing a point in their implicitizations, sometimes more is missing. In this example the intersection points are not among the missing. Unlike the plane, in 3-space and higher it is not expected that two curves, parameterized or not, will have a common point.

The purpose of this note is to explore several methods for deciding if there is a common complex projective point of the parameterizations and, if so, find the point and the parameter values. We will see later that Bézout's theorem, albeit the one in my Space Curve Book, will still have something to say about it in higher dimensions.

The motivation for studying this material comes from the more general problem of finding rational curves in surfaces. Unlike the plane, rational curves in a two dimensional surface are not guaranteed to intersect but often do in predicable configurations. Efficient calculation of these intersections often helps in finding additional curves suggested by the configuration. A number of the examples here come from my investigations of this phenomena. The topics are

1. Brute force method
2. Affine linear case
3. Implicitization
4. Hybrid method
5. Projection and lifting
6. Multiplicity of Intersections

1. Brute force method.

The simplest way to find the common point above is to write the two plane curves with separate named parameters and then solve for a common point with, say, **NSolve.** It helps that we have the same number of coordinates as variables.

In[ ]:=
$$L1t = \left\{\frac{2\,t}{1+t^2}, \frac{1-t^2}{1+t^2}\right\};$$

```
L2s = {2 + s, 2 s};
```

In[ ]:=
```
sol = NSolve[L1t - L2s]
```
Out[ ]= {{s → -0.4 + 0.663325 ⅈ, t → 0.666667 - 1.10554 ⅈ},
 {s → -0.4 - 0.663325 ⅈ, t → 0.666667 + 1.10554 ⅈ}}

In[ ]:=
```
L1t /. sol[[1]]
L2s /. sol[[1]]
```
Out[ ]= {1.6 + 0.663325 ⅈ, -0.8 + 1.32665 ⅈ}

Out[ ]= {1.6 + 0.663325 ⅈ, -0.8 + 1.32665 ⅈ}

shows the common value.

Actually note the built in Mathematica function **FindInstance** works directly in this exact situation giving an exact answer

In[ ]:=
```
FindInstance[L1t == L2s, {t, s}, 2]
```
Out[ ]= $\left\{\left\{t \to \frac{1}{3}\left(2 - i\sqrt{11}\right),\ s \to \frac{1}{5} i \left(2 i + \sqrt{11}\right)\right\},\ \left\{t \to \frac{1}{3}\left(2 + i\sqrt{11}\right),\ s \to -\frac{1}{5} i \left(-2 i + \sqrt{11}\right)\right\}\right\}$

This may not work so well for space curves or numerical curves. Consider the following

In[ ]:=
```
f1t = {4.4577934452533` + 1.2005375928194961` t,
  -5.1000087955131415` - 1.4720419842010817` t,
  3.0076111620619237` + 1.2715043913815856` t}
f2s = {-0.25754825919834656` - 1.4720419842150836` s, 0.5084795140223721` +
   1.2005375928244255` s, -1.1751566551004844` + 1.271504391390658` s}
```
Out[ ]= {4.45779 + 1.20054 t, -5.10001 - 1.47204 t, 3.00761 + 1.2715 t}

Out[ ]= {-0.257548 - 1.47204 s, 0.50848 + 1.20054 s, -1.17516 + 1.2715 s}

In[ ]:=
```
NSolve[f1t - f2s]
```
Out[ ]= {}

FindInstance is no better

In[ ]:=
```
FindInstance[f1t == f2s, {t, s}]
```
RowReduce : Result for RowReduce of badly conditioned matrix {{1.47204 , 1.20054 , 4.71534 }, {-1.20054 , -1.47204 , -5.60849 }, {-1.2715 , 1.2715 , 4.18277 }} may contain significant numerical errors .

Out[ ]= {}

There is no solution but in fact, numerically, there is an intersection point. The main problem here is that there is a small numerical error that makes **NSolve** think these systems are inconsistent. The brute force procedure here is to look at the consistent projection to the first two coordinates and then test the solutions on the third component.

In[ ]:=
```
sol2 = NSolve[Take[f1t - f2s, 2]]
```
Out[ ]= {{s → -0.286625 , t → -3.57625}}

In[ ]:=
```
p1 = f1t /. sol2[[1]]
p2 = f2s /. sol2[[1]]
```

Out[ ]= {0.164376, 0.164376, -1.5396}

Out[ ]= {0.164376, 0.164376, -1.5396}

So it appears that these are the same point. To check further

In[ ]:=
```
p1 - p2
```

Out[ ]= $\{-4.44089\times10^{-16}, -2.77556\times10^{-16}, -9.25575\times10^{-11}\}$

So the third coordinates are close but not close enough for **NSolve**. We can refine this using a few steps of Gauss-Newton

In[ ]:=
```
ts0 = {t, s} /. sol2[[1]];
ts1 = gaussNewtonMD[f1t - f2s, ts0, {t, s}, 3]
```

» Z {-3.57625, -0.286625}

» {Change, SVL, Residue}= $\{5.03258\times10^{-11}, \{2.67258, 1.81856\}, 1.38179\times10^{-11}\}$

» Z {-3.57625, -0.286625}

» {Change, SVL, Residue}= $\{2.83103\times10^{-16}, \{2.67258, 1.81856\}, 1.38179\times10^{-11}\}$

» Z {-3.57625, -0.286625}

» {Change, SVL, Residue}= $\{2.83103\times10^{-16}, \{2.67258, 1.81856\}, 1.38183\times10^{-11}\}$

Out[ ]= {-3.57625, -0.286625}

Now

In[ ]:=
```
p1g = f1t /. {t -> ts1[[1]]}
p2g = f2s /. {s → ts1[[2]]}
```

Out[ ]= {0.164376, 0.164376, -1.5396}

Out[ ]= {0.164376, 0.164376, -1.5396}

In[ ]:=
```
p1g - p2g
```

Out[ ]= $\{-9.66099\times10^{-12}, -9.6621\times10^{-12}, -2.06279\times10^{-12}\}$

so the first two coordinates are give larger residues but the last coordinate gives an equivalently accurate value. This is the best we can do using machine numbers.

When one or both of the parameterized curves is not polynomial then, as in my book and paper, I will assume there is a common denominator for each curve, they do not need to be the same. Then we can look at the curves as projective polynomial curves by adding a last coordinate which gives the denominators. For my opening example I would have

f3t = {2 t, 1 - t^2, 1 + t^2}

f4s = {2 + s, 2 s, 1}

However as f3t is projective any constant multiple will be the same curve so I need a homogenizing variable, say κ, to make it homogenous. In the second case that variable can just be 1 because $1^m = 1$. So I get

In[ ]:=
```
f3tκ = {2 t κ, κ^2 - t^2, κ^2 + t^2}
f4s = {2 + s, 2 s, 1}
```

Out[ ]= $\{2\,t\,\kappa,\ -t^2+\kappa^2,\ t^2+\kappa^2\}$

Out[ ]= {2 + s, 2 s, 1}

I now have 3 variables to play with so I can go brute force.

In[ ]:=
```
sol3 = NSolve[f3tκ - f4s]
```

Out[ ]=
```
{{s → -0.4 - 0.663325 i, t → -1.0045 - 0.330177 i, κ → -0.620814 + 0.534238 i},
 {s → -0.4 + 0.663325 i, t → -1.0045 + 0.330177 i, κ → -0.620814 - 0.534238 i},
 {s → -0.4 + 0.663325 i, t → 1.0045 - 0.330177 i, κ → 0.620814 + 0.534238 i},
 {s → -0.4 - 0.663325 i, t → 1.0045 + 0.330177 i, κ → 0.620814 - 0.534238 i}}
```

There are 4 solutions but notice both are multiple so there are only 2 distinct solutions

In[ ]:=
```
f3tκ /. sol3[[1]]
f4s /. sol3[[1]]
```

Out[ ]= $\{1.6 - 0.663325\,i,\ -0.8 - 1.32665\,i,\ 1. + 1.11022\times 10^{-16}\,i\}$

Out[ ]= {1.6 - 0.663325 i, -0.8 - 1.32665 i, 1}

and

In[ ]:=
```
f3tκ /. sol3[[2]]
f4s /. sol3[[2]]
```

Out[ ]= $\{1.6 + 0.663325\,i,\ -0.8 + 1.32665\,i,\ 1. - 1.11022\times 10^{-16}\,i\}$

Out[ ]= {1.6 + 0.663325 i, -0.8 + 1.32665 i, 1}

Since the last coordinates of all of these are 1 the affine specializations are jus the truncations, eg,

In[ ]:=
```
Take[{1.5999999999999996` - 0.6633249580710795` i,
   -0.8000000000000005` - 1.326649916142159` i, 1}, 2]
Take[{1.5999999999999996` + 0.6633249580710795` i,
   -0.8000000000000005` + 1.326649916142159` i, 1}, 2]
```

Out[ ]= {1.6 - 0.663325 i, -0.8 - 1.32665 i}

Out[ ]= {1.6 + 0.663325 i, -0.8 + 1.32665 i}

This is, of course, the same as we got above.

We now consider the case of infinite points. Since a rational curve will have a small number of infinite points it is easiest to calculate the infinite points of each curve separately and then compare to see if any are the same.

Polynomial curves have a unique infinite point where *t* goes to ±∞. Since for large *t* each component is dominated by the highest degree we need use only the term with highest degree. But the value of points will be dominated by the largest component. So we actually need only terms of the highest degree of the polynomial parameterization. So for example the curve

{2t^3-t^2+t-3,-t^2+t+3,-4t^3+t^2-3t+2}

has infinite points

{2, 0, -4, 0} and {-2, 0, 4, 0}.

Next consider the curve

```
In[ ]:= g = {t / (t ^ 2 + t - 1), t ^ 2 / (t ^ 2 + t - 1), t ^ 3 / (t ^ 2 + t - 1)}
```

Out[ ]= $\left\{\frac{t}{-1+t+t^2}, \frac{t^2}{-1+t+t^2}, \frac{t^3}{-1+t+t^2}\right\}$

We can get infinite points either from zeros of the denominator or where the degree of the numerator exceeds that of the denominator, eg. the last coordinate. So clearly {0,0,1,0} is an infinite point. Setting the denominator to zero gives

```
In[ ]:= {s1, s2} = t /. NSolve[t ^ 2 + t - 1]
Out[ ]= {-1.61803, 0.618034}
```

Homogenizing and putting in homogenous coordinates

```
In[ ]:= gh = {u ^ 2 t, u t ^ 2, t ^ 3, -u ^ 2 + u t + t ^ 2}
```

Out[ ]= $\{t\,u^2, t^2\,u, t^3, t^2 + t\,u - u^2\}$

When u = t/s1 we have

```
In[ ]:= t ^ 2 + t ^ 2 / s1 - t ^ 2 / s1 ^ 2
Out[ ]= 0.
```

```
In[ ]:= gh1 = {t ^ 3 / s1 ^ 2, t ^ 3 / s1, t ^ 3, 0}
```

Out[ ]= $\{0.381966\,t^3, -0.618034\,t^3, t^3, 0\}$

So the infinite point is

```
In[ ]:= {s1 ^ -2, s1 ^ -1, 1, 0}
Out[ ]= {0.381966, -0.618034, 1, 0}
```

When u = 1/s2 we have

```
In[ ]:= t ^ 2 + t ^ 2 / s2 - t ^ 2 / s2 ^ 2
```

Out[ ]= $-4.44089\times10^{-16}\,t^2$

```
In[ ]:= gh2 = {t ^ 3 / s2 ^ 2, t ^ 3 / s2, t ^ 3, 0}
```

Out[ ]= $\{2.61803\,t^3, 1.61803\,t^3, t^3, 0\}$

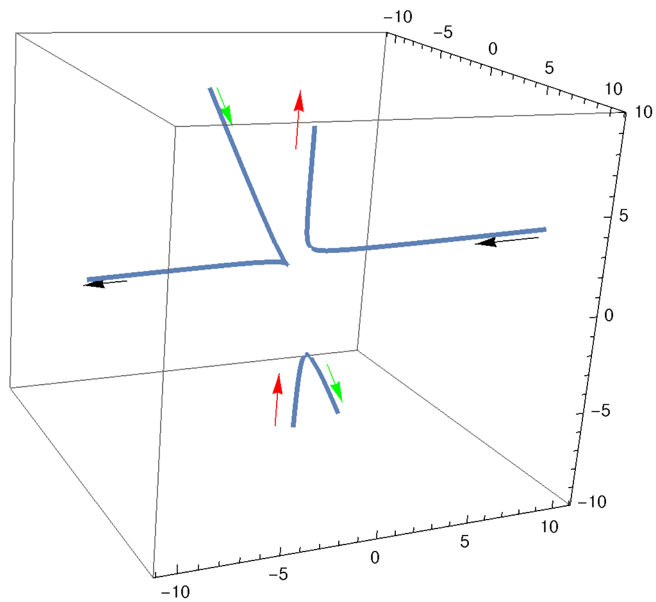
which gives infinite point

In[ ]:= `{s2^-2, s2^-1, 1, 0}`

Out[ ]= `{2.61803, 1.61803, 1, 0}`

A plot of this example where the red, green and black arrows give the approximate directions of the curves going to and from infinite points.

```
In[ ]:= Show[ParametricPlot3D[g, {t, -8, 8}, PlotRange → {-10, 10}], Graphics3D[
  {{Red, Arrowheads[Medium], Arrow[{{0, 0, 6}, {0, 0, 9}}], Arrow[{{0, 0, -9}, {0, 0, -6}}]},
   {Green, Arrowheads[Medium], Arrow[{10 {1/s1^2, 1/s1, 1}, 8 {1/s1^2, 1/s1, 1}}],
    Arrow[{-7 {1/s1^2, 1/s1, 1}, -10 {1/s1^2, 1/s1, 1}}]},
   {Black, Arrowheads[Medium], Arrow[{4.3 {1/s2^2, 1/s2, 1}, 3.3 {1/s2^2, 1/s2, 1}}],
    Arrow[{-3.3 {1/s2^2, 1/s2, 1}, -4.3 {1/s2^2, 1/s2, 1}}]}}]]
```



Since this is a parametric curve, in the projective 3-space following the arrows it is just a loop, specifically it is a pseudo-line, see section 1.3 of my Space Curve book.

2. The Affine Linear Case

As an alternate to the brute force method we can implicitize both curves and directly solve for common points, then work backwards, if necessary, to find the parameter values. As explained above these might not actually exist. The reason for looking at this separately is that we will not be using NSolve which does not work with overdetermined numerical systems.

The affine linear case is much simpler since we can use numerical linear algebra where we can control the precision and use simpler implitizing methods from my Space curve Book. We do that first.

First we recall from Section 1.1 of my *Plane Curve Book* that we have a function that finds the implicit equation of a plane line from two points, one of which can be infinite. So given, say, parametric line {1+2t,3+4t} then {1, 3} is the point where t=0 and {2,4} is the tangent direction which we viewed as the infinite point {2,4,0}. So our implicit equation is

In[ ]:=
```
line2D[{1, 3}, {2, 4, 0}, x, y]
```
Out[ ]=
```
0.759336 + 1.51867 x - 0.759336 y
```

This works also in MD, so given a parametric line in 4 dimensions

In[ ]:=
```
parl1 = {1 + 2 t, 3 + 4 t, 5 + 6 t, 7 + 8 t}
p1 = parl1 /. {t → 0}
q1 = Append[(parl1 - p1) /. {t → 1}, 0]
eq1 = lineMD[p1, q1, {x, y, z, w}]
```
Out[ ]=
```
{1 + 2 t, 3 + 4 t, 5 + 6 t, 7 + 8 t}
```
Out[ ]=
```
{1, 3, 5, 7}
```
Out[ ]=
```
{2, 4, 6, 8, 0}
```
Out[ ]=
```
{-0.18234 - 0.432366 w - 0.508102 x + 0.0368185 y + 0.72131 z,
 -0.436537 + 0.309304 w - 0.663355 x + 0.326401 y - 0.408888 z,
 0.419747 - 0.342623 w + 0.0423441 x + 0.831929 y - 0.111904 z}
```

we get a system of 3 equations in 4 variables. We can do the same for the second parametric line and join the 2 equations. Rather than trying to solve 6 equations in 4 unknowns we think of homogenizing this system and solving for 0, which is just finding the nullspace of the 6×5 Sylvester matrix of order 1 which has rank 4. We work carefully with the numerics by using the SVD getting a vector **nl** of length 5 which is a point in projective 4-space with last coordinate 0. If the first variable of **nl** is not zero there is an affine answer which is obtained by dividing **nl** by its first component and then discarding the resulting 1. If the rank of the Sylvester matrix was not 4 either there is no solution (rank=5) or possibly several solutions or even the lines could be the same.

Here is the code, preceded by two linear algebra subroutines. These have been added to my Global-FunctionsMD.

In[ ]:=
```
matrixrank[M_, tol_] := Module[{s, k, l},
   s = SingularValueList[N[M], Tolerance -> 0];
   If[s[[1]] < tol, Return[0]];
   l = Length[s];
   s = s / s[[1]];
   k = 1;
   While[k <= l, If[s[[k]] < tol, Return[k - 1], k++]];
   k - 1];
```

In[ ]:=
```
nullspace[M_, tol_] :=
 Take[SingularValueDecomposition[M][[3]], All, -(Dimensions[M][[2]] - matrixrank[M, tol])]
```

In[ ]:=
```
pLineIntersectionMD [L1_, L2_, t_, X_, tol_] :=
 Module[{n, cr1, cr2, p1, p2, v1, v2, eq1, eq2, S, r, ans},
  n = Length[X];
  If[Length[L1] ≠ n, Echo["Line 1 error"]; Abort[]];
  If[Length[L2] ≠ n, Echo["Line 2 error"]; Abort[]];
  p1 = Chop[L1 /. {t → 0}];
  v1 = Append[Chop[(L1 - p1) /. {t → 1}], 0];
  eq1 = lineMD[p1, v1, X];
  p2 = Chop[L2 /. {t → 0}];
  v2 = Append[Chop[(L2 - p2) /. {t → 1}], 0];
  eq2 = lineMD[p2, v2, X];
  S = sylvesterMD [Join[eq1, eq2], 1, X];
  r = matrixrank [S, tol];
  If[r < n, Return[Fail]];
  If[r > n, Return[{}]];
  ans = Flatten[nullspace [S, tol]];
  If[Abs[ans[[1]]] < tol, RotateLeft [Chop[ans, tol], 1], Take[ans / ans[[1]], -n]]
 ]
```

As an example we take the following parametric line and find its intersection with the line **parl1** above.

In[ ]:=
```
parl2 = {4.857409341135908` - 0.44145714818604453` t,
   11.00848757377149` + 0.02627731817798784` t, 16.725434723479104` -
    0.850047295730326` t, 23.0195911974063` + 0.0606538619390089` t};
```

In[ ]:=
```
pLineIntersectionMD [parl1, parl2, t, {x, y, z, w}, dTol]
```

Out[ ]=
```
{5., 11., 17., 23.}
```

Checking :

In[ ]:=
```
t0 = t /. NSolve[parl1[[1]] - 5][[1]]
s0 = t /. NSolve[parl2[[1]] - 5][[1]]
Flatten[parl1 /. {t → t0}]
Flatten[parl2 /. {t → s0}]
```

Out[ ]=
```
2.
```

Out[ ]=
```
-0.323
```

Out[ ]=
```
{5., 11., 17., 23.}
```

Out[ ]=
```
{5., 11., 17., 23.}
```

The following two lines 3-space intersect in the infinite plane in a looser tolerance

In[ ]:=
```
l26 = {1.` - 2.` t, 0.15469892331131768` + 1.9999999999246267` t, -2.3093978465294582` };
l27 = {1.` - 2.` t, -0.42264892323998693` + 1.999999999999317` t, -1.1547021535217088` };
pLineIntersectionMD [l26, l27, t, {x, y, z}, dTol]
pLineIntersectionMD [l26, l27, t, {x, y, z}, 1.*^-10]
```

Out[ ]= {}

Out[ ]= {-0.707107 , 0.707107 , 0, 0}

3. Implicitization

The method for affine linear lines may work more generally with care using the implictization method of my space curve book which converts rational parametric curves into affine curves. In low degrees we may be able to automate this part of the process, but in general we will not try to do this because special handling may be warranted. Here is an example

Let f5 be the twisted cubic curve

In[ ]:=
```
f5 = {t, t^2, t^3};
```

We know an implicitization for f5

In[ ]:=
```
F5 = {y^2 - x z, x y - z, x^2 - y}
```

Out[ ]= $\{y^2 - x z, x y - z, x^2 - y\}$

For our other curve let

In[ ]:=
```
f6 = {2.414213562373095` t, 1.2071067811865475` - 1.2071067811865475` t^2,
  0.8535533905932736` + 0.8535533905932736` t^2}
```

Out[ ]= $\{2.41421\, t, 1.20711 - 1.20711\, t^2, 0.853553 + 0.853553\, t^2\}$

To implicitize this curve we let

In[ ]:=
```
A = {{0.`, 2.414213562373095` , 0.`}, {-1.2071067811865475` , 0.`, 1.2071067811865475` },
   {0.8535533905932736` , 0.`, 0.8535533905932736` }, {0.`, 0.`, 1.`}};
A // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0. & 2.41421 & 0. \\ -1.20711 & 0. & 1.20711 \\ 0.853553 & 0. & 0.853553 \\ 0. & 0. & 1. \end{pmatrix}$$

In[ ]:=
```
F6 = FLTMD[tBasis2 , A, 2, {x2, x1}, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 5}

» Final Hilbert Function {1, 3, 5}

Out[ ]= $\{1. - 0.414214\, y - 0.585786\, z, -0.5\, x^2 - 0.5\, y^2 + 1.\, z^2\}$

So we simply solve the implicit equations

In[ ]:=
```
NSolve[Join[F5, F6]]
```
Out[ ]=
```
{}
```

We have an overdetermined system of 5 equations in 3 unknowns with one equation being numeric. Thus NSolve will not solve this seeing this as inconsistent. But we can check with my Bézout theorem, the proved part, in the space curve book to see if there should be a solution. For this it is enough to take m to be the maximum total degree of any equation or any larger number. Here we try $m = 3$. If the Sylvester matrix has maximal rank, alternatively the nullspace is {0} for a loose tolerance, then we can conclude that there are no solutions.

In[ ]:=
```
S3 = sylvesterMD[Join[F5, F6], 3, {x, y, z}];
Dimensions[S3]
```
Out[ ]=
```
{26, 20}
```

Now the numerical matrix rank is given using our procedure above

In[ ]:=
```
matrixrank[S3, dTol]
```
Out[ ]=
```
19
```

So the nullspace will have rank 1

In[ ]:=
```
ns = Flatten[nullspace[S3, dTol]];
ns / ns[[1]]
```
Out[ ]=
```
{1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.}
```

But in the rank one case we can guess from the form of fVecMD that since the first component is 1 then the point will be given by the next 3 components, thus guess

p = {1, 1, 1}.

Obviously the twisted cubic goes through this point.

In[ ]:=
```
f5 /. {t → 1}
```
Out[ ]=
```
{1, 1, 1}
```

But for f6 we need to find a parameter value. The brute force method suggests we calculate

In[ ]:=
```
t6 = t /. NSolve[f6[[1]] - 1, t][[1]]
```
Out[ ]=
```
0.414214
```

In[ ]:=
```
f6 /. {t → t6}
```
Out[ ]=
```
{1., 1., 1.}
```

so we see the curves do intersect at {1, 1, 1}.

If the matrix rank of S3 had been 20 then one might try a looser tolerance or higher m. If it had been less than 19 then one would expect to have two or more intersection points. In this case may wish to square the system and check solutions as is done with numerical system solvers, use Gauss Newton, brute force or projection, as in section 5. Mathematica users also have the option of directly using the built in function FindInstance in the non-numeric case. For the example above note

```
In[ ]:= f6s = f6 /. {t → s};
        FindInstance[f5 == f6s, {t, s}]
        FindInstance[f5 == f6s, {t, s}, 2]

Out[ ]= {{t → 1., s → 0.414214}}

Out[ ]= {}
```

So **FindInstance** agrees that this exact system has 1 but not 2 solutions.

4. Hybrid method

We may have occasion to find the intersection of two curves, one given by a parameterization and one given implicitly. As an example consider the curves above

```
In[ ]:= f5 = {t, t^2, t^3};
        f6 = {2.414213562373095` t, 1.2071067811865475` - 1.2071067811865475` t^2,
           0.8535533905932736` + 0.8535533905932736` t^2};
```

The first is just the twisted cubic given implicitly by

```
In[ ]:= F5 = twCubic
Out[ ]= {-y^2 + x z, -x^2 + y, -x y + z}
```

We proceed by evaluating the implicit curve on the parametric curve.

```
In[ ]:= f56 = Expand[F5 /. Thread[{x, y, z} → f6]]
Out[ ]= {-1.45711 + 2.06066 t + 2.91421 t^2 + 2.06066 t^3 - 1.45711 t^4,
         1.20711 - 7.03553 t^2, 0.853553 - 2.91421 t + 0.853553 t^2 + 2.91421 t^3}
```

We need to find t making f56 = 0. In an exact case NSolve might be able to do this, numerically we first find a solution to one component and apply to f56

```
In[ ]:= sol56 = NSolve[f56[[1]]]
Out[ ]= {{t → -0.707107 - 0.707107 ⅈ}, {t → -0.707107 + 0.707107 ⅈ}, {t → 0.414214}, {t → 2.41421}}

In[ ]:= f56 /. sol56
Out[ ]= {{0. - 6.66134 × 10^-16 ⅈ, 1.20711 - 7.03553 ⅈ, 4.97487 + 0.853553 ⅈ},
         {0. + 6.66134 × 10^-16 ⅈ, 1.20711 + 7.03553 ⅈ, 4.97487 - 0.853553 ⅈ},
         {1.66533 × 10^-16, -2.22045 × 10^-16, -2.22045 × 10^-16}, {-3.55271 × 10^-15, -39.799, 39.799}}
```

We see the third answer is the only reasonable one. Therefore the common point is **p** where

In[∘]:=
```
p = f6 /. sol56[[3]]
F5 /. Thread[{x, y, z} → p]
```

Out[∘]= {1., 1., 1.}

Out[∘]= $\{1.11022\times10^{-16}, -1.11022\times10^{-16}, 0.\}$

5. Projection

If we project on the first few coordinates then we are essentially doing the brute force method. But, as in my Space Curve Book we can get better results with random projections, especially to the plane where we can make sure we don’t miss any intersection points. Again here is an example.

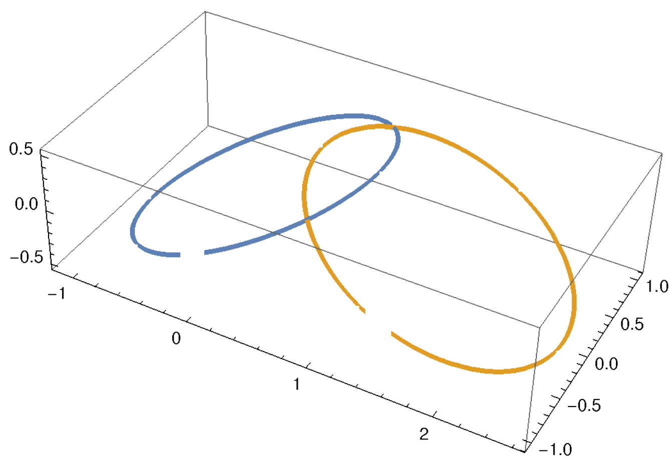
$$f7 = \left\{\frac{2\,t}{1+t^2}, \frac{1-t^2}{1+t^2}, \frac{t}{1+t^2}\right\};$$

$$f8 = \left\{\frac{1.5` + 2.`\,t + 1.5`\,t^2}{1.` + t^2}, \frac{1-t^2}{1+t^2}, -\frac{t}{1+t^2}\right\};$$

Plotting it seems like there are two real intersection points.

In[∘]:=
```
ParametricPlot3D [{f7, f8}, {t, -20, 20}, PlotRange → All]
```

Out[∘]=



To deal with all coordinates at once we project with a pseudorandom projection, in this case we will use our pseudorandom FLT **fprd3D**.

In[∘]:=
```
h7 = Simplify[fltMD[f7, fprd3D]]
h8 = Simplify[fltMD[f8, fprd3D]]
```

Out[∘]= $$\left\{\frac{0.952289 - 0.610395\,t - 0.952289\,t^2}{1. + t^2}, \frac{-0.0454808 + 0.705012\,t + 0.0454808\,t^2}{1. + t^2}\right\}$$

Out[∘]= $$\left\{\frac{0.494493 - 0.610395\,t - 1.41009\,t^2}{1. + t^2}, \frac{-0.258347 - 1.27266\,t - 0.167386\,t^2}{1. + t^2}\right\}$$

We could either implicitize both or use brute force. In either case we expect degree 2 for both so by Bézout we expect 4 complex solutions. Brute force is easier here

In[◦]:= **h8s = h8 /. {t → s}**

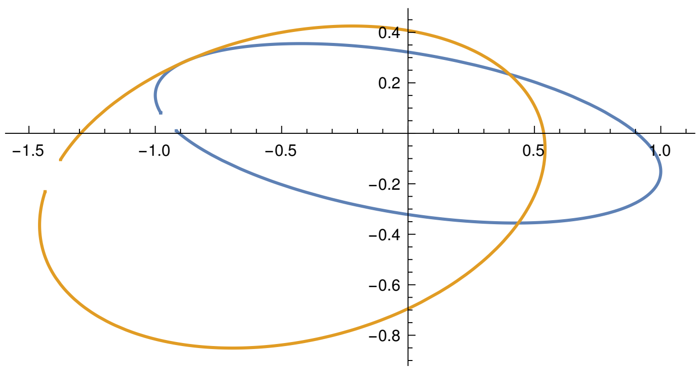
Out[◦]= $\left\{\frac{0.494493 - 0.610395\, s - 1.41009\, s^2}{1. + s^2}, \frac{-0.258347 - 1.27266\, s - 0.167386\, s^2}{1. + s^2}\right\}$

In[◦]:= **sol78 = NSolve[h7 - h8s]**

Out[◦]= {{s → -2.21525 , t → 2.21525}, {s → -2.33576 , t → 2.45276},
{s → 0.0771368 , t → -0.867668}, {s → -0.451416 , t → 0.451416}}

so we actually get 4 real solutions. As points they can be given by

In[◦]:= **ParametricPlot[{h7, h8}, {t, -20, 20}, PlotRange → All]**

Out[◦]=


In[◦]:= **Q78 = h1 /. sol78**

Out[◦]= {{-0.858778 , 0.294462}, {-0.894218 , 0.278984},
{0.436422 , -0.355397}, {0.400982 , 0.234297}}

Since they are all real we can fiber lift as in my Space Curve Book section 2.8. But we need implicitiza - tions of our original curves. Let

```
B1 = {{0, 2, 0}, {-1, 0, 1}, {0, 1, 0}, {1, 0, 1}};
B2 = {{1.5, 2, 1.5}, {-1, 0, 1}, {0, -1, 0}, {1, 0, 1}};
```

Then the implicit equations are

In[◦]:= **F7 = FLTMD[tBasis2 , B1, 2, {x2, x1}, {x, y, z}, dTol]**

» Initial Hilbert Function {1, 3, 5}

» Final Hilbert Function {1, 3, 5}

Out[◦]= $\{-0.5\, x + 1.\, z, 1. - 1.\, x^2 - 1.\, y^2\}$

In[◦]:= **F8 = FLTMD[tBasis2 , B2, 2, {x2, x1}, {x, y, z}, dTol]**

» Initial Hilbert Function {1, 3, 5}

» Final Hilbert Function {1, 3, 5}

Out[◦]= $\{1. - 0.666667\, x - 1.33333\, z, -0.2\, x^2 + 0.45\, y^2 - 0.8\, x\, z + 1.\, z^2\}$

So we can try to lift using **fFiberMD**

In[ ]:=
```
a7 = fFiberMD[F7, prd3D, Q78[[1]], {x, y, z}, 1.*^-8][[1]]
b7 = fFiberMD[F7, prd3D, Q78[[2]], {x, y, z}, 1.*^-8][[1]]
c7 = fFiberMD[F7, prd3D, Q78[[3]], {x, y, z}, 1.*^-8][[1]]
d7 = fFiberMD[F7, prd3D, Q78[[4]], {x, y, z}, 1.*^-8][[1]]
```

Out[ ]= {0.75, -0.661438, 0.375}

Out[ ]= {0.699188, -0.714938, 0.349594}

Out[ ]= {-0.99001, 0.141, -0.495005}

Out[ ]= {0.75, 0.661438, 0.375}

This gives 4 points on f7 as expected with 4 points on f8

In[ ]:=
```
a8 = fFiberMD[F8, prd3D, Q78[[1]], {x, y, z}, 1.*^-8][[1]]
b8 = fFiberMD[F8, prd3D, Q78[[2]], {x, y, z}, 1.*^-8][[1]]
c8 = fFiberMD[F8, prd3D, Q78[[3]], {x, y, z}, 1.*^-8][[1]]
d8 = fFiberMD[F8, prd3D, Q78[[4]], {x, y, z}, 1.*^-8][[1]]
```

Out[ ]= {0.75, -0.661438, 0.375}

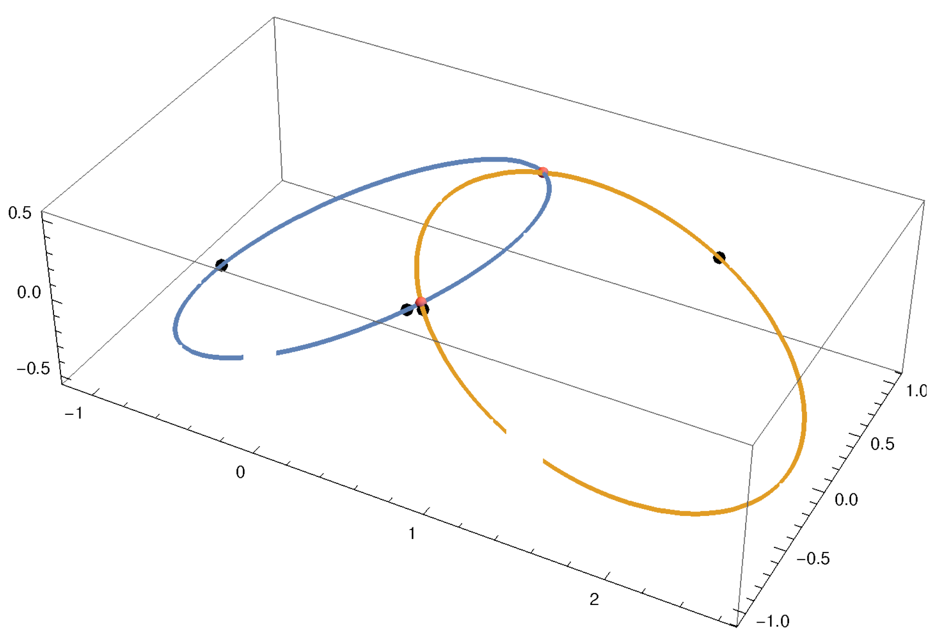
Out[ ]= {0.77638, -0.690199, 0.36181}

Out[ ]= {1.65336, 0.98817, -0.0766805}

Out[ ]= {0.75, 0.661438, 0.375}

The first and last points in both cases match but the others don' t. Thus we have 2 real intersection points and no complex points. The picture is now

In[ ]:=
```
Show[ParametricPlot3D[{f7, f8}, {t, -20, 20}, PlotRange → All],
 Graphics3D[{{Pink, Ball[a7, .03], Ball[d8, .03], Opacity[.5]},
   {PointSize[Large], Black, Point[{b7, c7, b8, c8}]}}]]
```

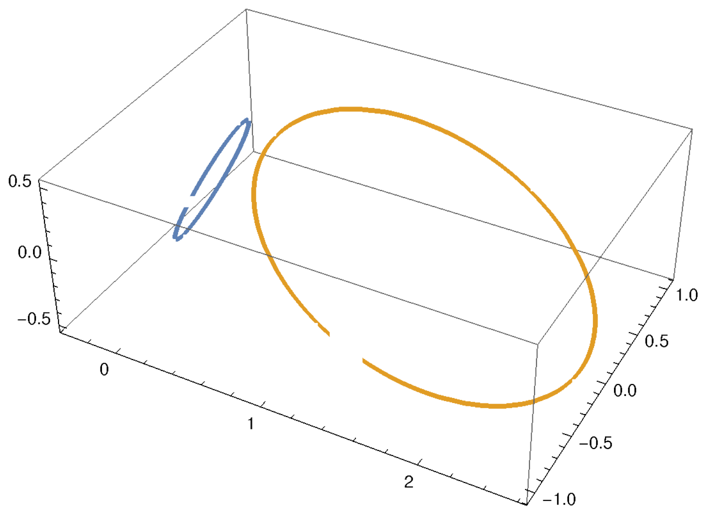
Out[ ]=


If we make a change to f7 but leave f8 alone

```
f7a = {0.5` t/(1 + t^2), 0.25` (1 - t^2)/(1 + t^2), t/(1 + t^2)};
```

```
In[ ]:= Show[ParametricPlot3D [{f7a, f8}, {t, -20, 20}, PlotRange → All],
  Graphics3D [{Red, PointSize [Large]}]]
```

Out[ ]=



The implicitizaton of **f7a** is now

```
B1a = {{0, .5, 0}, {-.25, 0, .25}, {0, 1, 0}, {1, 0, 1}};
```

```
In[ ]:= F7a = FLTMD[tBasis2, B1a, 2, {x2, x1}, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 5}

» Final Hilbert Function {1, 3, 5}

Out[ ]= $\{-2.\ x + 1.\ z,\ 1. - 16.\ x^2 - 16.\ y^2\}$

There appear to be no real intersection points. But projecting and brute force gives

```
In[ ]:= h7a = Simplify[fltMD[f7a, fprd3D]]
h8s = Simplify[fltMD[f8, fprd3D]] /. {t → s}
```

Out[ ]= $\left\{\frac{0.238072 - 0.152599\ t - 0.238072\ t^2}{1. + t^2},\ \frac{-0.0113702 + 0.917879\ t + 0.0113702\ t^2}{1. + t^2}\right\}$

Out[ ]= $\left\{\frac{0.494493 - 0.610395\ s - 1.41009\ s^2}{1. + s^2},\ \frac{-0.258347 - 1.27266\ s - 0.167386\ s^2}{1. + s^2}\right\}$

```
In[ ]:= sol78a = NSolve[h7a - h8s];
Q78a = h7a /. sol78a
```

Out[ ]= {{0.0457379, 0.398648}, {0.203249 - 0.189871 ⅈ, -0.602879 - 0.127114 ⅈ},
{0.203249 + 0.189871 ⅈ, -0.602879 + 0.127114 ⅈ}, {-0.166361, 0.424051}}

again 4 solutions, this time two are complex. Lifting the two complex solutions gives

In[ ]:=
```
fFiberMD[F7a, prd3D, Qa[[1]], {x, y, z}, 1.*^-8, complex → True]
fFiberMD[F8, prd3D, Qa[[1]], {x, y, z}, 1.*^-8, complex → True]
```

Out[ ]= {{0.220095, 0.118567, 0.44019}}

Out[ ]= {{0.523568, 0.215827, 0.488216}}

In[ ]:=
```
fFiberMD[F7a, prd3D, Qa[[2]], {x, y, z}, 1.*^-8, complex → True]
fFiberMD[F8, prd3D, Qa[[2]], {x, y, z}, 1.*^-8, complex → True]
```

Out[ ]= {{-0.325707 - 0.074777 i, 0.109047 - 0.223349 i, -0.651414 - 0.149554 i}}

Out[ ]= {{2.05068 + 0.209221 i, 0.870652 - 0.132331 i, -0.275341 - 0.10461 i}}

In[ ]:=
```
fFiberMD[F7a, prd3D, Qa[[3]], {x, y, z}, 1.*^-8, complex → True]
fFiberMD[F8, prd3D, Qa[[3]], {x, y, z}, 1.*^-8, complex → True]
```

Out[ ]= {{-0.325707 + 0.074777 i, 0.109047 + 0.223349 i, -0.651414 + 0.149554 i}}

Out[ ]= {{2.05068 - 0.209221 i, 0.870652 + 0.132331 i, -0.275341 + 0.10461 i}}

In[ ]:=
```
fFiberMD[F7a, prd3D, Qa[[4]], {x, y, z}, 1.*^-8, complex → True]
fFiberMD[F8, prd3D, Qa[[4]], {x, y, z}, 1.*^-8, complex → True]
```

Out[ ]= {{0.228481, -0.101471, 0.456963}}

Out[ ]= {{0.500104, -0.0144187, 0.499948}}

So in this space curve situation, unlike the plane, there are really no intersections, real or complex.

For infinite points as in the brute force method it is easiest to calculate the infinite points of each curve and compare. My global function **infinitePointsMD** will give real and complex infinite points for implicitly defined curves.

6. Multiplicity

Given my previous work in the area I would be remiss not to include a discussion of multiplicity of intersection. This is covered well for plane implicit curves in my Plane Curve Book where curves are always intersecting. For space curves even intersection is rare and multiple intersections are more rare. But they can happen.

The most obvious clue that there will be a multiple intersection is when tangent vectors are in the same direction. Note that for parameterized curves the tangent vector at a point is just given by differentia - tion.

In[ ]:=
```
f9 = {t, t^3, t^2};
f10 = {2 t, -2 t^3, t^3};
```

These curves intersect obviously at {0,0,0}

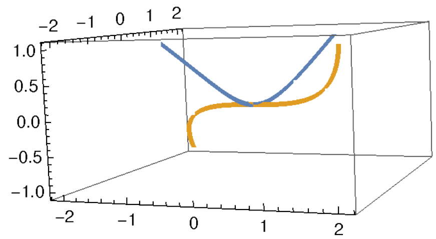
```
D[f9, t] /. {t → 0}
D[f10, t] /. {t → 0}
```

Out[ ]= {1, 0, 0}

Out[ ]= {2, 0, 0}

We do have a multiple intersection at {0,0,0}

```
ParametricPlot3D [{f9, f10}, {t, -1, 1}]
```

Out[ ]= 

If one curve is singular at the intersection point then it’s tangent vector is zero so is dependent to any vector so we will automatically have a multiple intersection at that point.

More specifically if we have several curves $\{f_1, f_2, \ldots, f_k\}$ through the same point **p0** at the same parame - ter value, say $t = 0$ can calculate their tangent vectors jointly by

D[{f₁,f₂,...,fₖ },t]/.{t -> 0}. We need the rank of this matrix to have rank k to be non-singular, anything less will be a singular point.

Consider the following

```
In[ ]:= matrixrank [D[{{t, 0}, {0, t}}, t] /. {t → 0}, dTol]
matrixrank [D[{{t, 0}, {t, t}, {0, t}}, t] /. {t → 0}, dTol]
matrixrank [D[{{t, 0, 0}, {0, t, 0}, {0, 0, t}}, t] /. {t → 0}, dTol]
```

Out[ ]= 2

Out[ ]= 2

Out[ ]= 3

To find the multiplicity we need to implicitize the curves, then we should have enough equations from the two curves to get a system with only isolated solutions so we can use our multiplicity calculation. For the present example, **f9,f10** we note that f9 is simply the twisted cubic with two coordinates flipped so by inspection the implicitatizaton is

```
In[ ]:= F9 = {x y - z^2, -x^2 + z, y - x z};
```

For f10 we use our standard procedure

```
In[ ]:= A10 = {{0, 0, 2, 0}, {-2, 0, 0, 0}, {1, 0, 0, 0}, {0, 0, 0, 1}};
F10 = FLTMD[tBasis3 , A10, 3, {x3, x2, x1}, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 6, 9}

» Final Hilbert Function {1, 3, 6, 9}

Out[ ]= $\{0.5\ y + 1.\ z,\ 0.25\ x^3 + 1.\ y\}$

In[ ]:= Note

In[ ]:= **NSolve[Join[F9, F10]]**

Out[ ]= {{x → 0., y → 0., z → 0.}, {x → 0., y → 0., z → 0.}}

so the origin is the only affine intersection point. Mathematica gives multiplicity 2 which we confirm with

In[ ]:= **multiplicityMD[Join[F9, F10], {0, 0, 0}, {x, y, z}, dTol]**

Out[ ]= 2

Here is a more complicated example with a higher multiplicity.

In[ ]:= **f11 = {t, t + t^4, t^3}**
f12 = {t, t + t^3, t^4}

Out[ ]= $\{t,\ t + t^4,\ t^3\}$

Out[ ]= $\{t,\ t + t^3,\ t^4\}$

In[ ]:= **D[f11, t] /. {t → 0}**
D[f12, t] /. {t → 0}

Out[ ]= {1, 1, 0}

Out[ ]= {1, 1, 0}

These appear to have a multiple intersection at {0, 0, 0}.

In[ ]:= **A11 = {{0, 0, 0, 1, 0}, {1, 0, 0, 1, 0}, {0, 1, 0, 0, 0}, {0, 0, 0, 0, 1}};**
F11 = FLTMD[tBasis4, A11, 4, {x4, x3, x2, x1}, {x, y, z}, dTol]

» Initial Hilbert Function {1, 4, 9, 13, 17}

» Final Hilbert Function {1, 4, 9, 13, 17}

Out[ ]= $\{-1.\ x + 1.\ y - 1.\ x\ z,\ -1.\ x^3 + 2.\ x^2\ y - 1.\ x\ y^2 + 1.\ z^3,\ 1.\ x^3 - 1.\ x^2\ y + 1.\ z^2,\ -1.\ x^3 + 1.\ z\}$

In[ ]:= **Expand[F11 /. Thread[{x, y, z} → f11]]**

Out[ ]= $\{0. - 3.33067\times10^{-16}\ t,\ 0. - 2.22045\times10^{-16}\ t^3 - 2.22045\times10^{-16}\ t^6,$
$0. - 5.55112\times10^{-16}\ t^6,\ 1.11022\times10^{-15}\ t^3\}$

In[ ]:= **A12 = {{0, 0, 0, 1, 0}, {0, 1, 0, 1, 0}, {1, 0, 0, 0, 0}, {0, 0, 0, 0, 1}};**
F12 = FLTMD[tBasis4, A12, 4, {x4, x3, x2, x1}, {x, y, z}, dTol]

» Initial Hilbert Function {1, 4, 9, 13, 17}

» Final Hilbert Function {1, 4, 9, 13, 17}

Out[ ]= $\{1.\ x^2 - 1.\ x\,y + 1.\ z,\ -1.\ x^3 + 3.\ x^2\,y - 3.\ x\,y^2 + 1.\ y^3 - 1.\ x\,z^2,\ 1.\ x^2 - 2.\ x\,y + 1.\ y^2 - 1.\ x^2\,z,\ -1.\ x - 1.\ x^3 + 1.\ y\}$

In[ ]:= **Expand[F12 /. Thread[{x, y, z} → f12]]**

Out[ ]= $\{1.11022\times10^{-15}\,t^2 + 8.88178\times10^{-16}\,t^4,\ -2.22045\times10^{-16}\,t^3 + 1.77636\times10^{-15}\,t^5 + 2.22045\times10^{-15}\,t^7 + 8.88178\times10^{-16}\,t^9,\ 7.77156\times10^{-16}\,t^2 + 8.88178\times10^{-16}\,t^4 - 2.22045\times10^{-16}\,t^6,\ -7.77156\times10^{-16}\,t + 1.22125\times10^{-15}\,t^3\}$

In[ ]:= **NSolve[Join[F11, F12]]**

Out[ ]= $\{\{x \to 1.,\ y \to 2.,\ z \to 1.\},\ \{x \to 5.93026\times10^{-15} - 8.65131\times10^{-8}\,i,\ y \to 5.93026\times10^{-15} - 8.65131\times10^{-8}\,i,\ z \to 0.\},\ \{x \to 5.93026\times10^{-15} + 8.65131\times10^{-8}\,i,\ y \to 5.93026\times10^{-15} + 8.65131\times10^{-8}\,i,\ z \to 0.\},\ \{x \to 0.,\ y \to 0.,\ z \to 0.\}\}$

In[ ]:= **multiplicityMD [Join[F11, F12], {0, 0, 0}, {x, y, z}, dTol]**

Out[ ]= 3

So the multiplicity is 3. We can see what happens after psuedorandom projection

In[ ]:= **h11 = FLTMD[F11, fprd3D , 4, {x, y, z}, {x, y}, 1.*^-9][[1]]**

» Initial Hilbert Function {1, 3, 6, 10, 14}

» Final Hilbert Function {1, 3, 6, 10, 14}

Out[ ]= $0.289591\,x + 0.272543\,x^2 - 3.67538\,x^3 + 0.000017299\,x^4 + 1.\,y + 2.24529\,x\,y + 0.25229\,x^2\,y + 0.00144884\,x^3\,y + 4.50345\,y^2 + 3.1214\,x\,y^2 + 0.0455044\,x^2\,y^2 + 6.70231\,y^3 + 0.635189\,x\,y^3 + 3.32494\,y^4$

In[ ]:= **h12 = FLTMD[F12, fprd3D , 4, {x, y, z}, {x, y}, 1.*^-9][[1]]**

» Initial Hilbert Function {1, 3, 6, 10, 14}

» Final Hilbert Function {1, 3, 6, 10, 14}

Out[ ]= $0.289591\,x - 0.482243\,x^2 + 2.98358\,x^3 - 5.22287\,x^4 + 1.\,y - 2.84001\,x\,y + 13.0495\,x^2\,y - 4.05659\,y^2 + 0.636495\,x\,y^2 + 4.44236\,y^3$

In[ ]:=
```
NSolve[{h11, h12}]
```

Out[ ]=
```
{{x → -0.275899 - 2.09403 i, y → -2.32073 - 0.603077 i},
 {x → -0.275899 + 2.09403 i, y → -2.32073 + 0.603077 i},
 {x → 1.1877 - 0.863081 i, y → -1.245 - 1.33413 i},
 {x → 1.1877 + 0.863081 i, y → -1.245 + 1.33413 i}, {x → 1.59938 , y → 0.755961},
 {x → -0.762615 + 1.67895 i, y → -0.417967 - 1.601 i},
 {x → -0.762615 - 1.67895 i, y → -0.417967 + 1.601 i},
 {x → -0.329327 + 0.487255 i, y → -0.329981 - 0.609184 i},
 {x → -0.329327 - 0.487255 i, y → -0.329981 + 0.609184 i},
 {x → 0.257676 - 0.421795 i, y → -0.126584 - 0.261041 i},
 {x → 0.257676 + 0.421795 i, y → -0.126584 + 0.261041 i},
 {x → 0.0269927 - 0.217521 i, y → -0.012448 + 0.102628 i},
 {x → 0.0269927 + 0.217521 i, y → -0.012448 - 0.102628 i},
 {x → 6.32415 × 10^-8, y → -1.83142 × 10^-8},
 {x → -6.32415 × 10^-8, y → 1.83142 × 10^-8}, {x → 0. , y → 0.}}
```

Note the 16 solutions as these cubic space curves project to 4th degree plane curves. Three of the solutions are essentially zero so the Mathematica multiplicity is 3. This agrees with our multiplicity.

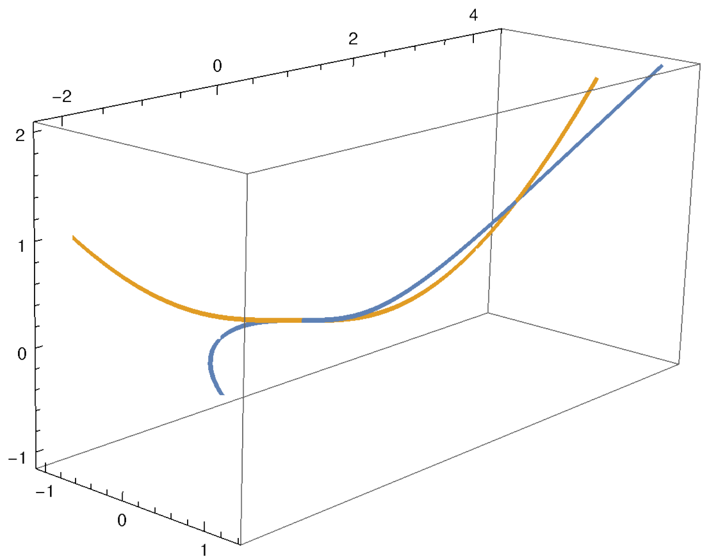
In[ ]:=
```
multiplicityMD[{h11, h12}, {0, 0}, {x, y}, dTol]
```

Out[ ]= 3

The multiplicity stays the same under projection. In general this will happen with a random or pseudo - random projection mapping an intersection point if two curves in $\mathbb{R}^n$ to the intersection in $\mathbb{R}^2$. For a non-random projection it can happen that the intersection in the plane has higher multiplicity. In particular there will generally be non-intersection points mapping to an intersection point so the multiplicity will go from 0 to a positive number.

In[ ]:= `ParametricPlot3D[{f11, f12}, {t, -1, 1.3}]`

Out[ ]=



In[ ]:= `ContourPlot[{h11 == 0, h12 == 0}, {x, -1, 2}, {y, -1, 1}]`

Out[ ]=