4/2020

In spite of their unsolvability, inconsistent equations arise in practice and must be solved. [Gilbert Strang]

Algebraic Space Curves.

barryhdayton.space

Space curves present two challenges that were not present with plane curves. First, rather than just one equation, space curves require several equations; a space curve in $\mathbb{R}^n$, $n \geq 3$, requires at least $n-1$ equations, possibly more. Unlike the equation of a plane curve which is unique up to scalar multiplication, these equations are not at all unique. Second the complement of the curve in $\mathbb{R}^n$, unlike in the plane case, is connected, possibly complicated, and of limited use in understanding the curve.

I will distinguish between two cases, first the *naive* case of curves given by 2 equations in $\mathbb{R}^3$, the case seen in multivariable calculus textbooks. We will see that some of plane curve techniques can still be used thanks to the existence of the cross product in $\mathbb{R}^3$. The general case, which consists of perhaps more than $n-1$ equations in $n \geq 3$ variables will require new techniques and, in particular, heavy use of numerical linear algebra.

It is assumed that the reader have some familiarity with my plane curve book or the Mathematica Journal article. Also, the code is in the Mathematica notebook GlobalFunctionsMD.nb available at my website listed above.

Table of Contents

1 | Naive Case: curves in $\mathbb{R}^3$

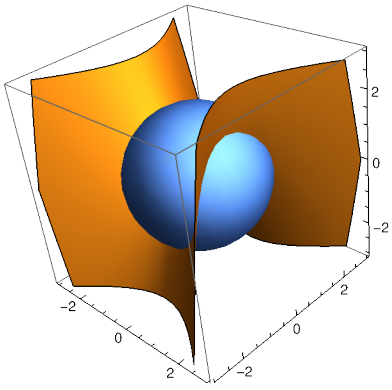
1.1 Emulating Plane Curves

As a seemingly simple example consider the curve produced by intersecting a hyper-boloid and an ellipsoid.

Example 1.

In[97]:=
```
F1 = {f11, f12} = {x^2 − y^2 − z, x^2 + y^2 + z^2 − 4};
ContourPlot3D [{f11 == 0, f12 == 0},
  {x, −3, 3}, {y, −3, 3}, {z, −3, 3}, Mesh → None ]
```

Out[97]=



The two equations $\{f_{11} = 0,\ f_{12=0}\}$ give an under determined system but Mathematica will still give a pseudo random points

In[122]:=
```
p1 = {x, y, z} /. NSolve [{f11, f12}, {x, y, z}, Reals ][[1]]
```

NSolve : Infinite solution set has dimension at least 1. Returning intersection of solutions with $-\frac{142003\ x}{115806} + \frac{40299\ y}{38602} - \frac{69046\ z}{57903} == 1.$

Out[122]= {1.15413 , 1.44616 , −0.75935 }

The first thing to notice is that at each point we have a tangent vector.

First we can find the normal vector to each of the surfaces at p_1. Recall the gradient, Grad, gives the vector {D[f,x],D[f,y],D[f,z]}.

In[123]:=
```
nv1 = Grad [f11, {x, y, z}] /. Thread [{x, y, z} → p1]
nv2 = Grad [f12, {x, y, z}] /. Thread [{x, y, z} → p1]
```

Out[123]= {2.30826 , −2.89231 , −1}

Out[124]= {2.30826 , 2.89231 , −1.5187 }

The tangent vector is simply the cross product

In[125]:=
```
tv1 = Cross[nv1, nv2]
```

Out[125]=
```
{7.28487, 1.19729, 13.3524}
```

More generally we can use the function below to get a unit tangent vector.

In[98]:=
```
tangentVector3D [{f_, g_}, p_, x_, y_, z_] := Module[{n1, n2, bi, tol},
    tol = 1.*^-8;
    n1 = Grad[f, {x, y, z}] /. Thread[{x, y, z} → p];
    n2 = Grad[g, {x, y, z}] /. Thread[{x, y, z} → p];
    bi = Cross[n1, n2];
    If[Norm[bi] < tol, Echo[p, "No tangent vector at "]; bi, Normalize[bi]]]
```

In[126]:=
```
tangentVector3D [{f11, f12}, p1, {x, y, z}]
```

Out[126]=
```
{0.477462, 0.0784727, 0.875141}
```

A point with a tangent vector is called *regular* while one without a tangent vector is called *singular*. As noticed in the plane curve book singular points may be unstable, further there are some new technical problems with this defintion that will be discussed later.

In this naive case we can also get critical points.

In[99]:=
```
criticalPoints3D [{f_, g_}, {x_, y_, z_}] :=
  Module[{J, ob}, ob = RandomReal[{.7, 1.3}, 3].{x^2, y^2, z^2};
    J = D[{f, g, ob}, {{x, y, z}}];
    {x, y, z} /. NSolve[{f, g, N[Det[J]]}, {x, y, z}, Reals]]
```

In[130]:=
```
critpts = criticalPoints3D [{f11, f12}, {x, y, z}]
```

Out[130]=
```
{{1.24962, 0., 1.56155}, {1.41921, -1.4089, 0.029167},
  {-1.41921, 1.4089, 0.029167}, {0., 1.24962, -1.56155},
  {0., -1.24962, -1.56155}, {-1.24962, 0., 1.56155},
  {1.41921, 1.4089, 0.029167}, {-1.41921, -1.4089, 0.029167}}
```

As in the plane curve case we can also find points on the curve by picking an arbitrary point and finding the point on the curve closest to it.

In[100]:=
```
closestPoint3D [{f_, g_}, p_, x_, y_, z_] := Module[{J, sol},
    J = D[{f, g, (x - p[[1]])^2 + (y - p[[2]])^2 + (z - p[[3]])^2}, {{x, y, z}}];
    sol = {x, y, z} /. NSolve[{f, g, Det[J]}, {x, y, z}, Reals];
    MinimalBy[sol, Norm[# - p] &][[1]]
  ]
```

There may be infinitely many closest points.

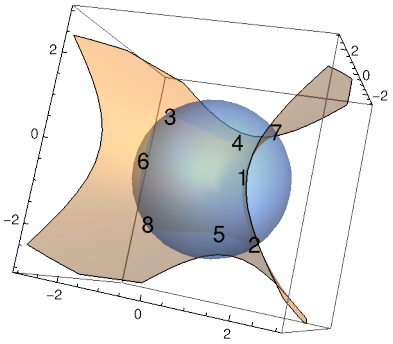
```
In[80]:= p2 = closestpoint3D [{f11, f12}, {1, 1, 1}, x, y, z]
Out[80]= {1.40516 , 0.962189 , 1.04867 }
```

One of the main things we can do in the naive case is to trace curves. Typically we first attempt a plot with critical points labeled so we can trace from one critical point to the next.

In Example 1 we plot

```
In[131]:= Show [ContourPlot3D [{f11 == 0, f12 == 0}, {x, −3, 3}, {y, −3, 3},
            {z, −3, 3}, Mesh → None , ContourStyle → Opacity [.4]],
          Graphics3D [Table [{Text [Style [i, FontSize → 14], critpts [[i]]]}, {i, 8}]]]
```

Out[131]=



Perhaps by manually rotating this plot we can decide how to travel through these points. Since the order the critical points are listed may change in different runs we leave this to the reader.

As with plane affine curves the existence of the cross product gives us a canonical direction and hence path tracing routines. We use the analog of the plane **pathFinderT** as this is the most efficient. Note that we can never trace out of a singular point, but these will attempt to trace into a singular point. Switching the order of the equations, or replacing one by its negative will change direction of the tracing.

Here $\{f, g\}$ are the equations, p the initial point, q the terminal point, and s the step size.

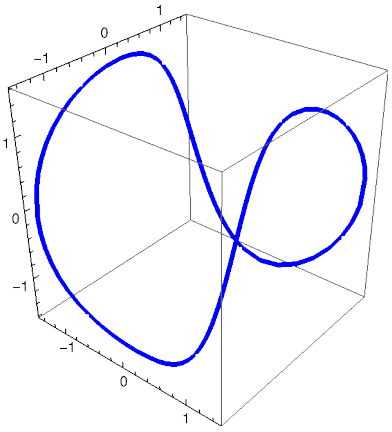
For example, tracing from critical point 6 to critical point 3 of Example 1 with a larger than normal step size gives

```
In[107]:= L1 = pathFinder3D [{f11, f12}, critpts [[6]], critpts [[3]], .3, {x, y, z}]
Out[107]= {{−1.24962 , 0., 1.56155 }, {−1.26629 , 0.286675 , 1.5213 },
           {−1.30888 , 0.55283 , 1.40755 }, {−1.36178 , 0.786664 , 1.2356 },
           {−1.41054 , 0.984504 , 1.02036 }, {−1.44484 , 1.14627 , 0.773625 },
           {−1.45773 , 1.27283 , 0.504894 }, {−1.41921 , 1.4089 , 0.029167 }}
```

We will not show the rest of the work but continuing along our path and tracing each segment, then joining them into one long list Lex1 we can plot our complete curve.

In[93]:=
```
Graphics3D [{Thick, Blue, Line[L]}, Axes → True]
```

Out[93]=



Normally infinite points will exist. We can again handle them as in the plane case by transforming each infinite point to a finite one. The FLT transformations will be discussed later in this summary so we will not pursue this here.

1.2 Projection

Later in this book a major tool will be projection. Here a projection is a linear transformation $\mathbb{R}^3 \longrightarrow \mathbb{R}^2$ expressed in matrix form with two orthogonal rows. While random or pseudo-random projections are better, discussed in the next section, for our Example 1 the simple projection by eliminating the z-coordinate will be good enough.

Projection Pxy

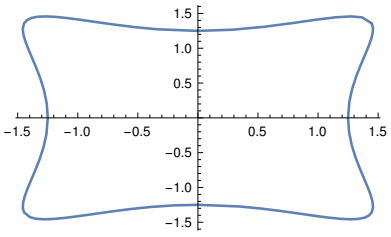
In[9]:=
```
Pxy = {{1, 0, 0}, {0, 1, 0}};
TPxy = {#[[1]], #[[2]]} &;
TPxy[{7, 9, 13}]
```

Out[11]=
```
{7, 9}
```

Projection of Example 1

In[67]:=
```
PLex1 = TPxy /@ Lex1;
ListLinePlot [PLex1 ]
```

Out[68]=



Given a naive space curve we can apply a projection to the equations by finding a sufficient number of random points and interpolating. Although degree is not as an important invariant as with plane curves in the naive case of a curve $\{f_1, f_2\}$ we can say the *degree* is $\mathrm{tdeg}[f_1] * \mathrm{tdeg}[f_2]$ for the total degree tdeg of the defining equations.

For Example 1 both equations are quadratics so the degree of the curve is 4. By interpolation we need $6 * 5 \div 2\text{-}1 = 14$ points. We might need several tries.

In[60]:=
```
SPLex1 = RandomSample [PLex1 , 14]
```

Out[60]=
```
{{−1.45706 , −1.29843 }, {1.28575 , 0.426256 },
  {1.28028 , −0.391597 }, {−1.42304 , 1.40434 }, {−1.44667 , −1.35882 },
  {−1.38353 , 0.874503 }, {0., 1.24962 }, {0.984504 , 1.41054 },
  {−0.837583 , 1.37434 }, {−1.20675 , −1.45353 }, {−1.45353 , 1.20675 },
  {−0.569215 , −1.31221 }, {1.43865 , −1.11231 }, {−1.2854 , −1.45761 }}
```

In[61]:=
```
h = ACurve [SPLex1 , x, y]
```

Out[61]=
$$-1.90801 - 2.5125 \times 10^{-12}\, x + 0.477003\, x^2 + 1.42986 \times 10^{-12}\, x^3 + 0.477003\, x^4 - 3.17788 \times 10^{-13}\, y + 4.45114 \times 10^{-13}\, x\, y + 6.22864 \times 10^{-14}\, x^2\, y - 2.61348 \times 10^{-13}\, x^3\, y + 0.477003\, y^2 - 4.87154 \times 10^{-14}\, x\, y^2 - 0.954007\, x^2\, y^2 + 5.48053 \times 10^{-13}\, y^3 + 3.61575 \times 10^{-13}\, x\, y^3 + 0.477003\, y^4$$

By symmetry we don't expect terms with odd degrees in either variable so we can chop small coefficients.

In[63]:=
```
h = Chop [h, 1.*^−11 ]
```

Out[63]=
$$-1.90801 + 0.477003\, x^2 + 0.477003\, x^4 + 0.477003\, y^2 - 0.954007\, x^2\, y^2 + 0.477003\, y^4$$

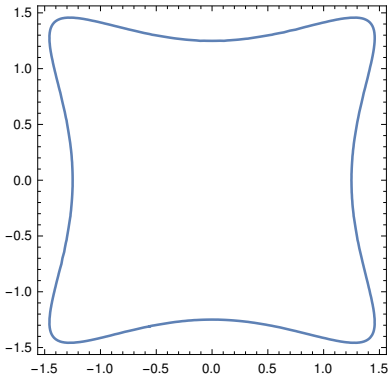
In fact this looks like an exact polynomial

In[69]:=
```
he = Expand [h / Coefficient [h, y^4]]
```

Out[69]=
$$-4. + 1.\, x^2 + 1.\, x^4 + 1.\, y^2 - 2.\, x^2\, y^2 + 1.\, y^4$$

In[70]:= ContourPlot [he == 0, {x, −1.5, 1.5}, {y, −1.5, 1.5}]

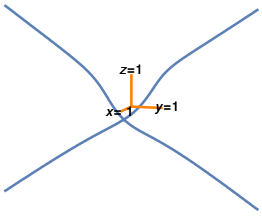
Out[70]=



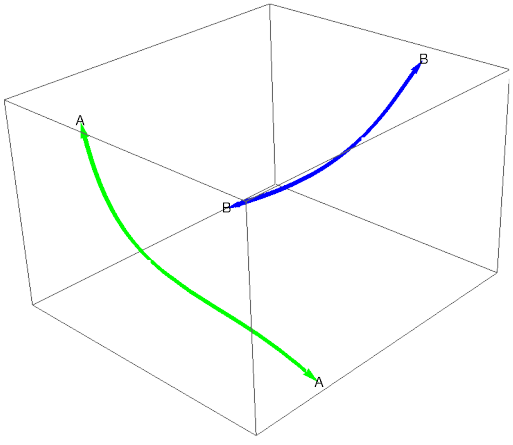
Briefly, one important final example

```
f = x y + z ;
g = −x ^ 2 + y ^ 2 − 2 z ^ 2 + z + 2;
```

Projecting via a default random projection, **PRD**, we get



The crossing is where two different components project to the same point. In $\mathbb{R}^3$ the picture is



where **A,B** are two distinct infinite points, so each component is a loop with one infinite point. We can call them, as in the plane case, *pseudo-lines.*

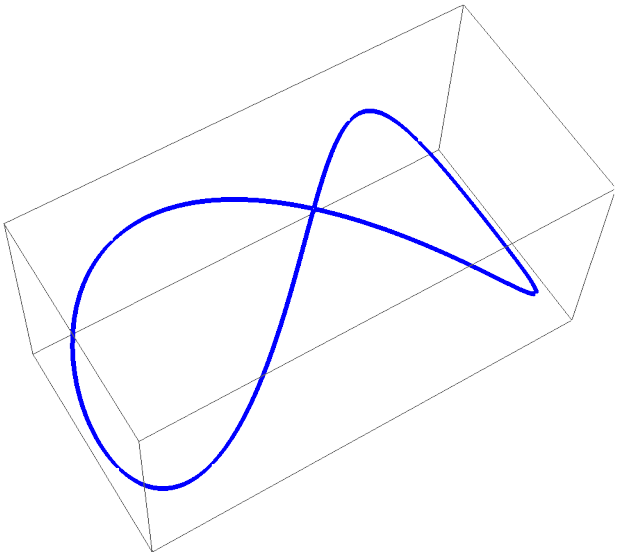
1.2.1 Nice Example: Viviani Curve

The Viviani Curve [see WolframAlpha.com] gives a nice example of a singular space curve which looks very different depending on the projection. The curve, often seen as a parametric curve, is given implicitly by

In[284]:=
```
v1 = x^2+y^2+z^2-4;
v2 = (x-1)^2+y^2-1;
V = {v1, v2}
```

Out[286]= $\{-4+x^2+y^2+z^2, -1+(-1+x)^2+y^2\}$

One can use either method of 1.1 or 1.2, or a parameterization, to draw the curve:



Note that the point where the branches seem to cross is actually the singular point {2, 0, 0} where they do cross

In[287]:=
```
tangentVector3D[V, {2, 0, 0}, {x, y, z}]
```

» No tangent vector at {2, 0, 0}

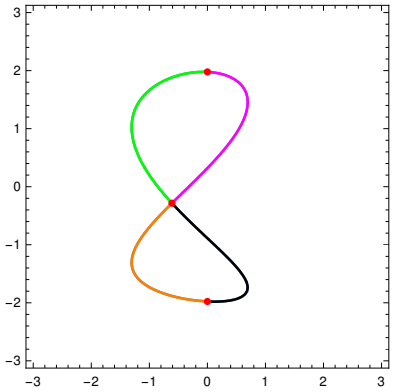
Out[287]= {0., 0., 0.}

Using projections the best is, as usual our pseudo-random prd3D or FLT version fprd3d which gives a 4th degree plane curve.

In[288]:= **vd2 = FLTMD[V, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]**

Out[288]= $1. + 4.30229\,x + 3.68817\,x^2 + 0.024428\,x^3 + 0.000444366\,x^4 - 2.00048\,y + 0.312204\,x\,y + 1.0116\,x^2\,y - 3.77986\,y^2 - 1.05115\,x\,y^2 + 0.0400199\,x^2\,y^2 + 0.511479\,y^3 + 0.901056\,y^4$

This curve can be drawn in color giving 4 segments where the center red point is the image of the singularity, the other 2 are 2D critical points.



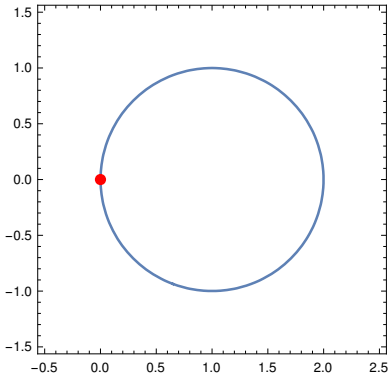
Projecting on the x,y plane using the projection `fCompProj[3,3]` gives the circle

In[303]:= **vxy = FLTMD[V, fCompProj[3, 3], 4, {x, y, z}, {x, y}, dTol][[1]]**

Out[303]= $-2.\,x + 1.\,x^2 + 1.\,y^2$

In[305]:= **ContourPlot[vxy == 0, {x, −.5, 2.5}, {y, −1.5, 1.5},**
Epilog → {Red, PointSize[Large], Point[s2d1]}]

Out[305]=



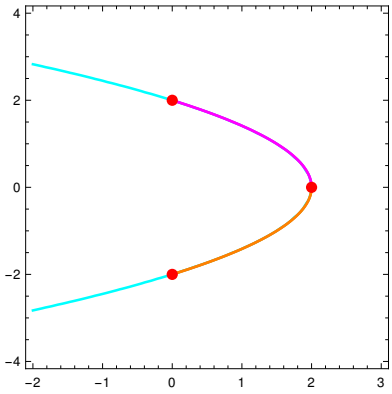
where again the red point is the image of the singular point. Each other point of the circle has a 2 point fiber.

It is weirder to project onto the x,z plane

In[304]:= **vxz = FLTMD[V, fCompProj[2, 3], 4, {x, y, z}, {x, z}, dTol][[1]]**

Out[304]= $1. - 0.5\,x - 0.25\,z^2$

Here we get a parabola. But the bounded Viviani curve can't linearly project on the unbounded parabola. In fact the image



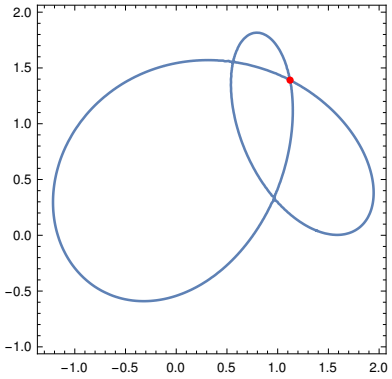
lies in the range $0 \le x \le 2$ where each point image other than the end points has a two point fiber. As one starts at the singularity of the Viviani curve and goes around a loop the projection starts at {2, 0} goes out one colored branch of the parabola and back on the same branch to {2, 0}. This is a good example of where a space curve may not map onto the FLT projection curve, particularly in the case of a non-random projection.

The non-random projection on the y-z plane does act somewhat like the random prd3d projection giving a 4th degree curve.

Another random projection with FLT matrix

Out[326]=
```
RA = {{0.5611043190123369`, 0.6690386434437178`, −0.4873902304772753`, 0},
      {0.6953402146462944`, −0.7004233312648177`,
        −0.1609631725443459`, 0}, {0, 0, 0, 1}};
```

gives the following degree 4 projection



The red point is the image of the singular point of the Viviani curve and the other 2 singularities are artifactual singularities from the projection. This is expected since the Viviani curve having a rational parameterization means it

has genus 0 so we expect, generically 3 singular points in the projection. Actually the `fprd3D[2,3]` and non-random projection on the y-z plane have isolated singularities, the former a double singularity at the infinite point {1,0,0} and the later two real plane isolated singularities. So all the projec - tions remain rational curves.

1.3 Ovals and Pseudo Lines

In $\mathbb{R}^n$, $n \geq 3$, we can still distinguish between ovals and pseudo-lines by counting, according to multiplicity, infinite points, but things work differently than in the plane case. Because higher dimensional projective spaces allow skew lines, pseudo-lines may not intersect, thus non-singular space curves, even in even degree, can have multiple pseudo-lines. Ovals no longer separate projective space into two compo - nents and do not have well-defined interiors. A curve can intersect an oval in an odd number of points. The basic difference between an oval and pseudo-line is that an oval can be deformed continuously in projective space to a point, whereas a pseudo-line cannot. For this reason some authors call an oval a *null-homotopic component* and a pseudo-line a *non-null-homotopic component.*

1.4 Fractional Linear Transformations on 3-Space.

As in the plane we will consider only invertible FLT here. Non-invertible FLT will be covered in Chapter 2.

So let A be an invertible 4×4 matrix. Then the transformation $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$ is given on the point level by Wolfram's **TransformationFunction.** More generally

```
fltMD[p, A] = TransformationFunction[A][p]
```

for any transformation matrix. In addition these transformation functions apply to the system of equations level. In 3D this function will send a naive 3D curve system to a naive 3D curve on an equation to equation basis. However in special cases we may wish to also send a larger system on equa - tion to equation basis in 3 variables so the input to this function will consist of a system F consisting of one or more equations, a 4×4 invertible matrix A and the variables $\{x, y, z\}$.

```
FLT3D[F_, A_, {x_, y_, z_}] := Module[{B, d, g, h, t},
    If[Dimensions[A] ≠ {4, 4}, Echo["need A to be 4×4"]; Abort[]];
    If[MatrixRank[A] ≠ 4, Echo["A must be invertible"]; Abort[]];
    B = Inverse[A].{x, y, z, t};
    Reap[Do[
        d = tDegMD[f, {x, y, z}];
        g = Expand[t^d (f /. {x → x/t, y → y/t, z → z/t})];
        h = Expand[g /. Thread[{x, y, z, t} → B]];
        Sow[Chop[h /. {t → 1}, dTol]], {f, F}]][[2, 1]]]
```

The Wolfram Language has many transformation matrices, see, for example the examples under Transformation Matrices in *n*D in the help page **Geometric Transforms** . In addition see the symbolic transformation functions, example

In[109]:=
```
TranslationTransform[{3, -3, 2}]
```

Out[109]=
$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\right]$$

so to translate the curve given by

In[111]:=
```
F = {z - x^2 - y^2, x + y + z};
```

use

In[114]:=
$$\text{FLT3D}\left[F, \left(\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \{x, y, z\}\right]$$

Out[114]=
$$\{-20 + 6x - x^2 - 6y - y^2 + z, -2 + x + y + z\}$$

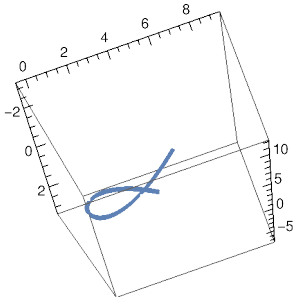
2 | General Case

We now treat the general case of a curve in $\mathbb{R}^n$, $n \geq 3$ with $k \geq n-1$ polynomial equations $F = \{f_1, f_2, \ldots, f_k\}$ in the n variables. But first, some more numerical linear algebra.

2.0 The Twisted Cubic

The standard example of a curve requiring more than $n-1$ equations is the *twisted cubic*.

In[116]:=
```
twCubic = {x z - y ^ 2, y - x ^ 2, z - x y};
```



The claim is that no two of these equations describe this curve, all 3 are needed. In fact the naive curve defined by any two contains a line in addition to the curve. Later, in section 3.2 we will learn how to analyze these curves defined by two quadratics known as QSIC. For now we use a trick. In $\mathbb{R}^3$ given a line and a plane they need not intersect but usually will, the exception is when the line is parallel to the plane. But if the line is given then a random choice of plane will intersect that line with probability very near 1. Now if we intersect the twisted cubic defined by all three equations with a random plane we get 3 points, possibly 2 are complex.

In[127]:=
```
l = RandomReal[{-1, 1}, 4].{x, y, z, 1}
```

Out[127]=
```
0.539366 - 0.665691 x - 0.249707 y - 0.449716 z
```

In[161]:=
```
sol = {x, y, z} /. NSolve[Append[twCubic, l]]
```

Out[161]=
```
{{-0.561007 + 1.34217 i, -1.4867 - 1.50594 i, 2.85528 - 1.15057 i},
 {-0.561007 - 1.34217 i, -1.4867 + 1.50594 i, 2.85528 + 1.15057 i},
 {0.566758, 0.321214, 0.182051}}
```

Thus it is enough to show that intersecting the QSIC defined by two of the three equations actually gives 4 intersection points, meaning the QSIC must have an extra component. This works fine in the first two cases

In[129]:= **NSolve[{x z – y ^ 2, y – x ^ 2, l}]**

Out[129]= {{x → –0.561007 + 1.34217 *i*, y → –1.4867 – 1.50594 *i*, z → 2.85528 – 1.15057 *i*},
{x → –0.561007 – 1.34217 *i*, y → –1.4867 + 1.50594 *i*, z → 2.85528 + 1.15057 *i*},
{x → 0., y → 0., z → 1.19935}, {x → 0.566758, y → 0.321214, z → 0.182051}}

In[131]:= **NSolve[{x z – y ^ 2, z – x y, l}]**

Out[131]= {{x → –0.561007 – 1.34217 *i*, y → –1.4867 + 1.50594 *i*, z → 2.85528 + 1.15057 *i*},
{x → –0.561007 + 1.34217 *i*, y → –1.4867 – 1.50594 *i*, z → 2.85528 – 1.15057 *i*},
{x → 0.810235, y → 0., z → 0.}, {x → 0.566758, y → 0.321214, z → 0.182051}}

But this fails for the last two equations!

In[158]:= **NSolve[{y – x ^ 2, z – x y, l}]**

Out[158]= {{x → –0.561007 + 1.34217 *i*, y → –1.4867 – 1.50594 *i*, z → 2.85528 – 1.15057 *i*},
{x → –0.561007 – 1.34217 *i*, y → –1.4867 + 1.50594 *i*, z → 2.85528 + 1.15057 *i*},
{x → 0.566758, y → 0.321214, z → 0.182051}}

The reason is that the extra line is contained in the infinite plane! So we use the trick from Chapter 6 of my plane curve book, we bring most of the line back into the affine plane by a random orthogonal FLT.

In[147]:= **A = Orthogonalize[RandomReal[{–1, 1}, {4, 4}]];**

In[151]:= **eq = FLT3D[{y – x ^ 2, z – x y, l}, A, {x, y, z}]**

Out[151]= {–0.0952451 + 0.54168 x – 0.062302 x^2 – 1.20908 y – 0.238854 x y –
0.589156 y^2 – 0.428905 z + 0.313495 x z + 0.238714 y z – 0.253297 z^2,
0.29157 – 0.202188 x + 0.0170885 x^2 + 0.0129579 y + 0.968291 x y – 0.0334473 y^2 – 0.786702 z –
0.142706 x z – 0.240355 y z – 0.275211 z^2, 0.636281 + 0.0746237 x + 0.33145 y + 0.691649 z}

In[159]:= **{x, y, z} /. NSolve[eq]**

Out[159]= {{4.45029, –0.237877, –1.28611},
{–1.0441 – 1.53794 *i*, 0.264329 – 0.376081 *i*, –0.933968 + 0.346156 *i*},
{–1.0441 + 1.53794 *i*, 0.264329 + 0.376081 *i*, –0.933968 – 0.346156 *i*},
{0.0572733, –1.99714, 0.0309371}}

There are our 4 solutions. Note these are consistent with our original 3 solutions of twCubic and *l*.

In[162]:= **fltMD[#, A] &/@ sol**

Out[162]= {{–1.0441 – 1.53794 *i*, 0.264329 – 0.376081 *i*, –0.933968 + 0.346156 *i*},
{–1.0441 + 1.53794 *i*, 0.264329 + 0.376081 *i*, –0.933968 – 0.346156 *i*},
{0.0572733, –1.99714, 0.0309371}}

We conclude it takes all 3 equations to define the twisted cubic! We will see

this curve several more times.

2.1 Macaulay and Sylvester Duality

We first generalize the Macaulay and Sylvester matrices of my Plane Curve book to an arbitrary number of variables. A problem often mentioned to me is that these matrices can get quite large, for example even in only 4 variables a Sylvester matrix of order 10 of a system of 4 degree 5 polynomials has 505K entries and takes 13.5 seconds to generate (64 bit,12 core 3.4GHZ Linux) while the Macaulay matrix of the same order and degree has as many as 2860K entries and can take 76 seconds to generate. Analyzing these matrices using singular value decompositions will take much longer. Fortunately there are enough interesting examples already in 3-space that we will only occasionally venture into higher dimensions.

The difference between the Macaulay and Sylvester matrices is that Macaulay matrices are defined at a point and measure local properties. Essentially the rows are *germs* of functions and can be truncated so monomial multiples of the defining polynomials will appear even if the resulting degree is larger than the order. The Sylvester matrix is independent of point and measures global properties. So if a monomial multiple of a defining polynomial has degree greater than the order this row is left out. This is why there are many more rows in the Macaulay matrix. For either the number of rows is dependent on the defining polynomials so there is no general count. The number of columns in both cases is always **`Length[expsMD[n,d]]`** where n is the number of variables and d is the order, that is **`Binomial[n+d,d]`**.

Already in 1916 Macaulay defined the *dual vectors* to his arrays. I implement these by the (right) null space of the Macaulay matrix as a column matrix, see for example section 2.2 of our paper [DLZ: Dayton, Li, Zeng, Math Comp 80 (276), free from ams.org/mcom]. Likewise we can also define the dual of a Sylvester matrix. Note that dual vectors of a Macaulay matrix should not be truncated but the dual vectors of a Sylvester matrix can be. So in a sense, the dual of a Macaulay matrix is a Sylvester matrix and conversely.

One can, essentially, recover the Macaulay and Sylvester matrix from their duals by taking the left nullspace. In a few cases later on constructions such as the important transformation FLTMD or taking unions of curves require working with duals and then taking the dual of the dual. Unfortunately the result can often be a system of more equations than necessary and possibly higher degrees than necessary.

Again, in 1916, Macaulay came up with the idea of a *H-basis* for a curve (or higher dimensional algebraic set) as an efficient set of equations to define an algebraic set. We have a function **`hBasisMD`** to produce such a system from an arbitrary system of equations defining a curve.

Consider the naive curve defined by equations

In[118]:=
```
f = x - z + x z + y z;
g = 2 x + y - z + x z + y z;
```

But note that since these equations vanish, by definition, on the curve then so also do

In[126]:=
```
f1 = g - f
g1 = Expand[f - f1 * z]
```

Out[126]= $x + y$

Out[127]= $x - z$

In fact the system defined by $\{f_1, g_1\}$ is equivalent to the system $\{f, g\}$ and much nicer, for example we immediately see this curve is a line. This latter system is a H-basis for this curve.

See **GlobalFunctionsMD.nb** for an implementation of these ideas.

2.2 Tangent Vectors and Definition of *curve.*

So suppose we have a system of $k \geq n-1$ polynomial equations in n unknowns. Our first task is to say what we mean by a *curve.* For example, if $k = n$ the typical situation is that the solution set is a set of isolated points. The key feature of curves, rather than other point sets is that there are *infinitely many solutions with tangent vectors and at most finitely many points without.* Here is a simple function using the Jacobian of the system, **D[F,{X}]**, to find the tangent vector at a point or to indicate that one does not exist:

In[101]:=
```
tangentVectorJMD[F_, p_, X_] := Module[{J, ns},
  If[Norm[F /. Thread[X → p]] > 1.*^-7,
   Echo["Large residue, use tangentVectorMD "];
   Return[Fail]];
  J = D[F, {X}] /. Thread[X → p];
  ns = NullSpace[J];
  If[Length[ns] == 1, Return[ns[[1]]], Echo["No unique tangent vector"]]]
```

Example 2 is the following simple space curve in $\mathbb{R}^3$ with 3 equations.

In[102]:=
```
F2 = {f21, f22, f23} = {x z - y, y z - x^2 - x, z^2 - x - 1}
```

Out[102]= $\left\{-y + x\,z,\ -x - x^2 + y\,z,\ -1 - x + z^2\right\}$

The point {0, 0, 1} is clearly on this curve, we check for a tangent vector

In[156]:=
```
tangentVectorJMD [F2, {0, 0, 1}, {x, y, z}]
```

Out[156]= {2, 2, 1}

If we try a different point on the z-axis we are told this may not be a solution.

In[157]:= tangentVectorJMD [F2, {0, 0, RandomReal [{−4, 4}]}, {x, y, z}]

» Large residue , use tangentVectorMD

Out[157]= Fail

Suppose we look at the naive curve obtained by deleting the last equation, this curve contains the entire z-axis.

In[158]:= tangentVectorJMD [{f21, f22}, {0, 0, 1}, {x, y, z}]

» No unique tangent vector

Out[158]= No unique tangent vector

In[159]:= tangentVectorJMD [{f21, f22}, {0, 0, RandomReal [{−4, 4}]}, {x, y, z}]

Out[159]= {0., 0., 1.}

So every point on the z-axis has tangent vector $\{0, 0, 1\}$ except for the point $\{0, 0, 1\}$.

The next example contains the entire x-y plane so there are many tangent vectors at a point of this plane.

In[160]:= tangentVectorJMD [{z (x − y), z (x + y)}, {2, 3, 0}, {x, y, z}]

» No unique tangent vector

Out[160]= No unique tangent vector

One important thing to note is that, unlike in the plane or naive case, the specific direction is not unique. That is, the solutions may not be canonical. This can make path tracing difficult. Also if there are more than n equations in the n unknowns then we have to solve overdetermined systems which may not work well.

Another problem with this function is given by the following example due to [L.Shen, C.Yaun, Implicitization using Univariate Resultants, J Sys Sci Complex (2010) 23, pp.804 - 814.]

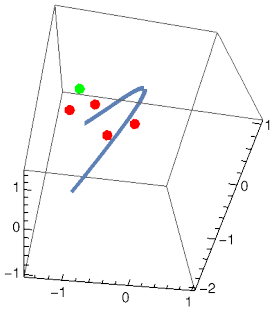
In[103]:=
$$\begin{aligned} SY = \{&-3 - 7x - 5x^2 - x^3 + 7y + 9xy + 3x^2y - 5y^2 - 3xy^2 + y^3,\\ &-x^2 - x^3 + 2xz + 3x^2z - 3xz^2 + z^3,\\ &-3y + 4y^2 - y^3 - 2yz + 3y^2z + 6z^2 - 3yz^2 + z^3\} \end{aligned}$$

Out[103]=
$$\begin{aligned} \{&-3 - 7x - 5x^2 - x^3 + 7y + 9xy + 3x^2y - 5y^2 - 3xy^2 + y^3,\\ &-x^2 - x^3 + 2xz + 3x^2z - 3xz^2 + z^3,\\ &-3y + 4y^2 - y^3 - 2yz + 3y^2z + 6z^2 - 3yz^2 + z^3\} \end{aligned}$$

In[188]:= tangentVectorJMD [SY, {−1, 1, 0}, {x, y, z}]

Out[188]= {2, −1, 2}

This is the green point in the following plot (see Section 3.1 below).



This isolated point has a tangent vector! The explanation is given in our paper [Dayton Li Zeng, Math. Comp. 80 (2011), 2143-2168] where such isolated points are called *breadth one*. But we don't want only points like this, we want non-isolated points on our curve!

We come to our key definition

Let $F = \{f_1, f_2, \ldots, f_k\}$ *be a system of* k *real polynomial equations in* n *variables.* F *defines a* ***curve*** *if there exists a non-isolated point in the solution set of* $F = 0$ *which has a unique tangent vector and all but finitely many points in this solution set are non-isolated and have unique tangent vectors.*

Note that, unlike the plane curve case, we are ruling out the case of a finite set of points. This is because when $k = n$ this is the default case and we don't want to think of a standard square non-linear system of equations as defining a curve.

The function **tangentVectorMD** in Global Functions is a more robust version of tangentVectorJMD. It does allow setting a tolerance and also gives the Hilbert function to order 4 by default and allows a larger or smaller, at least 2, by option. See the many examples in Global Functions paragraph 32.

For the example F2 above

```
In[199]:= tangentVectorMD [F2, {0, 0, 1}, {x, y, z}]
» Hilbert Function {1, 1, 1, 1, 1}
Out[199]= {−0.666667 , −0.666667 , −0.333333 }
```

For a non-isolated curve point we expect the Hilbert function to have all ones. Thus this will never *prove* that a point is non-isolated, but will strongly suggest it. In the example above it is true.

Another example is the following from my paper [*Numerical Local Rings and Local Solution of Nonlinear Systems,* SNC'07, ACM, 2007]

In[104]:=
```
nlr1 = 2 x^2 - x - x^3 + z^3;
nlr2 = x - y - x^2 + x y + z^2;
nlr3 = x y^2 z - x^2 z - y^2 z + x^3 z;
NLR = {nlr1, nlr2, nlr3}
```

Out[107]= $\left\{-x + 2x^2 - x^3 + z^3,\ x - x^2 - y + x y + z^2,\ -x^2 z + x^3 z - y^2 z + x y^2 z\right\}$

In[196]:=
```
tangentVectorMD [NLR, {0, 0, 0}, {x, y, z}]
```

» Hilbert Function {1, 1, 1, 1, 1}

Out[196]= {0., 0., −1.}

But

In[198]:=
```
tangentVectorMD [NLR, {0, 0, 0}, {x, y, z}, ord → 5]
```

» Hilbert Function {1, 1, 1, 1, 1, 0}

» Warning point may be isolated

Out[198]= {0., 0., 1.}

So even with a tangent vector there is no curve through the origin. The problem is this is what we call a *breath 1* isolated point which mimics a curve for a while in the Hilbert Function. Also note that the NLR system does contain a naive line $\{x = 1,\ z = 0\}$ but

In[200]:=
```
r = RandomReal [{0, 4}];
tangentVectorMD [NLR, {1, r, 0}, {x, y, z}]
```

» Hilbert Function {1, 2, 3, 3, 3}

» No unique tangent vector at {1, 2.61265, 0}

this line has infinitely many points without tangent vectors so does not qualify as general curve. The problem is that NLR is *non-reduced,* the multivariable generalization of non-square free. This may be discussed in a later chapter of this book.

Our definition of general curve implies the curve is reduced.

A non-isolated point on a curve with a tangent vector is called ***regular****, while a point on a curve with no tangent vector is called* ***singular****.*

Finally we mention how to get a *degree* for a general curve. This uses information from the Sylvester matrix. This may not actually be an invariant of a space curve and the software may be quirky.

For the degree use DegreeMD[F, m, X], where F is the curve, m is the order of Sylvester matrix used, and X is the list of variables. m should be about 2 more than the expected degree, if the degree is not 2 less than m, you may with larger m. For more than 3 variables this may take a while to run. The output is in text format, this function is not intended as a subroutine for other functions.

In[82]:= DegreeMD [F1, 6, {x, y, z}]

Degree is 4

In[83]:= DegreeMD [F2, 5, {x, y, z}]

Degree is 3

In[84]:= DegreeMD [C4, 4, {w, x, y, z}]

Degree is 6

In[86]:= DegreeMD [C4, 6, {w, x, y, z}]

Degree is ambiguous , try again with higher m

In[85]:= DegreeMD [C4, 8, {w, x, y, z}]

Degree is 4

2.3 Fractional Linear Transformations

The transformations we wish to use on our space curves are again *Fractional Linear Transformations, FLT*. The reader may wish to first review Chapter 6 in the Plane Curves Book. In general they are transformations $\mathbb{R}^n \longrightarrow \mathbb{R}^s$ of the form

$$T[\{x_1, x_2, \ldots, x_n\}] = \left\{ \frac{a_{1,1}\, x_1 + \cdots + a_{1,n}\, x_n + a_{1,n+1}}{d_1\, x_1 + \cdots + d_n\, x_n + d_{n+1}}, \ \ldots, \ \frac{a_{s,1}\, x_1 + \cdots + a_{s,n}\, x_n + a_{s,n+1}}{d_1\, x_1 + \cdots + d_n\, x_n + d_{n+1}} \right\}$$

That is, each component is a rational function with affine numerator and denomina - tor with all denominators the same.

As in the plane curve case it is more convenient to describe these in matrix format. So we let A be a $(s+1)\times(n+1)$ matrix where the first s rows consist of the $a_{s,j}$ and the last row consists of the d_j. Then T takes

$$\{x_1, \ldots, x_n\} \ \mapsto \ A \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix} = \{u_1, \cdots, u_s, d\} \ \mapsto \ \left\{ \frac{u_1}{d}, \cdots, \ \frac{u_n}{d} \right\}$$

On the point level we have the function fltgd[] where the gd refers to “general dimen - sions” to distinguish from the 2 variable flt which we may still wish to use in later chapters of this book.

In[108]:=
```
fltgd[p_, A_] := Module[{U, d},
    U = A.Append[p, 1];
    d = U[[-1]];
    Table[U[[i]] / d, {i, Dimensions[A][[1]] - 1}]]
```

So if

```
A = {{1, 1, 1}, {2, 2, 2}, {3, 3, 3}, {4, 4, 4}};
A // MatrixForm
p = {2, 3};
```

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{pmatrix}$$

```
fltgd[p, A]
```

$$\left\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}$$

The big change between these FLTs and the plane FLTs is that they need not be 1-1 or onto and hence may not have inverses. So the action on curves is more complicated. But note that the middle step is just matrix multiplication, that is, a linear transformation which is a simple case of an algebraic transformation. But this is not transforming the original affine curve F, rather the associated homogeneous projective curve. Thus our process is to first homogenize the curve and transformation. Then we take its Sylvester matrix *S*, find the dual DS, then multiply by the transformation matrix from GMap to get a transformed dual TDS. Taking the left dual, ST, of that we get the transformed Sylvester matrix which we can multiply by the new variables to get get a basis of multivariate polynomials B to which we can apply HBasis to get a nice basis of a projective curve. We finally specialize to get our desired basis of the transformed affine curve. The software is on the accompanying software notebook

The format is FLTgd[F, A, m, X, Y,tol] where *F* is the curve, *A* is the FLT in matrix form, m is the order of Sylvester matrices used, X are the variables in the domain, Y are variables in range and tol is the tolerance, with the global variable dtol giving the default. The output is a set of equations for the range. Also displayed will be a projective Hilbert function which is used by the software to decide if *m* is large enough. More information will be forthcoming in the actual book.

Let

In[52]:=
```
A = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}};
A // MatrixForm
```

Out[53]//MatrixForm=
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

which is the projection used in Section 1.

```
In[102]:= g = FLTgd[F1, A, 4, {x, y, z}, {x, y}, dtol]
```

» Hilbert Function {1, 3, 6, 10, 14}

```
Out[102]= {1. − 0.25 x^2 − 0.25 x^4 − 0.25 y^2 + 0.5 x^2 y^2 − 0.25 y^4}
```

```
In[104]:= Expand[−4 g[[1]]]
```

```
Out[104]= −4. + 1. x^2 + 1. x^4 + 1. y^2 − 2. x^2 y^2 + 1. y^4
```

This is the projection function he we obtained with much more difficulty in Section 1!

2.4 Projections

In general a projection will be a linear transformation from $\mathbb{R}^n \longrightarrow \mathbb{R}^k$, $k < m$, given by a $k \times n$ matrix P with orthogonal rows. Such a matrix can be embedded into a $(k+1) \times (n+1)$matrix A by filling the last row and column with zeros except for the entry in the lower right which will be a 1, just as in the last example above. This is so we can treat the projection, as above, as an FLT and have it transform curves as well as points.

We will distinguish ordinary projections like the one in the last example from *generic* projections. These are random or pseudo-random projections.

A random projection is one made with random numbers such as

```
In[54]:= P = Orthogonalize[RandomReal[{−1, 1}, {2, 3}]]
```

```
Out[54]= {{0.395675, −0.671311, −0.626724}, {−0.415005, 0.478072, −0.774092}}
```

Generally different random projections will be defined as above for each application. However we could also define a random projections, with some constraints on the random numbers used and use this projection many times. Such a projection is called pseudo-random. An example is our *default pseudorandom projection*

```
In[109]:= PRD = {{−0.30519764945947847`, 0.9522890290055899`, 0.`},
               {−0.14191095867181538`,
                −0.045480825358668514`, 0.9888340479238873`}};
```

The associated fractional linear transformation is

```
In[110]:= FPRD = {{−0.30519764945947847`, 0.9522890290055899`, 0.`, 0.`},
                 {−0.14191095867181538`, −0.045480825358668514`,
                  0.9888340479238873`, 0.`}, {0.`, 0.`, 0.`, 1.`}};
```

Both of these are assigned global variables.

I like this particular projection because the axes come out like the old fashioned 3-space axes for pictures we drew on the blackboard in Calculus III.

In our notebook of global functions we have a black-box function which plots, using FLTgd, a curve in $\mathbb{R}^3$ using a given projection. In showProjection[F, A , m , X , Y, rng},dtol] *F* is the list of equations of the curve, *A* is the matrix of the projection in FLT form, *m* is the order of the Sylvester matrices used, X is the list of variables in $\mathbb{R}^3$ and Y is the list of variable in $\mathbb{R}^2$ and rng is a positive number, the curve will be plotted in the square $\{x, -\text{rng}, \text{rng}\}, \{y, -\text{rng}, \text{rng}\}$.

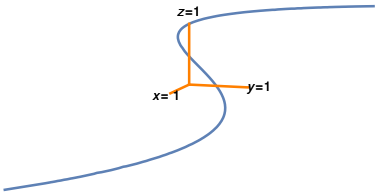
In[60]:=
```
showProjection [F2, FPRD , 4, {x, y, z}, {x, y}, 3]
```

» Hilbert Function {1, 3, 6, 9, 12}

» projection Function

$$\{1. - 2.37355\ x + 0.0574214\ x^2 + 0.000243617\ x^3 - 2.18663\ y + 0.955595\ x\,y + 0.0153028\ x^2\,y - 1.02271\ y^2 + 0.320413\ x\,y^2 + 2.2363\ y^3\}$$

Out[60]=

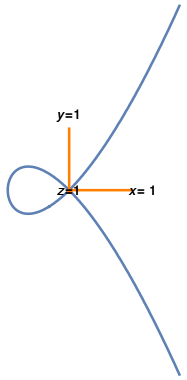


In[61]:=
```
A = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}};
showProjection [F2, A, 4, {x, y, z}, {x, y}, 3]
```

» Hilbert Function {1, 3, 6, 9, 12}

» projection Function $\{-1.\ x^2 - 1.\ x^3 + 1.\ y^2\}$

Out[62]=



A problem with ordinary projections is that the projection may interfere with the curve to get unwanted features.

For example, if we take a curve such as $\{x^2 + z^2 - 1, y\}$ under the projection A above we get the curve projection as a line $y = 0$.

In[64]:= FLTgd[{x^2 + z^2 - 1, y}, A, 3, {x, y, z}, {x, y}, dtol]

» Hilbert Function {1, 2, 3, 4}

Out[64]= {1. y}

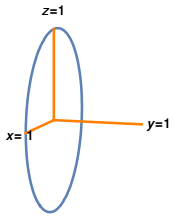
But the point projection is just the interval $-1 \le x \le 1$ of that line. Using our default pseudo-random projection the result

In[66]:= showProjection[{x^2 + z^2 - 1, y}, FPRD, 3, {x, y, z}, {x, y}, 2]

» Hilbert Function {1, 3, 5, 7}

» projection Function $\{1. - 10.957\ x^2 + 0.951082\ x\,y - 1.02271\ y^2\}$

Out[66]=



is correctly given as a circle.

This example gives one reason why generic projections are preferred over ordinary projections, the probability that the point projection of a curve is not the curve projection is much less with pseudo-random projections and even smaller with random projections. In classical algebraic geometry this fact is often known as *Noether's Normalization Theorem",* one of the rare algebraic geometry theorems attached to the name *Noether* due to the daughter Emmy, rather than father Max, of this famous mathematical family. Emmy Noether was known for her algebra while her father for geometry and, in fact, this theorem was originally stated as a theorem in algebra.

The previous example F2 showed the generic projection of the non-singular curve as still non-singular while the ordinary projection showed the projection to be a node, that is, singular. In fact this is unavoidable for projections to the plane even for generic projections. Both kinds of projections can also take a non-singular point to a cusp eg. $y^2 = x^3$. Further, when working with real curves, a projection can take a complex point to an isolated real point.

For A as above

In[71]:=
```
fltgd[{0, 1, i}, A]
```

Out[71]=
```
{0, 1}
```

We call points like this in the point projection *artifacts* or *artifactual singularities* to distinguish from singularities of the plane projection coming from singularities of the space curve. The curve projection may also contain additional components that are not part of the point projection, I call these *ghost* components. The important result is

Under any projection of a space curve to the plane, the projection of a singular point will still be singular. For generic projections, with high probability, the only artifactual singularities will be normal crossings (nodes), cusps or isolated points.

2.5 Fibers and Plotting Space Curves

A projection is not 1-1, in fact, if we restrict to projections $\mathbb{R}^n \longrightarrow \mathbb{R}^{n-1}$, which we will do in this subsection, the set of points mapping to a given point p in $\mathbb{R}^{n-1}$ is a line. We call this line *the fiber over p*. It is quite easy to calculate this from our original, not FLT, projection.

P is the original projection i.e. a $n \times (n-1)$ orthogonal matrix, p is a point in $\mathbb{R}^{n-1}$. The fiber is returned as a parameterized line with parameter t. Note that this function requires neither the curve or the list of variables.

In[111]:=
```
Pfiber[P_, p_, t_] := Module[{PC, q, m, n},
  {m, n} = Dimensions[P];
  If[! OrthogonalMatrixQ[P], Echo[P, " not orthogonal "];
    Abort[]];
  If[m ≠ n - 1, Echo[P, " not projection matrix "]; Abort[]];
  PC = Orthogonalize[N[Append[P, RandomReal[{-1, 1}, n]]]];
  q = Transpose[PC].Append[p, 0];
  q + t * PC[[n]]]
```

For example

In[58]:=
```
Pfiber[PRD, {1, 2}, t]
```

Out[58]=
```
{-0.58902 + 0.941656 t, 0.861327 + 0.30179 t, 1.97767 + 0.149021 t}
```

Our most important function in this subsection gives the set of points in a curve contained in the fiber over a point p, that is, the set of points on the curve projecting to p. This function is much easier than it looks however we want it to tell us if the number of points of the curve over p is different from 1. So this is both a diagnostic function as well as a function to find the actual points. Further, two important characteristics of this function are that it is very fast and it works even when the curve is defined by an overdetermined set of numerical polynomials. As we will see is these

properties that allow us to analyze general space curves.

F is the list of equations for the curve, possibly numerical and overdetermined, P is the original projection i.e. a $n \times (n-1)$ orthogonal matrix, p is a point in $\mathbb{R}^{n-1}$, X is the list of variables of F and tol is the tolerance which will often be weaker than our default tolerance.

In[112]:=
```
Ffiber[F_, P_, p_, X_, tol_] :=
  Module[{Pf, FF, FFs, sol, sol0, sol1, k, n, l, q, u, j, t0},
    n = Dimensions[P][[2]];
    k = Length[F];
    Pf = Pfiber[P, p, t734];
    FF = Chop[Expand[F /. Thread[X → Pf]], tol];
    t0 = RandomReal[{-1, 1}];
    FF = SortBy[FF, (# /. {t734 → t0}) == 0 &];
    If[AllTrue[FF, # == 0 &], Print["inf many sols at", p];
      Return[Fail]];
    sol = NSolve[FF[[1]], t734, Reals];
    If[Length[sol] == 0, Echo[p, "no point in fiber at"];
      Return[{}]];
    sol0 = t734 /. sol;
    j = 2;
    While[j ≤ k && Length[sol0] > 0 && (FF[[j]] /. {t734 → t0}) ≠ 0,
      sol = NSolve[FF[[j]], t734, Reals];
      If[Length[sol] == 0, Echo[p, "no point in fiber at"];
        sol0 = {}; Break[]];
      sol1 = t734 /. sol;
      sol0 =
        Flatten[
          Reap[Do[If[Norm[q - u] < tol, Sow[q]], {q, sol0}, {u, sol1}]][[2]]];
      j++];
    sol0 = DeleteDuplicates[sol0, Norm[#1 - #2] < tol &];
    If[Length[sol0] == 0, Echo[p, "no point in fiber at "]];
    If[Length[sol0] > 1, Echo[p, "multiple fiber points "]];
    Pf /. {t734 → #} & /@ sol0
  ]
```

This function returns the set of points in the fiber over p, possibly {}, in the curve as well as possible information. If no information is given there is a unique point given as a singleton set. When constructing a list of points in $\mathbb{R}^n$ over a List L in $\mathbb{R}^{n-1}$ in the curve use the form Flatten[Ffiber[F, P, #, X, tol]&/@L, 1]. If any warnings occur, eliminate the offending points in L and run again.

```
In[75]:= Ffiber[F2, Pxy, {1, Sqrt[2]}, {x, y, z}, dtol]
Out[75]= {{1., 1.41421 , 1.41421 }}

In[76]:= Ffiber[F2, Pxy, {0, 1}, {x, y, z}, dtol]
» no point in fiber at {0, 1}
Out[76]= {}

In[77]:= Ffiber[F2, Pxy, {0, 0}, {x, y, z}, dtol]
» multiple fiber points {0, 0}
Out[77]= {{0., 0., -1.}, {0., 0., 1.}}

In[78]:= Flatten[Ffiber[F2, Pxy, #, {x, y, z}, dtol] & /@ {{1, Sqrt[2]}, {1, -Sqrt[2]}}, 1]
Out[78]= {{1., 1.41421 , 1.41421 }, {1., -1.41421 , -1.41421 }}
```

2.5.1 Application to Example 2

Continuing with example 2 now using the default generic projecction

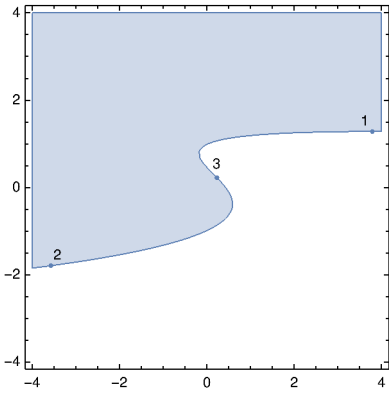
```
In[84]:= f = FLTgd[F2, FPRD, 4, {x, y, z}, {x, y}, dtol][[1]]
» Hilbert Function {1, 3, 6, 9, 12}
Out[84]= 1. - 2.37355 x + 0.0574214 x^2 + 0.000243617 x^3 - 2.18663 y +
    0.955595 x y + 0.0153028 x^2 y - 1.02271 y^2 + 0.320413 x y^2 + 2.2363 y^3

In[86]:= cp = DefCritPts2D [f, x, y]
         bp = {x, y} /. NSolve[{f, x^2 + y^2 - 16}, {x, y}, Reals]
Out[86]= {{-2.2102 × 10^13, 1.05558 × 10^12}, {-2.21017 × 10^13, 1.05557 × 10^12},
    {-210.502 , 15.985 }, {-210.499 , 15.9848 }, {0.228844 , 0.222392 }}

Out[87]= {{3.78876 , 1.2827 }, {-3.57476 , -1.79474 }}
```

This gives a list of 3 non-singular points for path tracing

```
In[88]:= ap = Append[bp, cp[[5]]]
Out[88]= {{3.78876 , 1.2827 }, {-3.57476 , -1.79474 }, {0.228844 , 0.222392 }}
```

In[91]:=
```
Show[RegionPlot[f > 0, {x, -4, 4}, {y, -4, 4}],
  ListPlot[Table[Labeled[ap[[i]], i], {i, 3}]]]
```

Out[91]=



In[138]:=
```
L1 = PathFinder2D[f, ap[[1]], ap[[3]], .3, x, y]
```

Warning : iteration limit reached

Out[138]=
```
{{3.78876, 1.2827}, {3.48876, 1.28151}, {3.18877, 1.279},
  {2.8888, 1.27499}, {2.58886, 1.2692}, {2.28897, 1.26132},
  {1.98917, 1.25088}, {1.68949, 1.23728}, {1.39003, 1.21965},
  {1.09096, 1.19665}, {0.792628, 1.16608}, {0.228844, 0.222392}}
```

Due to the sharp turn near point 3 we did not reach 3, jump from L1[[-2]] to ap[[3]].

In[139]:=
```
L1a = PathFinder2D[f, L1[[-2]], ap[[3]], .3, x, y]
```

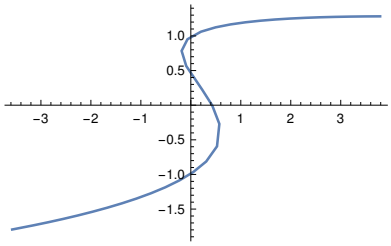
Out[139]=
```
{{0.792628, 1.16608}, {0.495874, 1.12389},
  {0.20353, 1.06116}, {-0.0668034, 0.952203}, {-0.184198, 0.78409},
  {-0.091331, 0.565073}, {0.228844, 0.222392}}
```

In[140]:=
```
L1 = Join[Drop[L1, -1], L1a];
L2 = PathFinder2DT[f, ap[[3]], ap[[2]], .3, x, y]
```

Out[141]=
```
{{0.228844, 0.222392}, {0.427728, -0.00265728}, {0.571122, -0.270517},
  {0.524209, -0.594063}, {0.307535, -0.813138}, {0.047808, -0.965813},
  {-0.227626, -1.08559}, {-0.510653, -1.18548}, {-0.79801, -1.27188},
  {-1.08811, -1.34844}, {-1.38009, -1.41745}, {-1.67342, -1.48044},
  {-1.96775, -1.53851}, {-2.26287, -1.59246}, {-2.5586, -1.64289},
  {-2.85484, -1.69028}, {-3.15149, -1.73501}, {-3.57476, -1.79474}}
```

In[142]:=
```
L = Join[L1, L2];
ListLinePlot [L]
```

Out[143]=



Now we can lift the plane curve to space

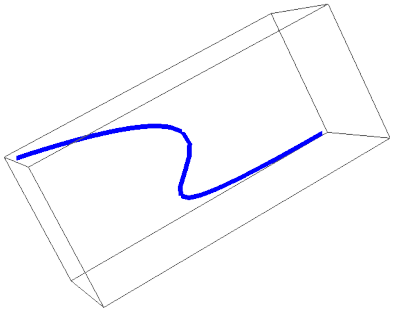
In[145]:=
```
{tm, SP} = Timing [Flatten [Ffiber [F2, PRD, ⌗, {x, y, z}, dtol] & /@ L, 1]]
```

Out[145]=
```
{0.176, {{2.5461, 4.79457, 1.88311}, {2.40338, 4.43381, 1.84482},
    {2.25635, 4.07167, 1.80454}, {2.10448, 3.708, 1.76195},
    {1.94709, 3.34259, 1.71671}, {1.78333, 2.97519, 1.66833},
    {1.61211, 2.60549, 1.6162}, {1.43193, 2.23305, 1.55947},
    {1.24077, 1.85733, 1.49692}, {1.03559, 1.47752, 1.42674},
    {0.811652, 1.09246, 1.34598}, {0.811652, 1.09246, 1.34598},
    {0.560652, 0.7004, 1.24926}, {0.265656, 0.298867, 1.12501},
    {−0.112896, −0.106332, 0.941862}, {−0.503973, −0.354944, 0.704292},
    {−0.806985, −0.354536, 0.439334}, {−0.993809, −0.0781947, 0.0786818},
    {−0.993809, −0.0781947, 0.0786818}, {−0.981149, 0.134711, −0.137299},
    {−0.854656, 0.325829, −0.38124}, {−0.559177, 0.371263, −0.663945},
    {−0.275648, 0.234601, −0.851089}, {−0.038588, 0.0378362, −0.980516},
    {0.170448, −0.184403, −1.08187}, {0.360629, −0.42066, −1.16646},
    {0.537078, −0.665864, −1.23979}, {0.702957, −0.91734, −1.30497},
    {0.860366, −1.1735, −1.36395}, {1.01078, −1.43331, −1.41802},
    {1.1553, −1.69608, −1.46809}, {1.29473, −1.9613, −1.51484},
    {1.42972, −2.22858, −1.55876}, {1.56079, −2.49766, −1.60025},
    {1.68836, −2.76828, −1.63962}, {1.86476, −3.15622, −1.69256}}}
```

Note there were no warnings so this is a good path which took less than .2 seconds to calculate from the plane curve path.

In[146]:=
```
Graphics3D [{Thick, Blue, Line[SP]}]
```

Out[146]=



2.5.2 Application to Cyclic 4

Here we sketch an analysis of the cyclic-4 curve using our method. Using lots of hindsight we go right to a pseudo-random projection $\mathbb{R}^4 \longrightarrow \mathbb{R}^2$ which factors through our default PRD.

In[104]:=
```
P43 = {{0.9749194263273511` , 0.13015457882712486` , -0.1507314794304482` ,
    -0.09935753060835883` }, {-0.1242169492514664` ,
    0.9851538622704206` , 0.09443037927788776` , -0.07158855105228731` },
  {0.17159792012482059` , -0.06309683839808246` ,
    0.9756771282924249` , 0.12094248269342328` }};
PC4 = PRD.P43
FPC4 = Join[Append[PC4, {0, 0, 0, 0}], {{0}, {0}, {1}}, 2];
```

Out[105]=
```
{{-0.415834 , 0.898428 , 0.135928 , -0.0378493 },
  {0.0369796 , -0.125668 , 0.981878 , 0.136948 }}
```

Trial and error require that $m \geq 6$

In[76]:=
```
h = FLTgd[C4, FPC4, 6, {w, x, y, z}, {x, y}, 1.*^-9]
```

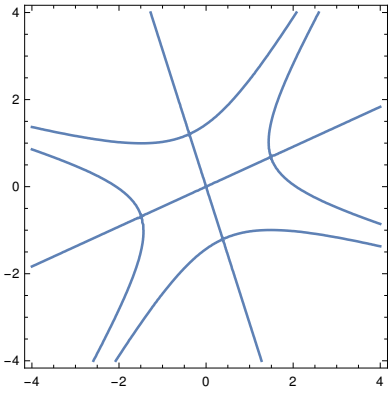
» Hilbert Function {1, 3, 6, 10, 15, 21, 27}

Out[76]=
$$\left\{-1.43989\ x^2 + 0.0789016\ x^6 + 2.68184\ x\,y + 0.323495\ x^5\,y + 1.\ y^2 - 0.558284\ x^4\,y^2 - 2.00039\ x^3\,y^3 + 1.90728\ x^2\,y^4 + 0.0432741\ x\,y^5 - 0.237496\ y^6\right\}$$

We plot h

In[77]:=
```
h = h[[1]];
ContourPlot[h == 0, {x, -4, 4}, {y, -4, 4}]
```

Out[78]=



It appears that we have two lines which will be components of the point curve V(h). From our Plane Curve book we find the infinite points associated to these lines and hence equations. The lines are

```
l1 = 1.1102230246251565`*^-16 +
       3.262382447969971` x + 1.038859871885362` y;
l2 = 0.` + 0.9545215062889317` x - 2.0817842202036227` y;
```

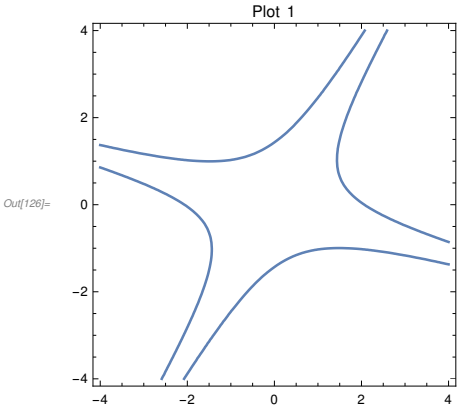
Using numerical division Ndivide from the plane curve appendix we get a degree 4 component which we then plot.

In[81]:=
```
h2 = -0.4623888112880111` + 0.025337572147313237` x^4 +
       0.15107565130494108` x^3 y + 0.11969969355723811` x^2 y^2 -
       0.3145167330145171` x y^3 + 0.10981547884135875` y^4;
h2 = Expand [h2 / h2[[1]]]
```

Out[82]=
$$1. - 0.0547971\ x^4 - 0.326729\ x^3\ y - 0.258872\ x^2\ y^2 + 0.6802\ x\ y^3 - 0.237496\ y^4$$

In[126]:=
```
ContourPlot [h2 == 0, {x, -4, 4}, {y, -4, 4}, PlotLabel → "Plot 1"]
```

Out[126]=



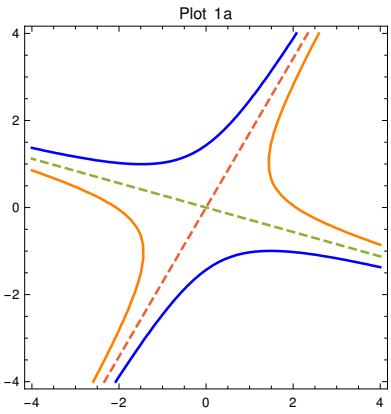
Again using our Plane Curve theory we analyze the two infinite points and find they have intersection multiplicity 2, from the Bezout bound on singularities we conclude that h2 is not irreducible. By tracing and interpolation we find two components

In[90]:=
```
c1 = -1.8416255421418233` -
       1.1221740125220023`*^-14 x - 0.43110213190591157` x^2 -
       1.285226441151662` x y + 0.8974896546959149` y^2;
c2 = 0.25107645431016734` - 0.05877394304523073` x^2 -
       0.1752202553917346` x y + 0.12235848989096718` y^2;
```

In[146]:=
```
asm1 = -0.1211216799073443` x - 0.4318223496972222` y;
asm2 = -1.3997630578307239` x + 0.8173735456714889` y;
```

In[149]:=
```
ContourPlot [{c1 == 0, c2 == 0, asm1 == 0, asm2 == 0}, {x, −4, 4}, {y, −4, 4},
  ContourStyle → {Blue, Orange, Dashed, Dashed}, PlotLabel → "Plot 1a"]
```

Out[149]=



In fact, these are two hyperbolas which meet the same dashed tangent lines, i.e. asymptotes, at infinity. Now the intermediate curve in $\mathbb{R}^3$ is

In[94]:=
```
C43 = {0.5153339676244552` x^2 + 0.026194632476755894` x y +
    0.000332870921431561` y^2 + 1.4357353065582186` x z +
    0.036489501034212016` y z + 1.` z^2, −0.6318496916664276` x^3 +
    0.561652322050535` x^2 y + 0.7325502919995197` x y^2 +
    0.018244750517105883` y^3 − 0.8801757382161406` x^2 z +
    0.8047596324567692` x y z + 0.9999999999999996` y^2 z,
  0.9999999999999997` + 0.5873103976744025` x^4 −
    1.1172938460939132` x^3 y − 0.5221632454139775` x^2 y^2 +
    0.2492408425958343` x y^3 − 0.028258224936660404` y^4 +
    0.849617204355458` x^3 z − 1.2674873405793519` x^2 y z};
```

which projects to h in $\mathbb{R}^2$ by our default projection PRD. By attempting to lift various points on the lines l1, l2 but not on the curve h2 we see that the fibers over these points contain no points in C43. So these are ghost lines. But points on h2 do lift by Ffiber to unique points in C43 and these in turn lift by Ffiber to points on C4, although in some cases we need to weaken the tolerance and maybe try several applications of Ffiber since that is a probabilistic function.

For example, perhaps run several times,

In[113]:=
```
p12 = {0.384516234120179` , −1.2075149555724785` }
p13 = Flatten[Ffiber[C43, PRD, p12, {x, y, z}, 1.*^−8], 1]
p14 = Flatten[Ffiber[C4, P43, p13, {w, x, y, z}, 1.*^−5], 1]
```

Out[113]=
```
{0.384516 , −1.20751 }
```

Out[115]=
```
{1., 1., −1., −1.}
```

The following is part of the lift of a trace of h2 to $\mathbb{R}^3$ containing the above

In[122]:=
```
L3 = {{1.647831384475943` , 0.5219061943329945` , -1.1924468972823483` },
      {1.3551629791066646` , 0.8380951220360562` , -0.9881185040331792` },
      {1.3046192322600219` , 0.9148462922973556` , -0.9532350890497597` },
      {1.3046192322600219` , 0.9148462922973555` , -0.9532350890497598` }};
```

Lifting to $\mathbb{R}^4$ gives

In[123]:=
```
L4 = Flatten[Ffiber[C4, P43, #, {w, x, y, z}, 1.*^-5] & /@ L3, 1]
```

Out[123]=
```
{{1.30801 , 0.764519 , -1.30801 , -0.764519 },
 {1., 1., -1., -1.}, {0.942706 , 1.06078 , -0.942706 , -1.06078 },
 {0.942706 , 1.06078 , -0.942706 , -1.06078 }}
```

We notice something important from this little bit of a trace, actually we looked at a bigger bit but this observation still holds, the first and 3rd coordinates are negatives of each other as are the second and 4th. Thus we recover a known fact about the curve C4, mainly it is contained in the 2-plane of $\mathbb{R}^4$ defined by equations

In[124]:=
```
plane2 = {w + y, x + z};
```

So in fact we have already seen the correct plot of C4 in plot1 and plot1a above, **this curve is a reducible curve which is the union of two hyperbolas meeting tangentially at infinity.** But we also see that, without changing the point set we can add the two equations of the plane to the equations C4. But applying our MBasis algorithm to clean up the basis we do get a surprise, after about 1 minute of CPU time,

In[128]:=
```
C4b = HBasis[Join[C4, plane2], 6, {w, x, y, z}, 1.*^-10]
```

» Hilbert Function {1, 2, 3, 4, 4, 4, 4}

Out[128]=
$$\left\{1.\, w + 1.\, y,\ 1.\, x + 1.\, z,\ -1. + 1.\, w^2\, x^2\right\}$$

In principle we can reverse the linear algebra involved to see that the equations in C4 are polynomial combinations of this simpler basis. So this is an improved, and better basis for the curve C4. In fact, notice

In[129]:=
```
h3 = FLTgd[C4b, FA1, 4, {w, x, y, z}, {x, y}, 1.*^-9]
```

» Hilbert Function {1, 3, 6, 10, 14}

Out[129]=
$$\left\{1. - 0.0547971\ x^4 - 0.326729\ x^3\, y - 0.258872\ x^2\, y^2 + 0.6802\ x\, y^3 - 0.237496\ y^4\right\}$$

recovers h2 directly without the ghost component. For those familiar with classical algebraic geometry this says the ideal of C4b is *radical*, whereas C4 was not. Further we now see that C4b, although not irreducible, is a complete intersection whereas C4 was not.

2.5.3 Example 3, naive curve in $\mathbb{R}^4$

The previous example shows how much we can learn from our method. Unfortunately the resulting curve was planar. Here, briefly is another example of a more interesting curve.

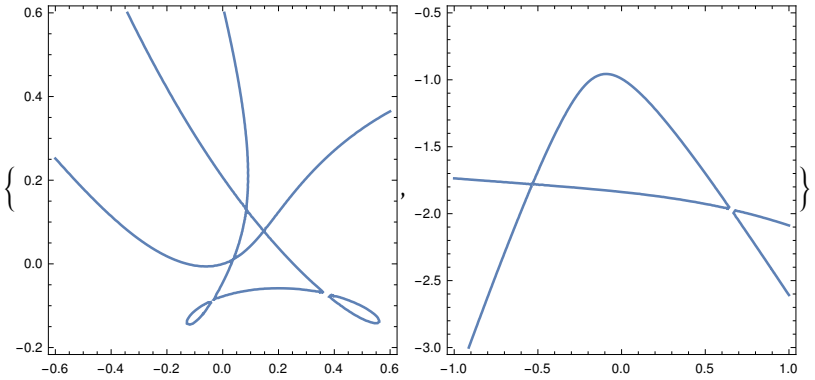
This curve was originally randomly generated as the intersection of 3 quadratic hypersurfaces of 4 space.

```
f31 = 3 w - 3 w^2 - x - x^2 + 3 y - 2 w y + 4 x y + 2 y^2 - 2 z - 4 w z + x z + 5 y z + 5 z^2;
f32 = -4 w + 3 w^2 + 2 w x + x^2 - 2 y - w y + 2 y^2 - 5 z - 4 w z + 4 x z - 2 y z - 5 z^2;
f33 = 2 w - 4 w^2 - 2 x - w x + 2 x^2 + y + 5 w y + 2 x y + y^2 + 3 z + w z + 5 x z - 2 y z + 2 z^2;
F3 = {f31, f32, f33};
```

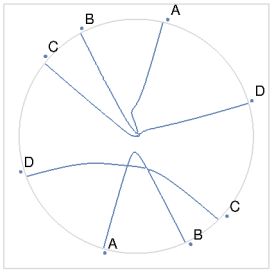
We first project with Pxyz, the simple projection setting $w = 0$. This gives a system of 7 equations of degree 5 in the three variables x, y, z. We will not reproduce this. We next project by our default pseudo-random projection PRD. We get a numerical plane curve of degree 8 with coefficients ranging in absolute value from 0.2 to 24 500. We call it g3 but do not give this here, it will eventually appear in my Space Curves book. The interesting parts are given below. Note that we have 7 singularities, all of which will turn out to be artifactual.

In[154]:=
```
{ContourPlot [g3 == 0, {x, -.6, .6}, {y, -.2, .6}],
 ContourPlot [g3 == 0, {x, -1, 1}, {y, -3, -.5}]}
```

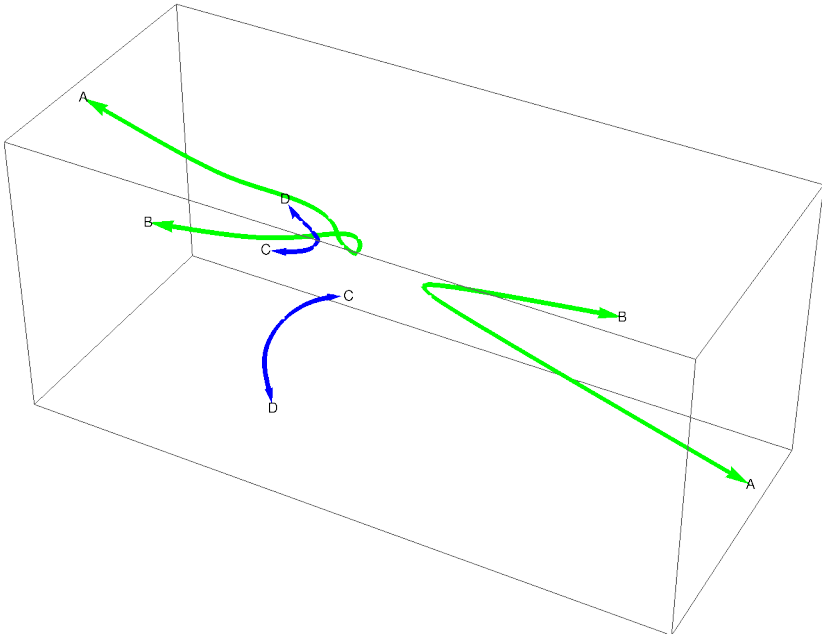
Out[154]=



Note the approximate position of the infinite points are



With difficulty we trace paths, first remembering that we must always trace into, but not out from singularities. Such delicate tracing is best done by our PathFinder2D using the ClosestPoint2D algorithm. But with a curve of degree 8 it is way too slow. PathFinderDE2D interpolates the curve with a piecewise linear curve but the points given are too approximate for Ffiber. So we use PathFinder2DT using normal planes which is a compromise. We are able to lift to $\mathbb{R}^3$ with Ffiber and get the following picture incorporating the curve in the union of the two regions above.



The blue and green curves are two different projective topological components. This is about as close as we can come to visualizing non-planar curves in $\mathbb{R}^4$. The points A, B, C, D represent the infinite points, consistent with the projection above, where each branch of the curve is heading. Do note that each of the singularities of the plane projections lift to two distinct points in $\mathbb{R}^4$ so the curve in $\mathbb{R}^4$ is non-singular. The plane curve is algebraically irreducible so the space curve also must be.

2.6 H-bases

In comutative algebra it is desirable that polynomial systems have certain nice properties. One such property is that of being an *H-basis,* a property defined by Macaulay in 1916 and largely forgotten since which has been making a resurgence lately since it is useful computationally. A system $\{h_1, \ldots, h_k\}$ is an H-basis if any polynomial combination

$$a_1\, h_1 + \cdots + a_k\, h_k$$

where the a_i are polynomials can be expressed in the form above with the total degree of each term $a_i\, h_i$ of degree less than or equal to the total degree of f.

Here is an example showing polynomials of degree 4 adding to a polynomial of degree 1

```
In[176]:= g1 = x - z + x z + y z;
          g2 = 2 x + y - z + x z + y z;
```

Let

```
In[193]:= a1 = (6 + 2 x + y + 5 z + x z + y z) / 6;
          a2 = (-x - 5 z - x z - y z) / 6;
```

Expanding

```
In[195]:= Expand[a1 * g1]
          Expand[a2 * g2]
```

$$\text{Out[195]=}\quad x+\frac{x^2}{3}+\frac{x y}{6}-z+\frac{3 x z}{2}+\frac{x^2 z}{2}+\frac{5 y z}{6}+\frac{2 x y z}{3}+\frac{y^2 z}{6}-\frac{5 z^2}{6}+\frac{2 x z^2}{3}+\frac{x^2 z^2}{6}+\frac{2 y z^2}{3}+\frac{1}{3} x y z^2+\frac{y^2 z^2}{6}$$

$$\text{Out[196]=}\quad -\frac{x^2}{3}-\frac{x y}{6}-\frac{3 x z}{2}-\frac{x^2 z}{2}-\frac{5 y z}{6}-\frac{2 x y z}{3}-\frac{y^2 z}{6}+\frac{5 z^2}{6}-\frac{2 x z^2}{3}-\frac{x^2 z^2}{6}-\frac{2 y z^2}{3}-\frac{1}{3} x y z^2-\frac{y^2 z^2}{6}$$

are 4^{th} degree polynomials, but their sum is a linear polynomial.

```
In[197]:= f = Expand[a1 * g1 + a2 * g2]
```

```
Out[197]= x - z
```

which cannot be written as a linear combination of g1, g2. This is what we **don't** want to see in an H-basis.

This concept of H-basis can best be described in terms of Sylvester matrices. Consider a polynomial system $\{g_1, \ldots, g_k\}$ in several variables. The rows of the Sylvester matrix of order m are the coefficients of polynomial obtained by multiplying one of the g_i by a monomial a_i in the variables so that the resulting $a_i\, g_i$ is of degree no larger than m. Thus LINEAR combinations of the rows give a vector space of polynomials of degree m or less. The H-basis condition says that this vector space is the vector space of ALL polynomials of degree m or less obtainable by polynomial combinations of the g_i. In the example above the Sylvester matrix of order 1 of the system $\{g_1, g_2\}$ has no rows, it generates the trivial $\{0\}$ vector space, but we saw that $x - z$ is a polynomial combination of $\{g_1, g_2\}$ that is not 0.

It is well known that every system of equations can be described by an H-basis. *H-bases are the bases that should be used with Sylvester matrices.*

Often we will find ourselves with a large system of equations describing a curve. As you have seen, an ongoing theme is that in general we can not find an acceptable system of $n - 1$ equations to describe an algebraic curve in $\mathbb{R}^n$. But we would still like as small a system as possible, hopefully a H-basis. Our procedure `hBasisMD` given in GlobalFunctionsMD attempts to do

just this.

As an example consider the large discussed below in 4 variables.

In[133]:=
```
B4 = {x3^2 x4^2 - x2 x4^3, x3^3 x4 - x1 x4^3, x3^4 - x4^3, x2 x3 x4^2 - x1 x4^3, x2 x3^2 x4 - x4^3,
      x2 x3^3 - x3 x4^2, x2^2 x4^2 - x4^3, x2^2 x3 x4 - x3 x4^2, x2^2 x3^2 - x2 x4^2, x2^3 x4 - x2 x4^2,
      x2^3 x3 - x1 x4^2, x2^4 - x4^2, x1 x3 x4^2 - x4^3, x1 x3^2 x4 - x3 x4^2, x1 x3^3 - x2 x4^2,
      x1 x2 x4^2 - x3 x4^2, x1 x2 x3 x4 - x2 x4^2, x1 x2 x3^2 - x1 x4^2, x1 x2^2 x4 - x1 x4^2,
      x1 x2^2 x3 - x4^2, x1 x2^3 - x3 x4, x1^2 x4^2 - x2 x4^2, x1^2 x3 x4 - x1 x4^2, x1^2 x3^2 - x4^2,
      x1^2 x2 x4 - x4^2, x1^2 x2 x3 - x3 x4, x1^2 x2^2 - x2 x4, x1^3 x4 - x3 x4, x1^3 x3 - x2 x4,
      x1^3 x2 - x1 x4, x1^4 - x4, x3^2 x4 - x2 x4^2, x3^3 - x1 x4^2, x2 x3 x4 - x1 x4^2, x2 x3^2 - x4^2,
      x2^2 x4 - x4^2, x2^2 x3 - x3 x4, x2^3 - x2 x4, x1 x3 x4 - x4^2, x1 x3^2 - x3 x4, x1 x2 x4 - x3 x4,
      x1 x2 x3 - x2 x4, x1 x2^2 - x1 x4, x1^2 x4 - x2 x4, x1^2 x3 - x1 x4, x1^2 x2 - x4,
      x1^3 - x3, x3^2 - x2 x4, x2 x3 - x1 x4, x2^2 - x4, x1 x3 - x4, x1 x2 - x3, x1^2 - x2};
```

This system has 53 equations.

In[139]:=
```
B4H = hBasisMD[B4, 4, {x1, x2, x3, x4}, dTol]
```

» Initial Hilbert Function {1, 4, 4, 4, 4}

» Final Hilbert Function {1, 4, 4, 4, 4}

Out[139]=
$$\{1.\,x1^2 - 1.\,x2,\ 1.\,x1\,x2 - 1.\,x3,\ 1.\,x1\,x3 - 1.\,x4,\ 1.\,x2^2 - 1.\,x4,\ 1.\,x2\,x3 - 1.\,x1\,x4,\ 1.\,x3^2 - 1.\,x2\,x4\}$$

We now have 6 equations which does form a H - basis. For this latter claim we also have a function hBasisMDQ which checks whether or not s system is a H-Basis by comparing it to a degree Gröbner Basis.

In[140]:=
```
hBasisMDQ[B4H, B4H, {x1, x2, x3, x4}, dTol]
```

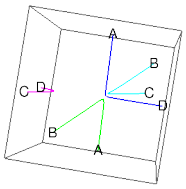
» {size of Groebner Basis, degrees} {8, {2, 3, 4}}

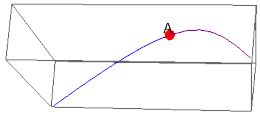
Out[140]= True

On our GlobalFunctions notebook we consider the Liu example which is relevant because it defines a curve. This system is very sensitive to perturba - tion and is best described by the H-basis calculated there

In[165]:=
```
hLiu = {-u - x - w z + y z, u + w y - x y - z,
        u + w x - y - w z, u - w - x y + x z, -u w + w^2 + u x + x^2 - u y + y^2 - u z + z^2};
```

It is interesting to note that this curve projects faithfully to the curve in $\mathbb{R}^3$ given by the plots





where A,B,C,D are the infinite points. Note that infinite point A is a singular point of multiplicity 3 even though it appears smooth. The curve is actually an oval.

To check this last statement we projectively "Rotate" our curve by applying FLTMD to the curve hLiu using matrix

$$\text{rot}=\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$$

Getting

```
LiuRot = {0.5 x + 0.5 u x - 0.5 w x + 0.5 y + 0.5 u y + 1. w z - 0.5 y z,
    0.5 x + 0.5 u x + 0.5 w x - 0.5 y - 0.5 u y - 1. w y + 1. x y + 1. z + 1. u z - 0.5 y z,
    1. u w - 1. w ^ 2 - 1. u x - 1. x ^ 2 + 1. u y - 1. y ^ 2 + 1. u z - 1. z ^ 2,
    1. w + 1. w ^ 2 + 0.5 x + 1.5 u x + 0.5 w x + 1. x ^ 2 - 0.5 y - 1.5 u y + 1. x y + 1. y ^ 2 - 1. u z -
      1. x z - 0.5 y z + 1. z ^ 2, 1. u + 1. u ^ 2 + 0.5 x + 0.5 u x + 0.5 w x - 0.5 y - 0.5 u y - 0.5 y z};
```

In[163]:=
```
fltiMD[{0, 0, 0, 0, -1, 0}, rot]
```

Out[163]=
```
{0, 0, 0, 0, -1}
```

In[164]:=
```
tangentVectorMD [LiuRot, {0, 0, 0, 0, -1}, {x, y, z, w, u}]
```

» Hilbert Function {1, 3, 3, 3, 3}

» No unique tangent vector at {0, 0, 0, 0, -1}

Generally speaking we expect FLTMD to take H-bases to H-bases. Unfortunately Groebner bases may not behave so well. So while hLiu , with a 14 member Groebner basis of maximum degree 5, checks easily under hBasisMDQ the system LiuRot does not, having a Groebner basis of 28 members of maximal degree 11. Using the option useF→True we can check piecewise, for example one term of this basis of degree 6 is

In[197]:=
```
g6a =
4.2045454545454115` u + 18.318181818181902` u^2 + 32.2954545454551` u^3 + 27.000000000000572` u^4 + 9.363636363636536` u^5 +
  0.5454545454545384` u^6 + 1.522727272726931` w - 9.454545454544949` u w - 26.31818181818118` u^2 w -
  27.659090909091205` u^3 w - 13.954545454545581` u^4 w - 1.6363636363636287` u^5 w + 9.34090909090894` w^2 +
  29.227272727272855` u w^2 + 32.70454545454616` u^2 w^2 + 16.13636363636396` u^3 w^2 + 1.9999999999999885` u^4 w^2 +
  3.045454545454712` w^3 + 2.02272727272734` u w^3 - 2.6363636363635967` u^2 w^3 - 1.4545454545454704` u^3 w^3 +
  1.9318181818181366` w^4 - 1.318181818181841` u w^4 - 0.09090909090908895` u^2 w^4 + 2.0909090909091623` w^5 +
  0.3636363636363631` u w^5 - 0.2727272727272781` w^6 - 4.818181818181321` y - 13.20454545454383` u y -
  8.386363636362487` u^2 y + 0.9318181818180364` w y - 0.9090909090909817` z + 1.6136363636353614` u z -
  1.0454545454554822` u^2 z - 8.522727272727355` u^3 z - 5.318181818181915` u^4 z - 0.3636363636363581` u^5 z -
  10.38636363636311` w z - 16.545454545454156` u w z - 4.454545454545404` u^2 w z + 3.5454545454546467` u^3 w z +
  0.9090909090909113` u^4 w z - 4.204545454545125` w^2 z - 0.20454545454590822` u w^2 z - 4.318181818182016` u^2 w^2 z -
  0.6363636363636451` u^3 w^2 z - 8.386363636364031` w^3 z - 2.227272727272733` u w^3 z + 0.5454545454545567` u^2 w^3 z -
  5.181818181818323` w^4 z + 1.` w^5 z - 3.5681818181813156` y z - 1.4090909090909842` z^2 - 2.818181818182394` u z^2 -
  3.659090909091062` u^2 z^2 - 3.7272727272719357` w z^2 + 1.4545454545453542` w^2 z^2 - 2.2499999999996363` z^3
```

In[198]:=
```
Timing[hBasisMDQ[{g6a}, LiuRot, {x, y, z, u, w}, 1.*^-10, useF → True]]
```

» {size of Groebner Basis, degrees} {1, {6}}

Out[198]= {5.32884, True}

Already this takes over 5 seconds to check. The 9th degree polynomial in this Groebner basis with 91 terms takes almost 4 minutes to check. To check the claim that LiuRot is an H-basis will take a very long time on your computer!

A system which is not a curve which has a difficult H-basis is the well known example called Caprasse4 which has only isolated solutions

```
cap4 = {x^2 z + 2 x y t - 2 x - z,
    -x^3 z + 4 x y^2 z + 4 x^2 y t + 2 y^3 t + 4 x^2 - 10 y^2 + 4 x z - 10 y t + 2,
    2 y z t + x t^2 - x - 2 z,
    -x z^3 + 4 y z^2 t + 4 x z t^2 + 2 y t^3 + 4 x z + 4 z^2 - 10 y t - 10 t^2 + 2};
```

Running our `hBasisMQ` function

In[194]:=
```
cap4a = hBasisMD[cap4, 6, {x, y, z, t}, 1.*^-10]
```

» Initial Hilbert Function {1, 4, 10, 17, 21, 29, 29}

Warning: HBasis did not terminate normally

» Final Hilbert Function {1, 4, 10, 17, 22, 24, 28}

Out[194]= $\{-1.\,x + 1.\,t\,x\,y - 0.5\,z + 0.5\,x^2\,z,\ -0.5\,x + 0.5\,t^2\,x - 1.\,z + 1.\,t\,y\,z,$
$-8.\,x - 8.\,t^2\,x - 5.\,z + 1.\,t^2\,z + 12.\,x^2\,z + 8.\,x\,z^2 - 1.\,z^3\}$

This is clearly not right as one would need at least 4 equations to get isolated solutions

In[195]:= **Length [NSolve [cap4a]]**

--- NSolve : Infinite solution set has dimension at least 1. Returning intersection of solutions with $-\frac{92003\,t}{83857}+\frac{70897\,x}{83857}-\frac{88092\,y}{83857}-\frac{131613\,z}{167714}==1.$

--- NSolve : Infinite solution set has dimension at least 2. Returning intersection of solutions with $-\frac{58857\,t}{39613}+\frac{135062\,x}{118839}-\frac{35780\,y}{39613}+\frac{168848\,z}{118839}==1.$

Out[195]= 1

But running **hBasisMDQ[cap4, cap4, {x, y, z, t}, 1.*^-10]**

In[196]:= **hBasisMDQ [cap4, cap4, {x, y, z, t}, 1.*^-10]**

» {size of Groebner Basis, degrees} {30, {3, 4, 5, 6}}

» Problem at poly i degree k {3, 2}

Out[196]= False

we get a Groebner Basis which is also a H-Basis of 30 equations. But running this 12 times, each time adding the "problem" polynomial to cap4 does give us a H-basis of 16 polynomials of degree 6 or less. Now using **hBasisMQ** to check the smaller set obtained by deleting one polynomial at a time we find that 2 of the polynomials we added are not needed since they are implied by latter additions. So we end up with a H-basis of 14 terms. I do not know if a smaller H-basis is possible.

2.7 Numerical Irreducible Decomposition.

In classical algebraic geometry algebraic sets can be decomposed into irreducible components, that is, subsets which are themselves algebraic sets but which are not the union of smaller algebraic sets. In principle the irreducible components should be described by a system of multivariable polynomial equations. In the case of real algebraic geometry the equations could conceivably have complex coefficients.

In numerical algebraic geometry this is usually not attempted. While some numerical solvers, eg. Bertini, can identify complex components and mark them with *witness sets* but equations are not attempted.

The methods we have used in the last subsection can at least identify topological components and plot them with piecewise linear paths which avoid singular points, such paths are contained in irreducible components. In the plane one can then use interpolation to find an equation. This is more difficult for space curves since several equations are needed and even the number of equations is not known in advance. Also polynomials in several variable can have many terms which requires many points which can lead to numerical errors.

In this book we will discuss *dual interpolation* which can sometimes find a system of equations for a component based on knowledge of a few points of the component, an idea I developed in 2011, see for example www.barryhdayton.space/AG11.

As an example consider the curve given by $\{y^2-x^3-x^2,\ x\,z-y\}$. One can easily see that any point $\{0, 0, z\}$ satisfies this equation, in fact the entire line with these points

consists of double points. On can look for the complementary component. Projecting this curve to the plane with the standard $z \to 0$ projection gives the nodal curve $y^2 = x^3 + x^2$. By interpolating the lifts of the non-singular points of this curve to $\mathbb{R}^3$ we find the system defining the non-singular component

In[3]:= F2

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

which we discussed earlier.

A more general problem is finding the intersection or union of curves. Generally the intersection is just a finite set of points which can be easily found by **NSolve**. But if two curves have a common component then then a system for the intersection is given by the union, **Join**, of the systems. But this is usually not the most efficient system possible so our function **hBasisMD** can try to find a better system. Unfortunately the current function **hBasisMD** works best with exact systems, otherwise one needs to play with the tolerance and order.

To find the union of two curves the best approach is to **Join** the duals of the Sylvester matrices of the two systems column-wise. Then one takes the local (left) dual of this and converts back to a system multiplying by **mExpsMD** . The resulting system can be refined using **hBasisMD**.

Example: The union of three lines.

```
In[139]:= L1 = {x - y, z};
          L2 = {y - z, x};
          L3 = {-1 + x, 3 - 3 y + z}
Out[141]= {-1 + x, 3 - 3 y + z}

In[142]:= S1 = sylvesterMD[L1, 3, {x, y, z}];
          S2 = sylvesterMD[L2, 3, {x, y, z}];
          S3 = sylvesterMD[L3, 3, {x, y, z}];
In[145]:= DS1 = Chop[dualMatrix[S1, dTol]];
          DS2 = Chop[dualMatrix[S2, dTol]];
          DS3 = Chop[dualMatrix[S3, dTol]];
In[148]:= DSU = Join[DS1, DS2, DS3, 2];
          SU = Chop[localDualMatrix[DSU, dTol], dTol];
In[150]:= LU = SU.mExpsMD[3, {x, y, z}];
          Length[LU]
Out[151]= 10
```

In[152]:= LUH = hBasisMD [LU, 3, {x, y, z}, dTol]

» Hilbert Function {1, 3, 3, 3}

Out[152]= $\{-0.5\,x - 1.\,x^2 + 0.5\,y + 1.\,x\,y - 0.5\,z,\ -1.5\,x + 1.5\,y - 1.5\,z + 1.\,x\,z,$
$3.\,x - 3.\,x^2 - 3.\,y + 3.\,y^2 + 3.\,z - 4.\,y\,z + 1.\,z^2\}$

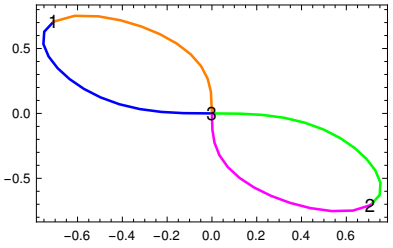
This system needs 3 equations for an H-basis.

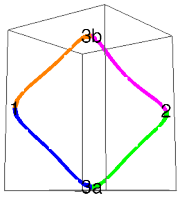
2.8 Fundamental Theorem

In my plane curve book I introduce the Fundamental Theorem. This does carry over to space curves in the general case. Again projection to the plane and fiber lifting can be used to find a graph of the space curve. Singular points in the plane projection may lift to several points so the corresponding vertex will be the image of several different vertices, but the edges will project to distinct edges in the base given a random enough projection. A simple example is a curve called E6a

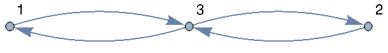
```
E6a = E6a = {x + y - x z + y z,
    -x - 2 x^2 y - 2 x y^2 - 2 y^3 + x z + 2 x^3 z, -1 + 6 x^2 +
     8 x y + 4 y^2 - 4 x^2 z + z^2 + 2 x^2 z^2, x^4 + x y + y^4};
```

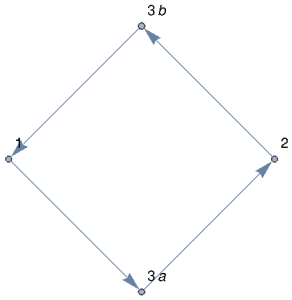
Projecting with the non - generic projection $z \to 0$ gives the last equation $x^4 + x\,y + y^4 = 0$. Plotting this with path tracing and lifting gives





where the segments of the space curve project the same colored segments of the plane curve. In the graphs vertices 1,2 in space go to 1,2 in the plane and vertices 3a,3b go to 3 in the plane.





Singularities in space will project to singularities in the plane but under a generic projection different singularities go to different singularities in the plane so the whole singularity will just lift. Thus we have the Fundamental theorem

Each space curve can be described by a graph with even vertices.

We pictured the graphs as directed graphs. While we saw that there was a natural direction, given a fixed equation, in the plane the directions in space may be arbitrary. But since each component of a graph with even vertices is a cycle, by Euler, the edge directions can be chosen so that following these directions allows one to get back to the starting point.

For a second example we consider the curve in 4-space F3 given above.

The infinite points of F3 are given by

In[130]:=
```
infF3 = infiniteRealPointsMD [F3, {x, y, z, w}, 1.*^-10 ]
```

Out[130]=
```
{{-0.538213 , 0.794671 , 0.245387 , 0.136415 , 0},
 {-0.750913 , -0.416097 , 0.208369 , 0.468589 , 0},
 {0.868006 , 0.134921 , -0.380594 , 0.28898 , 0},
 {0.0882663 , 0.729859 , -0.548269 , -0.398641 , 0}}
```

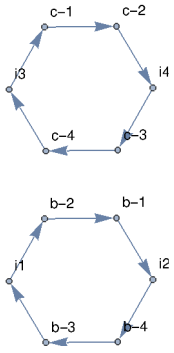
labeled by i1,…i4 which project to infinite plane points

```
{{-1.25675 , 1.55582 }, {1.07143 , 1.6888 },
 {-1.63582 , 1.1507 }, {-1.68853 , -1.07186 }}
```

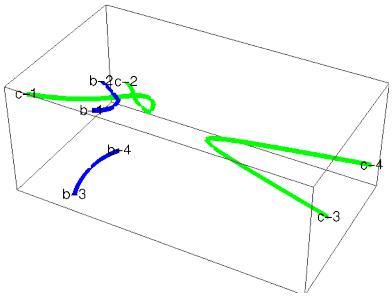
Using the Fundamental Theorem in the plane we can infer the following graph

In[134]:=
```
Graph [{"c-2" → "i4", "i4" → "c-3", "c-3" → "c-4", "c-4" → "i3",
    "i3" → "c-1", "c-1" → "c-2", "b-1" → "i2", "i2" → "b-4",
    "b-4" → "b-3", "b-3" → "i1", "i1" → "b-2", "b-2" → "b-1"},
  VertexLabels → "Name ", ImageSize → Small ]
```

Out[134]=



which compares to the plot above with endpoints labeled.



2.8.1 Ovals and pseudo lines

We can decompose the graph into loops, that is subgraphs where each vertex has order 2. In particular these are closed. If the curve is non-singular then each loop represents a topological component, the converse may not be true because of the existence of cusps etc. In the case of disjoint loops the decomposition is unique, but if there exist vertices of higher even order the decomposition is not unique.

The part of the curve represented by a loop is topologically a simple closed sub-curve. We can distinguish two types. If the closed sub-curve contains an even number of real infinite points, by multiplicity, we call it a *oval.* Otherwise we call it a *pseudo-line.*

Since any hyperplane can be considered in some specialization to be the infinite points then equivalently one can intersect the curve with any hyperplane and see if the number of intersection points is even or odd to determine whether we have an oval or pseudo-line. This is especially useful if the original graph has a vertex representing an infinite point of degree 4 or more, since there will be more than one loop with this vertex but the intersection multiplicity of the original curve with the infinite line at this point will count intersections with all loops through this vertex.

Of course, if the curve has bounded real part, then a far away hyperplane will miss the curve completely so it is automatically an oval. One difference between the space and plane situation is that while a non-singular plane curve can have at most one pseudo-line, a non-singular space curve can have more than one skew pseudo-line.

Pseudo-lines are not necessarily preserved under projections, in fact loops are not preserved. But one may still be able to get information from the projection.

The example we use is Case 8 from [Tu, Wang, Mourrain, Wang, *Using Signature sequences to classify intersection curves of two quadrics,* Computer Aided Geometric Design,26 (2009), 317-335]. Further details may appear later in this summary.

In[178]:= $\text{case8} = \{x\,y + z,\ 1 + 2\,x\,y + y^2 - z^2\};$

Checking infinite points

In[171]:= $\text{IP} = \text{infiniteRealPoints3D}\ [\text{case8}, \{x, y, z\}]$

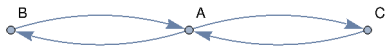
Out[171]= $\{\{0.,\ -0.707107,\ 0.707107,\ 0\},\ \{1.,\ 0.,\ 0.,\ 0\},$
$\{1.,\ 0.,\ 0.,\ 0\},\ \{0.,\ 0.707107,\ 0.707107,\ 0\}\}$

The second infinite point is singular which is why it repeats. We will label these distinct points C, A, B respectively. It can be shown that a graph for this 3 dimen-

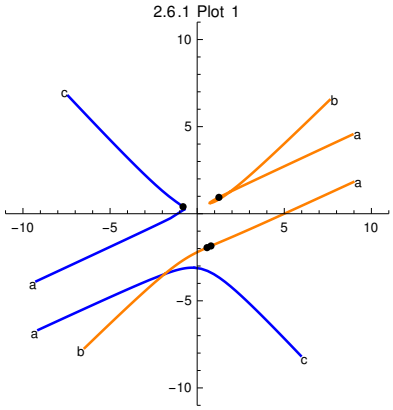
sional curve is

In[174]:= Graph[{"A" → "C", "C" → "A", "A" → "B", "B" → "A"}, VertexLabels → "Name"]

Out[174]=



To get an idea of what this curve actually looks like we project it to the plane using our default pseudo-random projection **fprd3D** obtaining



where c, a, b represent the infinite projections of C, A, B respectively. The intersections in this plot are artifactual, that is they are not in the original curve.

Since A is infinite it is impossible to attribute them to the individual loops ABA and ACA. Therefore we take a pseudo-random plane intersecting both loops

In[180]:= plane = 0.4645861830018325` + 0.1244823462922618` x +
0.847266521772098` y − 0.22539579656588946` z;

This plane intersects the space curve in

In[182]:= sol261 = {x, y, z} /. NSolve[Append[case8, plane]]

Out[182]= {{−4.38804, −0.575876, −2.52697}, {−3.90255, −0.655679, −2.55882},
{−3.73668, 0.112037, 0.418644}, {0.941646, −0.549126, 0.517083}}

These points project to the points

In[183]:= fltMD[#, fprd3D] & /@ sol261

Out[183]= {{0.790819, −1.84985}, {0.566653, −1.94661},
{1.24712, 0.93915}, {−0.810315, 0.402654}}

shown as black dots on Plot 1 above. We see that 3 lie in the orange curve aba while only the last one lines in aca. Thus we conclude that ABA and ACA are both pseudo-lines as reported in the paper quoted above.

The reader should note that although these points are not collinear any line in the plane will intersect both the blue and orange part in an odd number of points, counted by multiplicity. On the other hand the reader should note that in the projec-

tion there are 3 singularities and the non unique decomposition of the graph could have 3 or 4 loops, some of which will be ovals so projections do not directly answer the question for the space curve.

2.9 Bézout's Theorem

In plane curve theory Bézout's theorem counts the number of complex projective intersection points counting multiplicity. More generally in multiple variables Bezout's theorem counts the number of complex projective zeros by multiplicity of a *zero dimensional* system, that is, a non-linear system of equations with only isolated solutions, that is the solution set does not contain a curve, surface etc. It is well known that the solution set in this case must be finite. The case of a square zero dimensional system, eg. n equations in n unknowns is a classical result, namely if the equations have degree $d_1, \ldots, d_n$ then there are $d_1 * d_2 * \cdots * d_n$ solutions by multiplicity. There are no simple proofs, one must use advanced algebraic geometry.

In our case we generally have more equations than variables. In this case it is more complicated, typically adding more equations decreases the number of solutions. In this section we suggest a different solution, the *nullity* of large Sylvester matrices. Specifically we mean by nullity the difference between the number of columns and the matrix rank. While we do not claim a proof we will show by examples that this nullity is at least the number of distinct projective solutions. The reader wanting a proof might look at the paper [Telen, Mourrain, van Barel, *Solving polynomial systems via truncated normal forms,* Siam J. Matrix Anal. Appl. Vol39 no3 (2018) pp. 1421-1447] for ideas on how to prove the existence part of the theorem.

First we need two new functions. These produce *dual* vectors to Sylvester matrices for each affine or infinite point of the complex projective space $\mathbb{CP}^n$. I emphasize that the dual vectors are independent of any system, they depend only on the points and an *order* m. The variables are essentially dummies here, any set of n variables will do but since we are working with certain ones it is most convenient to use those.

In[105]:=
```
aVecMD[p_, m_, X_] := mExpsMD[m, X] /. Thread[X → p]
iVecMD[p_, m_, X_] := Module[{lS, lh},
    lS = Length[expsMD[Length[X], m]];
    lh = Length[hExpsMD[Length[X], m]];
    Join[Table[0, {lS - lh}], mhExpsMD[m, X] /. Thread[Append[X, ##t] → p]]]
```

I start with an example of a square integer system of 3 equations in 3 unknowns each of which has degree 2. which has both affine and infinite solutions.

In[216]:=
```
Clear[F]
F = {5 - 11 x^2 - 3 y - 17 x y - 17 y^2 + 4 z + 2 x z + 17 y z - 2 z^2, 1 + 5 x + 41 x^2 - 2 y + 59 x y + 53 y^2 +
      4 z - 8 x z - 59 y z + 8 z^2, 1 + 3 x + 9 x^2 + 3 y - 5 x y - 31 y^2 + 5 z - 4 x z + 5 y z + 4 z^2};
```

Note the sum of the degrees is 6 and the product is 8. We first find the complex affine and infinite solutions.

In[218]:=
```
asolF = {x, y, z} /. NSolve[F]
```

Out[218]=
```
{{-8.55422, 7.35644, 6.84027}, {-0.0649037, -0.112053, -1.15724},
  {-0.44003 - 0.234104 i, -0.0206914 - 0.232533 i, -0.990669 - 0.122708 i},
  {-0.44003 + 0.234104 i, -0.0206914 + 0.232533 i, -0.990669 + 0.122708 i}}
```

In[221]:=
```
isolF = infinitePointsMD [F, {x, y, z}, dTol]
```

Out[221]=
```
{{-0.298531 - 0.614054 i, 0.114688 - 0.027502 i, -1.1404 + 0.15798 i, 0},
  {-0.298531 + 0.614054 i, 0.114688 + 0.027502 i, -1.1404 - 0.15798 i, 0},
  {0.241102, 0.363299, 0.899935, 0}, {-0.645934, 0.552577, 0.526715, 0}}
```

So we have 2 real and 2 complex affine solutions and also 2 real and 2 complex infinite solutions. We calculate the dual vectors of order 6 to these points.

In[231]:=
```
adualsF = aVecMD [#, 6, {x, y, z}] & /@ asolF;
idualsF = iVecMD [#, 6, {x, y, z}] & /@ isolF;
dualsF = Transpose [Join[adualsF, idualsF]];
Dimensions [dualsF]
MatrixRank [dualsF]
```

Out[234]=
```
{84, 8}
```

Out[235]=
```
8
```

Note the columns are independent. Now we compare with the Sylvester matrix.

In[226]:=
```
S6F = sylvesterMD [F, 6, {x, y, z}];
Dimensions [S6F]
MatrixRank [S6F]
```

Out[227]=
```
{105, 84}
```

Out[228]=
```
76
```

Thus the nullity is 84-76=8 as expected. Now to check our dual vectors

In[230]:=
```
SingularValueList [S6F.dualsF]
```

Out[230]=
```
{7.36336 × 10^-8, 8.95816 × 10^-13, 2.97477 × 10^-13, 1.36979 × 10^-13,
  5.884 × 10^-14, 3.95082 × 10^-14, 7.41627 × 10^-15, 3.05139 × 10^-15}
```

we see that this is numerically the zero matrix. Since dualsF has 8 independent columns we conclude that these columns form a basis for the nullspace of S6F. The reader should be aware that although there are many linear algebra methods to calculate a nullspace they will not give this basis, essentially one must use non-linear methods, such as system solving, to obtain this particu-

lar basis.

We have illustrated our theorem:

Suppose F is an zero dimensional system of r non-linear real or complex polynomial equations in $n \le r$ variables $X = \{x_1, \ldots, x_n\}$. Suppose the equations have degrees $d_1, \ldots, d_n$ and $m = d_1 + d_2 + \cdots + d_n$. Let c_a be the number of distinct complex affine solutions and c_{inf} be the number of distinct complex infinite solutions, $c = c_a + c_{inf}$. Further let $k \ge m$ and for each affine solution y_j let $v_i = aVecMD[y_i, k, X]$ and for each infinite solution z_j let $w_j = iVecMD[z_j, k, X]$. Then $v_1, \ldots, v_{c_a}, w_1, \ldots, w_{c_{inf}}$ as column vectors, are contained the nullspace of the Sylvester matrix of F of order k.

Remarks: I conjecture that these vectors v_i, w_i are independent and that if there are multiple solutions there are additional vectors as in the 2D version to fully span the nullspace. So the dimension of the nullspace will count the number of complex projective solutions according to multiplicity.

The zero-dimensional hypothesis is non-trivial. In the r = n = 2 case this is equivalent to the usual hypothesis of no common divisor. In the general case the best way to test this hypothesis is to solve the system using `NSolve`. If the hypothesis is not true an information notice starting with

```
NSolve: Infinite solution set has dimension at least 1 ....
```

will appear.

The classical version r = n says that for the zero-dimensional hypothesis the total number of complex projective solutions is $d_1 * \cdots * d_n$, called the Bézout number. This is a deep result of algebraic geometry with no easily accessible proof. Note that if r > n the the count will generally be smaller.

The formula $m = d_1 + d_2 + \cdots + d_n$ is somewhat conjectural at this point. It is advised that one calculate the nullity of both the Sylvester matrix of order m and order m + 1. If these are not the same then either the zero-dimensional hypothesis or the conjecture on m does not hold. In the latter case this nullity will still stabilize at some point and that is the number to use.

Here is an application to curve theory with a non-square system. Consider the Shen-Yaun example in H-basis form

In[107]:=
```
SY = {3. + 6. x + 3. x ^ 2 - 4. y - 3. x y + 1. y ^ 2 - 1. z - 1. x z,
    -1. x - 1. x ^ 2 - 1. z + 1. y z, 3. x + 3. x ^ 2 - 1. x y - 3. x z + 1. z ^ 2};
```

This is a square system of 3 equations of degree 2 in 3 unknowns. But it is non zero-dimensional so Bezout does not hold.

In[109]:= **NSolve[SY]**

--- NSolve : Infinite solution set has dimension at least 1. Returning intersection of solutions with $\frac{40299\ x}{38602} - \frac{69046\ y}{57903} - \frac{142003\ z}{115806} == 1.$

Out[109]= $\{\{x \to -1.01508, y \to 0.994216, z \to -2.64656\},$
$\{x \to -2.78473 + 0.767326\,i, y \to -2.16878 - 0.716577\,i, z \to -1.0773 + 1.35012\,i\},$
$\{x \to -2.78473 - 0.767326\,i, y \to -2.16878 + 0.716577\,i, z \to -1.0773 - 1.35012\,i\}\}$

In[111]:=
```
S6sy = sylvesterMD [SY, 6, {x, y, z}];
Dimensions [S6sy]
MatrixRank [S6sy]
```

Out[112]= {105, 84}

Out[113]= 65

So the nullity is 19 rather than the expected Bezout number 8. Try again

In[117]:=
```
S7sy = sylvesterMD [SY, 7, {x, y, z}];
Dimensions [S7sy]
MatrixRank [S7sy]
```

Out[118]= {168, 120}

Out[119]= 98

Now the nullity is 22 and will continue to increase by 3 as the order is increased. Essentially this tells us we have a curve of effective degree 3.

Now we can use Bezout's theorem to calculate how many complex projective intersection points this curve will have with a hypersurface, that is, surface defined by one equation, in $\mathbb{CP}^3$. We start with a plane

In[133]:=
```
plane1 = -3 - 3 x + y + z;
SYp = Append [SY, plane1]
```

Out[134]= $\{3. + 6.\,x + 3.\,x^2 - 4.\,y - 3.\,x\,y + 1.\,y^2 - 1.\,z - 1.\,x\,z,$
$-1.\,x - 1.\,x^2 - 1.\,z + 1.\,y\,z, 3.\,x + 3.\,x^2 - 1.\,x\,y - 3.\,x\,z + 1.\,z^2, -3 - 3\,x + y + z\}$

The sum of degrees is now 7.

In[135]:=
```
S7syp = sylvesterMD [SYp, 7, {x, y, z}];
Dimensions [S7syp]
MatrixRank [S7syp]
```

Out[136]= {252, 120}

Out[137]= 117

It should not be a surprise that the nullity is 3. So we expect 3 complex projective points

In[138]:=
```
asolsya = {x, y, z} /. NSolve[SYp]
solsya = infinitePointsMD [SYp, {x, y, z}, 1.*^-5]
```

Out[138]= {{−3., 0., −6.}, {0., 3., 0.}, {−1., 0., 0.}}

Out[139]= {}

So we have 3 affine points and no infinite points.

In[141]:=
```
n7sya = Transpose [Table[aVecMD [p, 7, {x, y, z}], {p, asolsya}]];
```

In[143]:=
```
Dimensions [n7sya]
```

Out[143]= {120, 3}

In[144]:=
```
SingularValueList [S7syp.n7sya, Tolerance → 0]
```

Out[144]= $\{9.70843 \times 10^{-11}, 0., 0.\}$

So n7sya is the approximate 3 dimensional nullspace of the Sylvester matrix S7syp illustrating Bezout's theorem for a 4×3 system. Now lets try a surface of degree 3. Now the sum of the degrees is 9.

In[151]:=
```
s3 = x^2 y + x y z + y z^2;
SYs = Append[SY, s3]
```

Out[152]= $\{3. + 6.\,x + 3.\,x^2 - 4.\,y - 3.\,x\,y + 1.\,y^2 - 1.\,z - 1.\,x\,z,$
$-1.\,x - 1.\,x^2 - 1.\,z + 1.\,y\,z,\ 3.\,x + 3.\,x^2 - 1.\,x\,y - 3.\,x\,z + 1.\,z^2,\ x^2\,y + x\,y\,z + y\,z^2\}$

In[153]:=
```
S9sys = sylvesterMD [SYs, 9, {x, y, z}];
Dimensions [S9sys]
MatrixRank [S9sys]
```

Out[154]= {444, 220}

Out[155]= 211

The nullity is 9. Solving

In[156]:=
```
solsys = {x, y, z} /. NSolve[SYs]
```

Out[156]= {{−3., 0., −6.}, {0., 3., 0.}, {0., 3., 0.}, {−0.333333 − 0.3849 *i*, 0.333333 − 0.3849 *i*, 0.5 − 0.096225 *i*},
{−0.333333 + 0.3849 *i*, 0.333333 + 0.3849 *i*, 0.5 + 0.096225 *i*},
{0., 1., 0.}, {0., 1., 0.}, {−1., 0., 0.}, {−1., 0., 0.}}

In[159]:=
```
infinitePointsMD [SYs, {x, y, z}, 1.*^-10]
```

Out[159]= {}

This returns 9 points as expected, all affine, but we note that 3 of them are listed as being multiplicity 2 points. For example

In[158]:=
```
multiplicity0MD [SYs, 3, {0, 3, 0}, {x, y, z}, 1.*^-10]
```

» hilbert Function {1, 1, 0, 0}

» Depth 1

Out[158]= 2

So we have only 6 distinct affine points.

In[165]:=
```
n9sys = Transpose [
        aVecMD [#, 9, {x, y, z}] & /@ {{-3, 0, -6}, {0, 3, 0}, solsys [[4]], solsys [[5]], {0, 1, 0}, {-1, 0, 0}}];
```

In[167]:=
```
MatrixRank [n9sys]
```

Out[167]= 6

In[166]:=
```
SingularValueList [S9sys.n9sys, Tolerance → 0]
```

Out[166]= $\{2.2641 \times 10^{-14}, 1.70962 \times 10^{-14}, 2.15257 \times 10^{-30}, 9.41709 \times 10^{-31}, 0., 0.\}$

In this case it only says that n9sys is contained in the 9 dimensional nullspace of S9sys. The difference is that the nullspace of S9sys is counting by multiplicity. With more work we could find the missing 3 nullspace vectors similar to the work in the 2 dimensional Bezout theorem at `https://www.barryhdayton.space/curvebook/BezoutsTheorem.pdf`

A slightly different example is the twisted cubic of section 2.0. Consider all three equations

In[171]:=
```
twcubic = {-y^2 + x z, -x^2 + y, -x y + z}
l = RandomReal [{-1, 1}, 4].{x, y, z, 1}
```

Out[171]= $\{-y^2 + x z, -x^2 + y, -x y + z\}$

Out[172]= $-0.58838 - 0.122878\ x - 0.854448\ y - 0.523189\ z$

In[173]:=
```
Stw7 = sylvesterMD [Append [twCubic , l], 7, {x, y, z}];
Dimensions [Stw7]
MatrixRank [Stw7]
```

Out[174]= {252, 120}

Out[175]= 117

So Bezout says that the twisted cubic meets this random hyperplane in 3 complex projective points. If we take only the last 2 equations and l the sum of the degrees is only 5

In[176]:=
```
Stw5 = sylvesterMD [{-x^2 + y, -x y + z, l}, 5, {x, y, z}];
Dimensions [Stw5]
MatrixRank [Stw5]
```

Out[177]= {75, 56}

Out[178]= 52

Now Bezout reports 4 complex projective solutions. But note as in section 2.0

```
In[179]:= NSolve[{-x^2 + y, -x y + z, l}]

Out[179]= {{x → 0.102527 + 0.775424 i, y → -0.59077 + 0.159003 i, z → -0.183865 - 0.441795 i},
  {x → 0.102527 - 0.775424 i, y → -0.59077 - 0.159003 i, z → -0.183865 + 0.441795 i},
  {x → -1.83821, y → 3.379, z → -6.2113}}
```

we get only 3 affine solutions. So Bezout is telling us that, assuming these three solutions are simple which is true, there must be an infinite solution. In 2.0 we had to find this solution, with Bezout we can simply imply the existence of that solution.

3 | Applications

The last few sections of this Space Curve volume cover some of my other recent work. These will get somewhat technical and are aimed at mathematically sophisticated readers.

One section will cover Quadratic Surface Intersection Curves. Another application looks at classical resolution of plane curve singularities. I avoided this topic in my plane curve book because plane curve singularities are not numerically stable, by blowing up to a space curve we can often get a numerically stable model of the singularity.

The possibly final chapter covers the constructions of certain classical unions of lines in $\mathbb{R}^3$ including the Schläfli double 6 construction given by Hilbert. In my 2008 paper and subsequent improvements I leaned heavily on Bertini. Here I will use Mathematica and the methods of these books.

Here is the first section.

3.1 Implicitization of Parametric curves

3.1.1 General theory of parametric curves

It is well known that curves parameterized by polynomial, or more generally, rational functions are algebraic curves, that is can be described by a system of algebraic equations. In the past I have treated these separately, however I recently discovered that the theories are the same up to FLT.

So suppose we start with a rational curve in $\mathbb{R}^n$.

$$Q[t] = \left\{ \frac{p_1[t]}{\Delta[t]}, \frac{p_2[t]}{\Delta[t]}, \ldots, \frac{p_n[t]}{\Delta[t]} \right\} \tag{1}$$

where the common denominator $\Delta[t] \neq 0$ and the p_i and Δ are univariate polynomials in t. With this approach I do not need to make assumptions on the degrees of the numerators relative to each other or the denominator. In particular if $p_{n+1}[t] = \Delta[t] = 1$ is the constant polynomial then we say $Q[t]$ is a *polynomial curve*. The degree of a polynomial or rational curve is the largest degree d of $p_1, \ldots, p_{n+1}$.

A polynomial will be called *stripped* if the constant term is 0, that is $p[t]$ is stripped if $p[0] = 0$. We *strip* a polynomial by dropping the constant term, we write $\tilde{p}[t]$ for the stripped polynomial $p[t]$. Here we treat rational functions a bit differently from polynomial functions since we can only strip $Q[t]$ in equation (1) if Q[t] is not constant as stripping the constant polynomial Δ[t] =1 gives $\tilde{\Delta}[t] = 0$. For this reason we will only talk of stripping polynomials, not rational functions.

Given a rational curve as in (1), including polynomials, assuming

$$p_i[t] = a_{i0} + a_{i1}\, t + \cdots + a_{id}\, t^d \text{ for } i = 1, \ldots, n+1$$

we get a *projective stripped coefficient matrix*

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{d} \\ a_{21} & a_{22} & \dots & a_{2\,d} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n+1\times 1} & a_{n+1\times 2} & \dots & a_{n+1\,d} \end{pmatrix} \tag{2}$$

For example for the polynomial curve $\left\{2+3\,t+4\,t^2,\ 5+6\,t+7\,t^2\right\}$ the projective stripped coefficient matrix, including the stripped denominator is

$$\begin{pmatrix} 3 & 4 \\ 6 & 7 \\ 0 & 0 \end{pmatrix}$$

While for the rational function $\left\{\frac{2+3\,t+4\,t^2,}{1+8\,t+9\,t^2},\ \frac{5+6\,t+7\,t^2}{1+8\,t+9\,t^2}\right\}$ we get

$$\begin{pmatrix} 3 & 4 \\ 6 & 7 \\ 8 & 9 \end{pmatrix}$$

From this we get the *projective augmented coefficient matrix* by adjoining a last column containing the constant terms. For the two examples above

$$\text{A1} = \begin{pmatrix} 3 & 4 & 2 \\ 6 & 7 & 5 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{A2} = \begin{pmatrix} 3 & 4 & 2 \\ 6 & 7 & 5 \\ 8 & 9 & 1 \end{pmatrix}$$

The key observation is

```
In[119]:= fltMD[{t, t^2}, {{3, 4, 2}, {6, 7, 5}, {0, 0, 1}}]
Out[119]= {2 + 3 t + 4 t^2, 5 + 6 t + 7 t^2}

In[121]:= fltMD[{t, t^2}, {{3, 4, 2}, {6, 7, 5}, {8, 9, 1}}]
Out[121]= {(2 + 3 t + 4 t^2)/(1 + 8 t + 9 t^2), (5 + 6 t + 7 t^2)/(1 + 8 t + 9 t^2)}
```

More generally we have the following *FLT Parametric Curve Theorem:*

If $Q[t]$ is a rational curve of degree d with projective augmented coefficient matrix A then

$$\text{Q[t]=fltMD}\left[\begin{pmatrix} t \\ t^2 \\ \vdots \\ t^d \end{pmatrix}, \text{A}\right] \tag{3}$$

In particular, every rational curve of degree d is the FLT image of the stripped polynomial curve $\left\{t,\ t^2,\ \ldots,\ t^d\right\}$ in $\mathbb{R}^d$.

Note that this theorem implies that $T_d = \left\{t,\ t^2,\ \ldots,\ t^d\right\}$ is a universal curve for rational and polynomial curves. I call this curve a *parabola* after Kepler because it has a single

infinite point {0, …, 0, 1, 0}. When d is even the curve is tangent to the infinite hyperplane like the plane parabola T_2. Thus every rational curve is a specialization and/or projection of this family of curves. Further, it is not necessary to study rational curves separately from polynomial curves.

3.1.2 Shen-Yaun Example

This example from 2010 shows the problem of finding a good implicitization.

We use the example of L.Shen and C. Yaun in $\mathbb{R}^3$. [L.Shen, C.Yaun, Implicitization using Univariate Resultants, J Sys Sci Complex (2010) 23, pp.804 - 814.]

In[203]:=
```
sy = {-2 t^2 + t^3, 1 - t - t^2 + t^3, 2 t - 3 t^2 + t^3};
```

Their method gives the system of 3 equations, not actually stated in their paper:

In[165]:=
```
SY = {-3 - 7 x - 5 x^2 - x^3 + 7 y + 9 x y + 3 x^2 y - 5 y^2 - 3 x y^2 + y^3,
      -x^2 - x^3 + 2 x z + 3 x^2 z - 3 x z^2 + z^3,
      -3 y + 4 y^2 - y^3 - 2 y z + 3 y^2 z + 6 z^2 - 3 y z^2 + z^3};
```

They point out that the point {−1, 1, 0} satisfies these equations but is not on the curve. In fact there are actually 5 isolated points, all real, satisfying this system which are not on the curve. It is somewhat difficult to find these isolated points but with n equations in n unknowns we can use the fact that a small perturbation of the system will have only isolated solutions, using `FindRoot` we can locate nearby solutions on the non-perturbation system. We can check to see if they are actually on the paramet - ric curve using the parametric curve theorem above.

We use the random perturbation below which finds all the isolated points, this was found by trial and error

In[164]:=
```
rr = {0.01306586198991111` , -0.09887929561077524` , -0.05150297032114362` };
```

In[169]:=
```
solrr = {x, y, z} /. NSolve [SY + rr, {x, y, z}, Reals ]
```

Out[169]=
```
{{-1.32717 , 0.848784 , -0.413263 }, {-0.180712 , 0.588029 , -0.388907 },
 {-0.435589 , 0.308678 , -0.329942 }, {-1.03345 , -0.0117412 , 0.0489152 }}
```

These 4 real solutions are close to actual solutions of SY

In[199]:=
```
rsol = {x, y, z} /. FindRoot [SY, Transpose [{{x, y, z}, #}]] & /@ solrr
```

Out[199]=
```
{{-1.08567 , 0.966531 , -0.383168 }, {-0.0514731 , 0.809015 , -0.384493 },
 {-0.585515 , 0.190521 , -0.264511 }, {-0.988233 , 0.000269103 , 0.0116319 }}
```

In[194]:=
```
root6 = {x, y, z} /. Chop [FindRoot [SY, Transpose [
          {{x, y, z}, {-0.28172479907074977` - 0.10554902609695169` i,
            1.2050352897534784` - 0.004529970239375305` i,
            -0.29474952022205597` + 0.0027247859065144867` i}}]]]
```

Out[194]=
```
{-0.684747 , 1.10801 , -0.301161 }
```

In addition to these 4 real solutions of SY there is the multiplicity 2 solution {−1, 1, 0}

given by Shen-Yaun. Further we find one additional real solution starting from a complex solution of the perturbed system. Checking multiplicity

```
In[200]:= rsol = Join[rsol, {{-1, 1, 0}, root6}];
          Table[multiplicityMD [SY, s, {x, y, z}, dTol], {s, rsol}]
Out[201]= {1, 1, 1, 10, 2, 1}
```

The multiplicity of the fourth real solution is 10 because that is the default maximum multiplicity returned by multiplicityMD, this suggests that that point is non-isolated and thus on the parametric curve, while the others are not on the parametric curve. We can check this 4th point using our parametric curve theorem. The stripped curve is

```
sỹ = {-2 t^2 + t^3, -t - t^2 + t^3, 2 t - 3 t^2 + t^3};
```

the augmented projective stripped coefficient matrix is

```
In[209]:= symat = {{0, -2, 1, 0}, {-1, -1, 1, 1}, {2, -3, 1, 0}, {0, 0, 0, 1}};
```

giving the parametric equation as

```
In[210]:= sy = fltMD[{t, t^2, t^3}, symat]
Out[210]= {-2 t^2 + t^3, 1 - t - t^2 + t^3, 2 t - 3 t^2 + t^3}
```

We see that symat is an invertible matrix so the FLT given by this is also invertible. Thus the point rsol[[4]] comes from

```
In[211]:= q = fltMD[rsol[[4]], Inverse[symat]]
Out[211]= {0.988366 , 0.976868 , 0.965504 }
```

But note that this is on the curve $\{t, t^2, t^3\}$

```
In[212]:= {q[[1]], q[[1]]^2, q[[1]]^3}
Out[212]= {0.988366 , 0.976868 , 0.965504 }
```

So

```
In[215]:= rsol[[4]]
          fltMD[{q[[1]], q[[1]]^2, q[[1]]^3}, symat]
Out[215]= {-0.988233 , 0.000269103 , 0.0116319 }

Out[216]= {-0.988233 , 0.000269103 , 0.0116319 }
```

is on the curve sy. Thus the 5 isolated points of SY not on the curve sy are

In[213]:=
```
Drop[rsol, {4}]
```

Out[213]=
```
{{-1.08567, 0.966531, -0.383168},
 {-0.0514731, 0.809015, -0.384493}, {-0.585515, 0.190521, -0.264511},
 {-1, 1, 0}, {-0.684747, 1.10801, -0.301161}}
```

An important observation from this example is that, unlike for plane curves, none of these isolated points are singular because isolated points are the default case for 3×3 systems. This is what makes them hard to find.

In the next subsections we will show how to find a system for this last curve that does not have isolated points not on the curve.

3.1.3 Direct approach

The direct approach to implicitization for polynomial parameters has two parts, first we find all polynomials vanishing on the parametric curve up to a specified degree, then we find a H - basis of this ideal. We should check this as above to make sure that there are no points in this ideal that are not on the curve, but remember complex values of t are valid in this setting.

Use the indirect approach for rational parameters.

The user will need to decide the maximum degrees of the polynomials to be found. Often the correct degree is less than the maximum degree of a component of F, but apparently never larger. Using the maximum degree of a component the second step will give the lower correct degree so this is a safe, but maybe not the quickest choice. In the next subsection we will give a family of curves of arbitrarily large degree and dimension with impliciza - tion consisting of quadratic polynomials.

The following function takes as arguments a polynomial parametric curve F, a specified degree d the parameter t and the variables you wish to use on the target space. The number of variables should match the number of components of F. This routine is similar to the routine in section A.5 of the plane curve book but better even for 2 variables. This routine expects exact or at least very accurate numerical coefficients of F otherwise you may need to replace the built in NullSpace finder with an numerical one based on the SVD.

```
In[95]:= p2aRawMD[F_, d_, t_, X_] := Module[{n, TB, cr, ar, SA, NSA, FA},
           n = Length[X];
           If[Length[F] ≠ n, Echo["Dimension mismatch F,X"]; Abort[]];
           TB = Expand[Table[m /. Thread[X → F], {m, mExpsMD[d, X]}]];
           cr = <|CoefficientRules[#, {t}]|> &/@ TB;
           ar = Append[
              Flatten[Table[Table[{i, First[k] + 1} → cr[[i]][k], {k, Keys[cr[[i]]]}], {i, Length[cr]}], 1],
              {_, _} → 0];
           SA = SparseArray[ar];
           NSA = NullSpace[Transpose[SA]];
           If[Length[NSA] < n - 1, Echo["Fail, Try higher d"]; Abort[]];
           FA = NSA.mExpsMD[d, X];
           Echo[Table[Expand[FA[[j]] /. Thread[X → F]], {j, Length[FA]}], "Residues "];
           FA]
```

We will illustrate with the Shen-Yaun example above

```
In[96]:= sy = {-2 t^2 + t^3, 1 - t - t^2 + t^3, 2 t - 3 t^2 + t^3};

In[99]:= G = p2aRawMD [sy, 3, t, {x, y, z}]
```

» Residues {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

$$\text{Out[99]}= \{8x^2+8x^3-3x^2y+2xz-6x^2z+z^3,\ 3x+3x^2-xy-4xz-x^2z+yz^2,$$
$$-x-x^2-xy-x^2y-z+y^2z,\ 12+32x+28x^2+8x^3-13y-18xy-6x^2y+y^3-5z-8xz-3x^2z,$$
$$3x^2+3x^3-x^2y-3x^2z+xz^2,\ -x^2-x^3-xz+xyz,\ 3x+6x^2+3x^3-4xy-3x^2y+xy^2-xz-x^2z,$$
$$3x+3x^2-xy-3xz+z^2,\ -x-x^2-z+yz,\ 3+6x+3x^2-4y-3xy+y^2-z-xz\}$$

Note that 6 polynomials are returned. Now we find a H-basis

```
In[100]:= H = hBasisMD [G, 3, {x, y, z}, dTol]
```

» Hilbert Function {1, 3, 3, 3}

$$\text{Out[100]}= \{3.+6.x+3.x^2-4.y-3.xy+1.y^2-1.z-1.xz,\ -1.x-1.x^2-1.z+1.yz,\ 3.x+3.x^2-1.xy-3.xz+1.z^2\}$$

Note that 3 equations are returned. One needs to check that unlike the Shen-Yaun system, this has no isolated or other solutions not on the curve. We only check their point here

```
In[101]:= H /. Thread [{x, y, z} → {-1, 1, 0}]
```

$$\text{Out[101]}= \{4.44089\times10^{-16},\ -1.77636\times10^{-15},\ 1.\}$$

It does satisfy the first two equations but not the third.

3.1.4 The indirect approach.

The FLT Parametric Curve Theorem says every polynomial or rational parametric curve F is the image of the famous *rational normal curve* $\mathrm{T}_d = \{t, t^2, \ldots, t^d\}$ where d is the maximum degree of a polynomial in t in the numerator or denominator of F. So we use `FLTMD` on the FLT from the

theorem using a known implicitation of T_d. We have the

Theorem:[see Joe Harris' book] *The implicitization of T_d is given by quadratic binomials in {x1,…,xd}, in particular the $\binom{d}{2}$ monomials given by* `p2rawMD [{t, t^2, …, t^d}, 2, t, {x1, …, xd}]`

We will not prove this here but it is easy to check any case by the direct method in the last section, for example n=4

```
In[120]:= raw4 = p2aRawMD [{t, t^2, t^3, t^4}, 4, t, {x1, x2, x3, x4}]
» Residues {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
Out[120]= {x3^2 x4^2 - x2 x4^3, x3^3 x4 - x1 x4^3, x3^4 - x4^3, x2 x3 x4^2 - x1 x4^3, x2 x3^2 x4 - x4^3, x2 x3^3 - x3 x4^2, x2^2 x4^2 - x4^3,
    x2^2 x3 x4 - x3 x4^2, x2^2 x3^2 - x2 x4^2, x2^3 x4 - x2 x4^2, x2^3 x3 - x1 x4^2, x2^4 - x4^2, x1 x3 x4^2 - x4^3,
    x1 x3^2 x4 - x3 x4^2, x1 x3^3 - x2 x4^2, x1 x2 x4^2 - x3 x4^2, x1 x2 x3 x4 - x2 x4^2, x1 x2 x3^2 - x1 x4^2,
    x1 x2^2 x4 - x1 x4^2, x1 x2^2 x3 - x4^2, x1 x2^3 - x3 x4, x1^2 x4^2 - x2 x4^2, x1^2 x3 x4 - x1 x4^2, x1^2 x3^2 - x4^2,
    x1^2 x2 x4 - x4^2, x1^2 x2 x3 - x3 x4, x1^2 x2^2 - x2 x4, x1^3 x4 - x3 x4, x1^3 x3 - x2 x4, x1^3 x2 - x1 x4, x1^4 - x4,
    x3^2 x4 - x2 x4^2, x3^3 - x1 x4^2, x2 x3 x4 - x1 x4^2, x2 x3^2 - x4^2, x2^2 x4 - x4^2, x2^2 x3 - x3 x4, x2^3 - x2 x4,
    x1 x3 x4 - x4^2, x1 x3^2 - x3 x4, x1 x2 x4 - x3 x4, x1 x2 x3 - x2 x4, x1 x2^2 - x1 x4, x1^2 x4 - x2 x4,
    x1^2 x3 - x1 x4, x1^2 x2 - x4, x1^3 - x3, x3^2 - x2 x4, x2 x3 - x1 x4, x2^2 - x4, x1 x3 - x4, x1 x2 - x3, x1^2 - x2}

In[121]:= tBasis4 = hBasisMD [raw4, 4, {x1, x2, x3, x4}, dTol]
» Hilbert Function {1, 4, 4, 4, 4}
Out[121]= {1. x1^2 - 1. x2, 1. x1 x2 - 1. x3, 1. x1 x3 - 1. x4, 1. x2^2 - 1. x4, 1. x2 x3 - 1. x1 x4, 1. x3^2 - 1. x2 x4}

In[150]:= raw2 = p2aRawMD [{t, t^2, t^3, t^4}, 2, t, {x1, x2, x3, x4}]
» Residues {0, 0, 0, 0, 0, 0}
Out[150]= {x3^2 - x2 x4, x2 x3 - x1 x4, x2^2 - x4, x1 x3 - x4, x1 x2 - x3, x1^2 - x2}
```

One might think from the theorem that one could build tBasis up recursively by merely adding $d-1$ binomials to the previous tBasis. This is not true however, but the new terms do imply the old terms are also in the ideal generated by the larger basis. For example

```
In[114]:= tBasis3 = p2aRawMD [{t, t^2, t^3}, 2, t, {x1, x2, x3}]
» Residues {0, 0, 0}
Out[114]= {x2^2 - x1 x3, x1 x2 - x3, x1^2 - x2}
```

Here x2^2 − x1 x3 has been replaced by x2^2 − x4 and x1 x3 − x4 which imply the former.

For further use we will collect the first few cases of tBasis, they should be initialized in GlobalFunctionsMD

In[210]:=
```
tBasis2 = {x1 ^ 2 – x2};
tBasis3 = {x2 ^ 2 – x1 x3, x1 x2 – x3, x1 ^ 2 – x2};
tBasis4 = {x3 ^ 2 – x2 x4, x2 x3 – x1 x4, x2 ^ 2 – x4, x1 x3 – x4, x1 x2 – x3, x1 ^ 2 – x2};
tBasis5 = {x4^2 – x3 x5, x3 x4 – x2 x5, x3^2 – x1 x5, x2 x4 – x1 x5,
      x2 x3 – x5, x2^2 – x4, x1 x4 – x5, x1 x3 – x4, x1 x2 – x3, x1^2 – x2};
```

We can redo the Shaun-Yaun example by using the FLT from section 3.1.2

In[174]:=
```
symat = {{0, –2, 1, 0}, {–1, –1, 1, 1}, {2, –3, 1, 0}, {0, 0, 0, 1}};
sy = fltMD[{t, t ^ 2, t ^ 3}, symat]
```

Out[175]= $\{-2\,t^2+t^3,\ 1-t-t^2+t^3,\ 2\,t-3\,t^2+t^3\}$

In[176]:=
```
H2 = FLTMD[tBasis3, symat, 3, {x1, x2, x3}, {x, y, z}, dTol]
```

» Hilbert Function {1, 3, 3, 3}

Out[176]= $\{3.\,x+3.\,x^2-1.\,x\,y-3.\,x\,z+1.\,z^2,\ 1.\,x+1.\,x^2+1.\,z-1.\,y\,z,$
$1.+2.33333\,x+1.33333\,x^2-1.33333\,y-1.\,x\,y+0.333333\,y^2-0.333333\,x\,z-0.333333\,y\,z\}$

Note that this is different from the implicitization we got using the direct approach above because FLTMD works projectively and applies `hBasisMD` on a homogeneous system where our direct method works completely in the affine situation. But we can see these are the same by applying hBasisMD to the result. The fact that the Hilbert function is unchanged implies these systems are equivalent.

In[165]:=
```
hBasisMD [H2, 3, {x, y, z}, dTol]
```

» Hilbert Function {1, 3, 3, 3}

Out[165]= $\{3.+6.\,x+3.\,x^2-4.\,y-3.\,x\,y+1.\,y^2-1.\,z-1.\,x\,z,\ -1.\,x-1.\,x^2-1.\,z+1.\,y\,z,\ 3.\,x+3.\,x^2-1.\,x\,y-3.\,x\,z+1.\,z^2\}$

As a second example we look at a rational parameterization of the piriform.

In[118]:=
```
piriformpar = {(1 – t^4)/(1 + 2 t ^ 2 + t ^ 4), (4 t)/(1 + 2 t ^ 2 + t ^ 4)}
```

Out[118]= $\left\{\frac{1-t^4}{1+2\,t^2+t^4},\ \frac{4\,t}{1+2\,t^2+t^4}\right\}$

We can construct a 3×5 FLT matrix by labeling the columns by $t,\ t^2,\ t^3,\ t^4,\ 1$ and rows by coefficients of $1-t^4,\ 4\,t,\ 1+2\,t^2+t^4$ respectively.

In[206]:=
```
piriformA = {{0, 0, 0, –1, 1}, {4, 0, 0, 0, 0}, {0, 2, 0, 1, 1}};
piriformA // MatrixForm
```

Out[207]//MatrixForm=
$$\begin{pmatrix} 0 & 0 & 0 & -1 & 1 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 1 & 1 \end{pmatrix}$$

Checking

In[177]:= fltMD[{t, t^2, t^3, t^4}, piriformA]

Out[177]= $\left\{\frac{1-t^4}{1+2t^2+t^4}, \frac{4t}{1+2t^2+t^4}\right\}$

Thus an implicitization of the piriform is

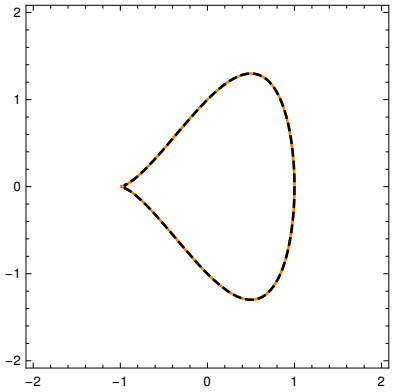
In[214]:= piriformEq = FLTMD[{x3^2 - x2 x4, x2 x3 - x1 x4, x2^2 - x4, x1 x3 - x4, x1 x2 - x3, x1^2 - x2}, piriformA, 4, {x1, x2, x3, x4}, {x, y}, dTol][[1]]

» Hilbert Function {1, 2, 3, 4, 4}

Out[214]= $1. + 2.x - 2.x^3 - 1.x^4 - 1.y^2$

In[220]:= Show[ContourPlot[piriformEq == 0, {x, -2, 2}, {y, -2, 2}, ContourStyle → Orange], ParametricPlot[piriformpar, {t, -6, 6}, PlotStyle → Directive[Dashed, Black], ImageSize → Small]]

Out[220]=



A more complicated example is

In[258]:= fpar = {(t+2)/(t^2+1), (t^2-1)/(t^2+1), (t^2-t+1)/(t^2+1), (4t^2)/(t^2+1)};

Again this can be actualized by an FLT with matrix

In[259]:= fparA = {{1, 0, 2}, {0, 1, -1}, {-1, 1, 1}, {0, 4, 0}, {0, 1, 1}};
fltMD[{t, t^2}, fparA]

Out[260]= $\left\{\frac{2+t}{1+t^2}, \frac{-1+t^2}{1+t^2}, \frac{1-t+t^2}{1+t^2}, \frac{4t^2}{1+t^2}\right\}$

So the implicit curve in $\mathbb{R}^4$ is

In[261]:= fparEq = FLTMD[tBasis2, fparA, 2, {x1, x2}, {x, y, z, w}, dTol]

» Hilbert Function {1, 2, 2}

Out[261]= $\{1.w - 1.x - 3.y - 1.z, 1. - 0.5x - 0.5y - 0.5z, 1.x^2 + 2.xy + 2.33333y^2 - 3.33333xz - 3.33333yz + 1.z^2\}$

At first we might be surprised that of the 3 equations two are linear which means this curve lies in a 2 dimensional subset of $\mathbb{R}^4$. But on further consid -

eration we see that this curve is contained in the image of a FLT defined on $\mathbb{R}^2$ which itself cannot have image greater than 2. Applying a somewhat random orthogonal FLT projection with matrix

```
In[264]:= projA = {{0.7071067811865475`, 0.`, 0.`, 0.7071067811865475`, 0.`},
            {0.4082482904638631`, 0.816496580927726`, 0.`, -0.4082482904638631`, 0.`},
            {0.`, 0.`, 0.`, 0.`, 1.`}}
```

```
Out[264]= {{0.707107, 0., 0., 0.707107, 0.}, {0.408248, 0.816497, 0., -0.408248, 0.}, {0., 0., 0., 0., 1.}}
```

we find that the parametric curve projects to

```
In[265]:= fparproj = N[fltMD[fpar, projA]]
```

$$\text{Out[265]}= \left\{\frac{2.82843\,t^2}{1.+t^2}+\frac{0.707107\,(2.+t)}{1.+t^2},\ -\frac{1.63299\,t^2}{1.+t^2}+\frac{0.408248\,(2.+t)}{1.+t^2}+\frac{0.816497\,(-1.+t^2)}{1.+t^2}\right\}$$

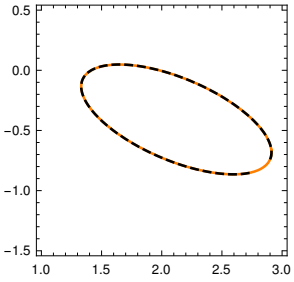
while the curve in $\mathbb{R}^4$projects to

```
In[267]:= fprojEq = FLTMD[fparEq, projA, 2, {x, y, z, w}, {x, y}, dTol][[1]]
» Hilbert Function {1, 2, 2}
```

$$\text{Out[267]}= 1.-1.21218\,x+0.357143\,x^2-0.699854\,y+0.742307\,x\,y+1.07143\,y^2$$

```
In[268]:= Show[ContourPlot[fprojEq == 0, {x, 1, 3}, {y, -1.5, .5}, ContourStyle → Orange],
            ParametricPlot[fparproj, {t, -8, 8}, PlotStyle → Directive[Black, Dashed]],
            ImageSize → Small]
```

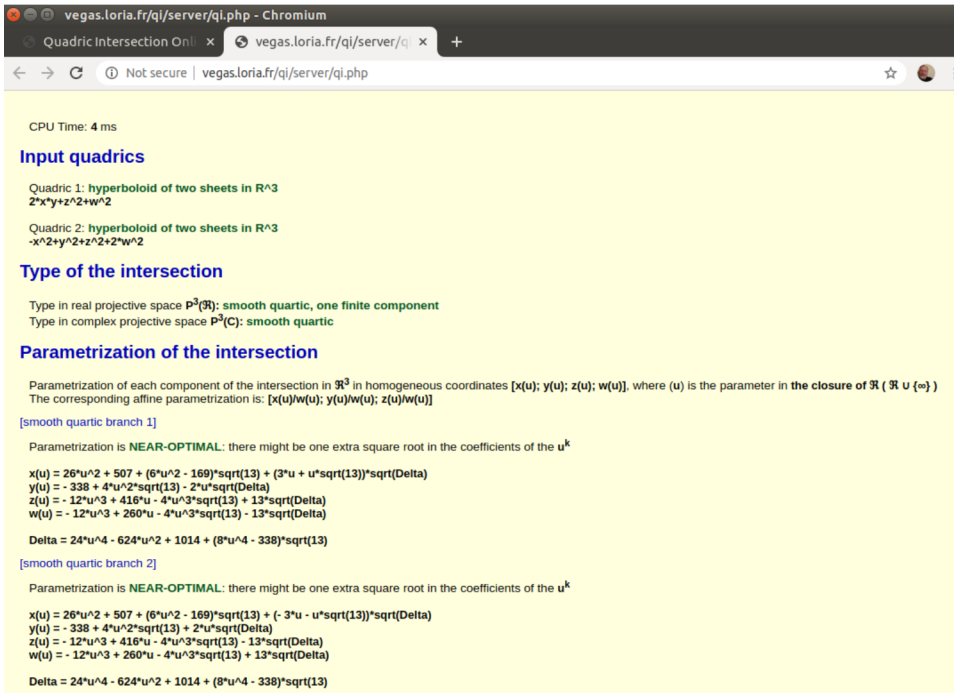
Out[268]=



So we merely have a plane ellipse lying in $\mathbb{R}^4$.

3.2 Quadratic Surface Intersection Curves (QSIC)

This is a classical area that only recently has seen a full solution. C.Tu, W.Wang, B. Mourrain and J. Wang, [TWMW], have given in the journal *Computer aided Geometric Design 2009* a complete classification of QSIC identifying 35 types including singular QSIC. L. Dupont, D. Lazard, S. Lazard and S. Petitjean [DLLP] presented a 65 page discussion and working black box algorithm in 2008 available on http://vegas.loria.fr/qi/index.html , a typical run looks like this



Here I give my take on this subject.

3.2 .1 The Theory

A *quadratic surface intersection curve* (QSIC) is a naive curve where the 2 equations are quadratic (degree 2) in three variables. It helps, however, to have the full general theory in understanding these curves.

In principle these curves have degree 4, that is, a generic plane projection will be a curve of degree 4. Alternatively a generic plane intersects a generic plane in 4 complex projective points. Using our Bezout theorem

```
In[123]:= X = mExpsMD[2, {x, y, z}];
          F1 = RandomInteger[{-9, 9}, {2, 10}].X
          plane1 = RandomInteger[{-9, 9}, 4].{x, y, z, 1}
          S7 = sylvesterMD[Append[F1, plane], 7, {x, y, z}];
          dim = Dimensions[S7];
          rnk = MatrixRank[S7];
          dim[[2]] - rnk
```

$$\text{Out[124]=}\ \{-2+7x+2x^2-6y-4xy+2z+8xz-6yz+2z^2,\ -2+4x-7x^2+8y-4xy-3y^2+9z-7xz+3yz-z^2\}$$

$$\text{Out[125]=}\ 4+2x-3y-6z$$

$$\text{Out[129]=}\ 4$$

For a non-singular QSIC classical mathematicians have determined this is a curve of genus 1. Plane curves of genus 1 include the elliptic curves

$y^2 - x^3 - a\,x - b$ and hyper-elliptic curves $y^2 - x^4 - a\,x^2 - b\,x - c$ where in both cases the cubic in x has no multiple zeros. As the screen image above shows DLLP can parameterize these curves in the form of rational functions of the form

$$\rho[u] = \{(U_1[u] + V_1[u]\,\mathbf{Sqrt}[\delta[u]])/\Delta,\ (U_2[u] + V_2[u]\,\mathbf{Sqrt}[\delta])/\Delta,\ (U_3[u] + V_3[u]\,\mathbf{Sqrt}[\delta[u]])/\Delta\}$$
$$\Delta[u] = U_4[u] + V_4[u]\,\mathbf{Sqrt}[\delta[u]] \qquad (4)$$

where U_i, V_i, δ are polynomials of degree 4 in u, and Δ, δ are the same for all three coordinates.

In my 2011 paper on QSIC I show that one can do better in that the U_i, V_i, δ can be polynomials of degree 3. Here is an exposition in terms of the Wolfram Language.

Here Q is the equation of our QSIC and p is a point on Q. We obtain an FLT projection Ω and right inverse $\mho$ which is not an FLT. In addition we obtain a cubic plane curve h which is the domain of $\mho$. The algorithm takes p to an infinite point so is not in the domain of Ω.

Suppose we take a random example, say the one above

```
In[113]:= Qr = {-2+7x+2x^2-6y-4xy+2z+8xz-6yz+2z^2,
              -2+4x-7x^2+8y-4xy-3y^2+9z-7xz+3yz-z^2};
```

We first obtain a point on the curve, in general this might not be random.

```
In[116]:= cp = criticalPoints3D[Q, {x, y, z}][[2]]
Out[116]= {0.199762, 0.0222691, 0.176374}
```

Next we use the following function with codifies the method in my 2011 paper. This returns a plane polynomial h of degree 3, an FLT Ω which takes the curve Q to h and a function $\mho$ which maps h back up to Q as a right inverse, that is $\mho[\Omega[q]] = q$ for almost all q in Q. One needs to be careful with the usage since the routine does use randomization and will give a different result each run. This randomization turns out to be essential since most integer coefficient examples one might use, eg. from [TWMW], are not full, that is the input polynomials must have non-zero coefficients for each monomial, for the classical trick we use to work. Also to avoid messy output I recommend running quietly with ";" .

In[108]:=
```
nsQSIC3D[Q_, p_, {x_, y_, z_}] := Module[{p0, A, F, h, L, M, R, S, Ω, ʊ},
    p0 = Normalize[Append[p, 1]];
    A = Reverse[Orthogonalize[Prepend[RandomReal[{-1, 1}, {3, 4}], p0]]];
    F = FLT3D[Q, A, {x, y, z}];
    L = formMD[F[[1]], 1, {x, y, z}];
    M = formMD[F[[2]], 1, {x, y, z}];
    R = formMD[F[[1]], 2, {x, y, z}];
    S = formMD[F[[2]], 2, {x, y, z}];
    h = Expand[L * S - R * M] /. {z → 1};
    Ω = Take[fltMD[#, A], 2] / (fltMD[#, A][[3]]) &;
    ʊ = Take[Inverse[A].Join[#, {1, (-R / L) /. Thread[{x, y, z} → Append[#, 1]]}], 3] /
          Last[Inverse[A].Join[#, {1, (-R / L) /. Thread[{x, y, z} → Append[#, 1]]}]] &;
    {h,
      Ω,
      ʊ}]
```

In[117]:= `{hr, Ωr, ʊr} = nsQSIC3D[Q, cp, {x, y, z}];` (* non evaluative cell for illustration only*)

Now we carefully look at the output. First we note that we do get a cubic polynomial for h. Note this will be numerical and full for the integer sparse input.

In[118]:= `hr` (* non evaluative cell *)

Out[118]=
$$47.2734 - 126.297\,x + 137.892\,x^2 - 2.33154\,x^3 + 38.6005\,y - 7.96867\,x\,y - 4.20928\,x^2\,y + 33.8287\,y^2 + 5.10199\,x\,y^2 - 5.79393\,y^3$$

Rather than look at the complicated definition of Ω, $\mho$ we evaluate the output functions using variables for values. We see that we do get an FLT.

In[121]:= `Ωr[{x, y, z}]` (* non evaluative cell *)

Out[121]=
$$\left\{\frac{-0.192492 + 0.511139\,x + 0.724164\,y + 0.421034\,z}{0.167346 - 0.654154\,x + 0.676769\,y - 0.293362\,z},\ \frac{0.0409707 + 0.523046\,x + 0.130794\,y - 0.841212\,z}{0.167346 - 0.654154\,x + 0.676769\,y - 0.293362\,z}\right\}$$

$\mho$ takes points on the plane to points in $\mathbb{R}^3$, it is easier to work with each coordinate separately.

In[265]:=
```
∇rx = Simplify[∇r[{x, y}][[1]]]   (* non evaluative cell *)
∇ry = Simplify[∇r[{x, y}][[2]]]
∇rz = Simplify[∇r[{x, y}][[3]]]
```

Out[265]=
$$\frac{6.77097 + 0.677412\,x^2 + x\,(-5.94613 + 0.730631\,y) - 6.95909\,y + 0.262376\,y^2}{-5.37988 + 5.23728\,x + 0.505216\,x^2 - 5.00917\,y + 0.619406\,x\,y + 1.\,y^2}$$

Out[266]=
$$\frac{7.79955 + 7.48938\,x - 0.775353\,x^2 + 1.51171\,y - 0.238621\,x\,y - 0.0380644\,y^2}{5.37988 - 5.23728\,x - 0.505216\,x^2 + 5.00917\,y - 0.619406\,x\,y - 1.\,y^2}$$

Out[267]=
$$\frac{2.73532 + 0.56636\,x^2 + x\,(-4.60656 - 0.725381\,y) + 8.75834\,y + 0.0731931\,y^2}{-5.37988 + 5.23728\,x + 0.505216\,x^2 - 5.00917\,y + 0.619406\,x\,y + 1.\,y^2}$$

Important Note: If you execute this code then even if you use the same input these values will change. To continue with this particular example between sessions we initialize now as follows:

In[252]:=
```
Qr = {-2 + 7 x + 2 x^2 - 6 y - 4 x y + 2 z + 8 x z - 6 y z + 2 z^2,
      -2 + 4 x - 7 x^2 + 8 y - 4 x y - 3 y^2 + 9 z - 7 x z + 3 y z - z^2};
hr = 47.27339766682051` - 126.29676742395714` x + 137.8924583577859` x^2 -
      2.3315379753889216` x^3 + 38.600477189804465` y -
      7.968669330520427` x y - 4.209276523596027` x^2 y +
      33.82865025275894` y^2 + 5.101992253822809` x y^2 - 5.793933359433088` y^3;
Ωr[x_, y_, z_] := {(-0.19249242259065155` + 0.511138659376466` x +
          0.7241643930306724` y + 0.4210342860178124` z) /
        (0.1673459529911122` - 0.6541543196115073` x +
          0.6767688687679541` y - 0.29336215914402825` z),
      (0.04097065367206614` + 0.5230464061760043` x + 0.13079378255133922` y -
          0.8412115363985226` z) / (0.1673459529911122` - 0.6541543196115073` x +
          0.6767688687679541` y - 0.29336215914402825` z)};
Ʊrx[{x_, y_}] := (6.770973793600855` - 5.946132153292747` x +
          0.6774118751211937` x^2 + (-6.959092852928387` + 0.7306313695805137` x) y +
          0.2623764415412226` y^2) /
      (-5.379876935689428` + 5.2372843516982615` x + 0.5052157400292752` x^2 -
          5.009169689826631` y + 0.6194057061472128` x y + 1.` y^2);
Ʊry[{x_, y_}] := (7.799549554433994` + 7.489375420102824` x -
          0.7753526722058538` x^2 + 1.511709673478418` y -
          0.23862093428751913` x y - 0.038064407750122875` y^2) /
      (5.379876935689428` - 5.2372843516982615` x - 0.5052157400292752` x^2 +
          5.009169689826631` y - 0.6194057061472128` x y - 1.` y^2);
Ʊrz[{x_, y_}] := (2.7353213149626185` - 4.606560493457439` x +
          0.5663595351370571` x^2 + (8.758337796121568` - 0.725381245932385` x) y +
          0.07319306835158754` y^2) /
      (-5.379876935689428` + 5.2372843516982615` x + 0.5052157400292752` x^2 -
          5.009169689826631` y + 0.6194057061472128` x y + 1.` y^2)
Ʊr[{x_, y_}] := {Ʊrx[{x, y}], Ʊry[{x, y}], Ʊrz[{x, y}]};
```

We observe that each is a fraction of two quadratics in t, s with a common denominator we will call Δ.In practice we will parameterize the cubic h by putting it in Weierstrass normal form as in Chapter 7 of my plane curve book $y\text{^}2 = x\text{^}3 + a\,x + b$. There is a 2 dimensional FLT taking We write $\delta = x\text{^}3 + a\,x + b$ so we can parameterize this latter curve by $\{t, \pm\text{Sqrt}[\delta[t]]\}$. There is a 2-dimensional flt taking h to this Weierstrass curve so h is parameterized by $t \longrightarrow$ **flt2D**[{t, ±Sqrt[δ[t]]},Inverse[Ah]] for a 3×3 invertible matrix Ah obtained as part of the reduction of h to Weier - strass form.

In[119]:= **allInflectionPoints2D[hr, x, y]**

Out[119]= {{39.5436, 14.2075}, {2.82384, 9.43578}, {−3.64555, 8.59508}}

In[121]:= **inflecPt = {2.8238358825981082`, 9.43577590447643`}**

Out[121]= {2.82384, 9.43578}

In[122]:= **{w, Aw} = weierstrassNormalForm2D[hr, inflecPt, x, y]**

Out[122]= $\{-4.07613 - 4.68308\,x + 1.\,x^3 - 1.\,y^2, \{\{0.60151, -0.00139735, -1.68538\},$
$\{0.516591, 0.875264, 0.182328\}, \{0.17991, -0.15676, 0.971112\}\}\}$

In[126]:= **ω = w /. {x → t, y → 0}**

Out[126]= $-4.07613 - 4.68308\,t + 1.\,t^3$

In[123]:= **Clear[s, t]**

Our transformation from the curve $s^2 = \omega[t]$ is given by

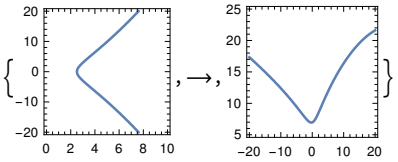
In[129]:= **{x, y} = TransformationFunction[Inverse[Aw]][{t, s}]**

Out[129]= $\left\{\frac{1.58421 + 0.285239\,s + 0.943677\,t}{0.566277 + 0.101011\,s - 0.256123\,t}, \frac{-1.05298 + 0.953119\,s - 0.503615\,t}{0.566277 + 0.101011\,s - 0.256123\,t}\right\}$

In pictures

In[135]:= **{ContourPlot[s^2 == ω, {t, 0, 10}, {s, −20, 20}, ImageSize → Tiny], "⟶", ContourPlot[hr == 0, {x, −20, 20}, {y, 5, 25}, ImageSize → Tiny]}**

Out[135]=



Now we note that composing the transformation function with $\mathfrak{U}\mathrm{rx}$ gives

In[124]:= **ux = Simplify[𝔘rx[TransformationFunction[Inverse[Aw]][{t, s}]]]**

Out[124]= $\frac{2.40371 - 4.35144\,s - 0.382852\,s^2 - 2.22272\,t + 3.05474\,s\,t + 1.78529\,t^2}{9.98189 - 2.60174\,s + 1.\,s^2 + 5.15708\,t + 2.25881\,s\,t - 2.53622\,t^2}$

which is again a quadratic rational expression, likewise for y,z. Now the upper and lower half of $s^2 = \omega[t]$ can be parameterized by $s = \pm\mathbf{Sqrt}[\omega]$. We have the following special function to replace s by the right hand side and simplify:

In[113]:=
```
specialExpand[w_, u_, s_, sgn_] := Module[{w1},
    w1 = Expand[w /. {s^2 → u}];
    Collect[w1, s] /. {s → sgn * Sqrt[u]}]
```

In[178]:=
```
μx := specialExpand[Numerator[ux], ω, s, 1] /. {t → #} &
```

Likewise

In[211]:=
```
uy = Simplify[Ury[TransformationFunction[Inverse[Aw]][{t, s}]]];
uz = Simplify[Urz[TransformationFunction[Inverse[Aw]][{t, s}]]];
μy = specialExpand[Numerator[uy], ω, s, 1] /. {t → #} &;
μz = specialExpand[Numerator[uz], ω, s, 1] /. {t → #} &;
Δ = specialExpand[Denominator[uy], ω, s, 1] /. {t → #} &;
```

So we have our local parameterization of Q as described above

In[216]:=
```
μ[t_] := {μx[t] / Δ[t], μy[t] / Δ[t], μz[t] / Δ[t]}
```

where

In[136]:=
```
μx[t]
μy[t]
μz[t]
Δ[t]
```

Out[136]=
$$3.96427 - 0.429792\,t + 1.78529\,t^2 - 0.382852\,t^3 + (-4.35144 + 3.05474\,t)\,\sqrt{-4.07613 - 4.68308\,t + 1.\,t^3}$$

Out[137]=
$$-7.64494 + 6.22854\,t + 2.31009\,t^2 - 0.380456\,t^3 + (-4.1584 + 1.70014\,t)\,\sqrt{-4.07613 - 4.68308\,t + 1.\,t^3}$$

Out[138]=
$$-11.5215 - 2.06748\,t + 4.49677\,t^2 + 0.893507\,t^3 + (2.89031 - 4.29325\,t)\,\sqrt{-4.07613 - 4.68308\,t + 1.\,t^3}$$

Out[139]=
$$5.90575 + 0.474002\,t - 2.53622\,t^2 + 1.\,t^3 + (-2.60174 + 2.25881\,t)\,\sqrt{-4.07613 - 4.68308\,t + 1.\,t^3}$$

Before we use these we need to find the domains, we need $\omega \geq 0$ and $\Delta[t] \neq 0$.

In[237]:=
```
Reduce[ω > 0]
a = t /. NSolve[ω][[3]]
NSolve[Δ[t]]
```

--- Reduce : Reduce was unable to solve the system with inexact coefficients . The answer was obtained by solving a corresponding exact system and numericizing the result .

Out[237]= t > 2.51122

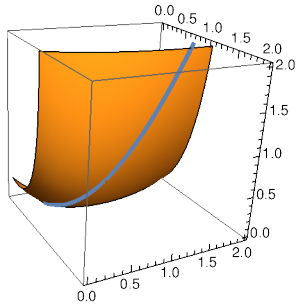
Out[238]= 2.51122

Out[239]= {}

The latter result says that $\Delta[t] \neq 0$ on the domain of $\omega \geq 0$, that is (a, ∞). We see, for instance, this curve lies on the second surface of Q.

In[261]:=
```
Show[ContourPlot3D[Qr[[2]] == 0, {x, 0, 2}, {y, 0, 2}, {z, 0, 2}, Mesh → False],
  ParametricPlot3D[μ[t], {t, a, 10}], ImageSize → Small]
```

Out[261]=



We are not done, we still need to consider the negatives of the square root of ω. But this is the basic method which should be fairly general as we started with a random Q.

We use the above as a template to do the example shown in the screen image of [DLLP]

In[121]:=
```
Q2 = {1 + 2 x y + z^2, 2 - x^2 + y^2 + z^2}
```

Out[121]= $\{1 + 2\,x\,y + z^2,\ 2 - x^2 + y^2 + z^2\}$

In[123]:=
```
cpF2 = criticalPoints3D[Q2, {x, y, z}]
```

Out[123]= {{1.45535, −0.343561, 0.}, {−1.45535, 0.343561, 0.}}

In[124]:=
```
{h2, Ω2, ʊ2} = nsQSIC3D[Q2, cpF2[[1]], {x, y, z}];
```

In[125]:=
```
h2
```

Out[125]=
$$2.01482 + 0.478216\,x + 0.00696313\,x^2 + 2.20142\,x^3 + 1.8164\,y - 0.761615\,x\,y - 1.1613\,x^2\,y + 2.41745\,y^2 + 2.26213\,x\,y^2 - 0.620397\,y^3$$

In[126]:=
```
Ω2[{x, y, z}]
```

Out[126]=
$$\left\{\frac{-0.475535+0.465969\,x+0.589738\,y-0.457109\,z}{0.464506-0.134245\,x+0.783365\,y+0.390579\,z},\ \frac{0.499082-0.332179\,x+0.0455429\,y-0.799062\,z}{0.464506-0.134245\,x+0.783365\,y+0.390579\,z}\right\}$$

In[127]:=
```
ʊ2x[{x_, y_}] = ʊ2[{x, y}][[1]];
ʊ2y[{x_, y_}] = ʊ2[{x, y}][[2]];
ʊ2z[{x_, y_}] = ʊ2[{x, y}][[3]];
```

In[130]:=
```
inflect2 = allInflectionPoints2D[h2, x, y][[1]]
```

Out[130]= {−0.868301, −0.10578}

In[131]:=
```
{w2, Aw2} = weierstrassNormalForm2D[h2, inflect2, x, y]
```

Out[131]= $\{-0.544673-0.529355\,x+1.\,x^3-1.\,y^2, \{\{-0.451752, 0.568297, -0.332142\}, \{-0.697489, 0.290734, 0.753708\}, \{0.728672, 0.202775, 0.654156\}\}\}$

In[132]:=
```
ω2 = w2 /. {x → t, y → 0}
```

Out[132]= $-0.544673-0.529355\,t+1.\,t^3$

In[144]:=
```
u2x = Simplify[ʊ2x[TransformationFunction[Inverse[Aw2]][{t, s}]]];
u2y = Simplify[ʊ2y[TransformationFunction[Inverse[Aw2]][{t, s}]]];
u2z = Simplify[ʊ2z[TransformationFunction[Inverse[Aw2]][{t, s}]]];
μ2x = specialExpand[Numerator[u2x], ω2, s, 1] /. {t → #} &;
μ2y = specialExpand[Numerator[u2y], ω2, s, 1] /. {t → #} &;
μ2z = specialExpand[Numerator[u2z], ω2, s, 1] /. {t → #} &;
Δ2 = specialExpand[Denominator[u2x], ω2, s, 1] /. {t → #} &;
```

Here is our parameterization for Q2, compare with [DLLP] above.

In[173]:=
```
μ2x[t]
μ2y[t]
μ2z[t]
Δ2[t]
```

Out[173]=
$$-1.33227\times10^{-15}-1.80591\,t-1.44353\,t^2-1.45535\,t^3+(2.13788\times10^{-14}+1.23957\times10^{-14}\,t)\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[174]=
$$5.55112\times10^{-17}-2.16386\,t+2.33569\,t^2+0.343561\,t^3+(-9.34093\times10^{-15}+1.8324\times10^{-14}\,t)\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[175]=
$$-6.53777\times10^{-15}-1.59014\times10^{-14}\,t-6.0631\times10^{-16}\,t^2+5.91741\times10^{-15}\,t^3+(2.31988\times10^{-16}+3.60332\,t)\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[176]=
$$4.44089\times10^{-16}-2.52873\,t-2.59678\,t^2+1.\,t^3+(-2.09392\times10^{-14}+3.03155\times10^{-16}\,t)\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

In[186]:=
```
μ2[t_] := {μ2x[t] / Δ2[t], μ2y[t] / Δ2[t], μ2z[t] / Δ2[t]};
```

This will change if one re - runs the above

In[169]:=
```
Reduce[ω2 > 0]
a2 = t /. NSolve[ω2][[3]]
b2 = t /. NSolve[Δ2[t]][[1, 1]]
```

--- Reduce : Reduce was unable to solve the system with inexact coefficients . The answer was obtained by solving a corresponding exact system and numericizing the result .

Out[169]= $t > 1.29839$

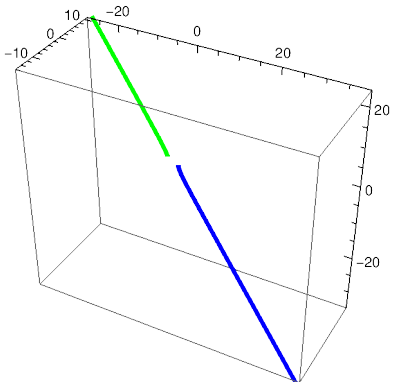
Out[170]= 1.29839

Out[171]= 3.35133

Note here, unlike our random example, there is a zero in the domain of ω so we need to avoid b2 also.

In[190]:=
```
Show[ParametricPlot3D[μ2[t], {t, a2, b2 - .0001}, PlotStyle → Blue],
  ParametricPlot3D[μ2[t], {t, b2 + .0001, 26}, PlotStyle → Green]]
```

Out[190]=



Now we need to consider negatives of square roots of ω

In[191]:=
```
μ2xn = specialExpand[Numerator[u2x], ω2, s, -1] /. {t → #} &;
μ2yn = specialExpand[Numerator[u2y], ω2, s, -1] /. {t → #} &;
μ2zn = specialExpand[Numerator[u2z], ω2, s, -1] /. {t → #} &;
Δ2n = specialExpand[Denominator[u2x], ω2, s, -1] /. {t → #} &;
μ2n[t_] := {μ2xn[t] / Δ2n[t], μ2yn[t] / Δ2n[t], μ2zn[t] / Δ2n[t]}
```

In[196]:=
```
c = NSolve[Δ2n[t]]
```

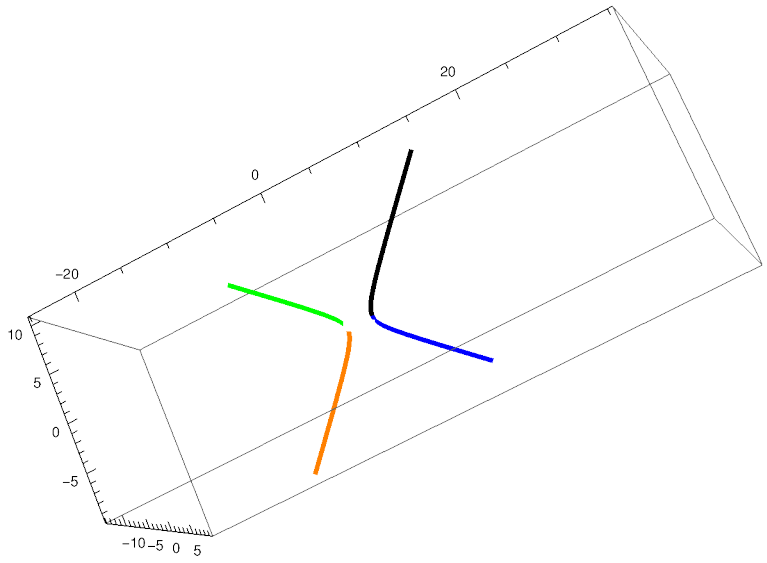
Out[196]=
```
{{t → 3.35133}, {t → 3.35133}, {t → -0.754546}, {t → -0.754546},
  {t → 1.75617 × 10^-16 - 8.66265 × 10^-15 i}, {t → 1.75617 × 10^-16 + 8.66265 × 10^-15 i}}
```

In[197]:=
```
c2 = t /. c[[1, 1]]
```

Out[197]=
```
3.35133
```

In[198]:=
```
Show[ParametricPlot3D[μ2[t], {t, a2, b2 - .0001}, PlotStyle → Blue],
  ParametricPlot3D[μ2[t], {t, b2 + .00001, 26}, PlotStyle → Green],
  ParametricPlot3D[μ2n[t], {t, a2, c2 - .0001}, PlotStyle → Black],
  ParametricPlot3D[μ2n[t], {t, c2 + .0001, 1000}, PlotStyle → Orange]]
```

Out[198]=



As described by [DLLP] we get an oval in projective 3 space. Note that

In[199]:=
```
rpts = RandomReal[{1.3, 3.3}, 4]
lpts = Table[μ2[rpts[[i]]], {i, 4}]
Q2 /. Thread[{x, y, z} → #] & /@ lpts
linearSetMD[lpts, {x, y, z}]
```

Out[199]= {2.98973, 2.17819, 2.98768, 2.17466}

Out[200]= {{14.1295, −5.8276, −12.7938}, {3.44573, −1.32356, −2.84978},
{14.0421, −5.79123, −12.7139}, {3.4316, −1.31727, −2.83561}}

Out[201]= $\{\{-2.84217\times10^{-14}, 3.41061\times10^{-12}\}, \{-1.77636\times10^{-15}, 1.98952\times10^{-13}\},$
$\{-5.68434\times10^{-14}, 3.2685\times10^{-12}\}, \{-3.55271\times10^{-15}, 1.95399\times10^{-13}\}\}$

Out[202]= {}

These random points are on our curve Q2 but are not planar.

3.2.2 Direct use of nsQSIC3D.

The function **nsQSIC3D** can be used directly as the image of Ω, h, returned is a cubic curve which can be path traced and then lifted to $\mathbb{R}^3$ by $\mho$, there is no

need to transform to Weierstrass form and re-format the resulting parameterization to look like that in [DLLP]. Although the method in **nsQSIC3D** follows a classical method to be applied to non-singular QSIC it still works for some singular examples.

In section 2.0 we introduced the famous twisted cubic which is parameterized by $p[t] = \{t,\ t^2,\ t^3\}$. We noticed that the naive curve curve given by the last two equations $\{y - x^2,\ z - x\,y\}$ contained the twisted cubic but also something else contained in the infinite plane of $\mathbb{R}^3$. But **nsQSIC3D** starts by doing a random projective transformation so is a good thing to try when a QSIC has something interesting going on at infinity.

So let

```
Q3 = {y - x^2, z - x y};
```

From the parameterization we see $\{2, 4, 8\}$ is a point on the curve. We apply **nsQSIC3D** to get a curve h3.

```
{h3, Ω3, Ʊ3} = nsQSIC3D[Q3, {2, 4, 8}, {x, y, z}];
```

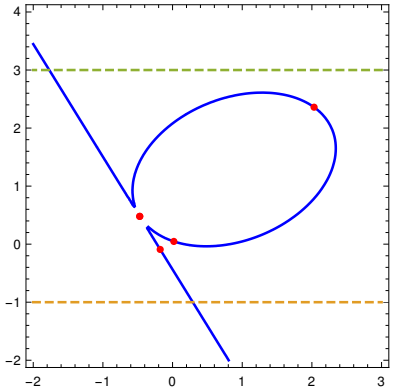
In[266]:=
```
h3
```

Out[266]=
$$0.0138615 - 0.0383705\,x - 0.344181\,x^2 + 0.449508\,x^3 - 0.230084\,y - 1.45555\,x\,y - 0.0402686\,x^2\,y - 0.477413\,y^2 + 0.406024\,x\,y^2 + 0.281218\,y^3$$

We plot

In[267]:=
```
ContourPlot[{h3 == 0, y + 1, y - 3}, {x, -2, 3}, {y, -2, 4}, MaxRecursion → 3,
  ContourStyle → {Blue, Dashed, Dashed}, Epilog → {Red, PointSize[Medium], Point[cp2D]}]
```

Out[267]=



By inspection this looks like the union of a line and an ellipse. We see h3 intersects the horizontal lines $y = 3,\ y = -1$ one point each on the line.

In[268]:=
```
sol1 = {x, y} /. NSolve[{h3, y - 3}]
sol2 = {x, y} /. NSolve[{h3, y + 1}]
```

Out[268]= {{-1.76987, 3.}, {1.40215 - 1.15192 i, 3.}, {1.40215 + 1.15192 i, 3.}}

Out[269]= {{0.192893 - 1.97656 i, -1.}, {0.192893 + 1.97656 i, -1.}, {0.290313, -1.}}

We notice that

In[274]:=
```
Ʊ3[sol1[[1]]]
Ʊ3[sol2[[3]]]
```

Out[274]= $\{14., 8.86404 \times 10^{16}, 3.66954 \times 10^{16}\}$

Out[275]= $\{2., -2.59898 \times 10^{16}, -4.53894 \times 10^{16}\}$

the two points on the line appear to lift to infinite points so the line in h3 comes from an infinite line. It is not hard to guess from the above what this infinite line is in homogeneous variables $\{x, y, z, w\}$. It is $\{x = 0, w = 0\}$.

We can find the affine line though these points

In[270]:=
```
L = linearSetMD[{sol1[[1]], sol2[[3]]}, {x, y}][[1]]
```

Out[270]= 0.195918 + 0.871784 x + 0.449008 y

Dividing h3 by this polynomial

In[271]:=
```
q3 = nDivideMD[h3, L, {x, y}, dTol]
```

Out[271]= $0.0707515 - 0.510676\,x + 0.515618\,x^2 - 1.33654\,y - 0.311758\,x\,y + 0.626308\,y^2$

we get the equation of the ellipse. Critical points are shown on the plot above

In[276]:=
```
cp2D = criticalPoints2D[h3, x, y]
```

Out[276]= {{2.03139, 2.35944}, {-0.177616, -0.09148},
{-0.471063, 0.478272}, {-0.471386, 0.478898}, {0.018471, 0.0468364}}

The third critical point is the intersection of the line and ellipse, but recall from the Plane Curve Book that singular points are not calculated accurately.

In[278]:=
```
L /. Thread[{x, y} → cp2D[[3]]]
q3 /. Thread[{x, y} → cp2D[[3]]]
```

Out[278]= 3.44465×10^{-8}

Out[279]= 4.52791×10^{-8}

We can plot the ellipse using path finding and lift using $\mho 3$

In[257]:=
```
pth1 = pathFinder2D[q3, cp2D[[1]], cp2D[[3]], .3, x, y]
```

Out[257]=
```
{{2.03139, 2.35944}, {1.7855, 2.51268}, {1.50431, 2.59389}, {1.2097, 2.60916},
 {0.91616, 2.56786}, {0.632671, 2.47853}, {0.365284, 2.34778}, {0.118914, 2.18051},
 {-0.101494, 1.98037}, {-0.290071, 1.75037}, {-0.439164, 1.49376},
 {-0.538478, 1.21532}, {-0.574809, 0.923918}, {-0.471063, 0.478272}}
```

In[258]:=
```
pth2 = pathFinder2D[-q3, cp2D[[1]], cp2D[[3]], .3, x, y]
```

Out[258]=
```
{{2.03139, 2.35944}, {2.21613, 2.13865}, {2.32044, 1.86968}, {2.3412, 1.57919}, {2.28835, 1.29001},
 {2.17598, 1.01637}, {2.01659, 0.765887}, {1.81977, 0.542839}, {1.59264, 0.35032},
 {1.34077, 0.191458}, {1.06901, 0.0701613}, {0.782406, -0.00830201}, {0.487498, -0.0368244},
 {0.194318, -0.00643236}, {-0.0808912, 0.0920785}, {-0.471063, 0.478272}}
```

In[280]:=
```
Path1 = ℧3/@pth1
Path2 = ℧3/@pth2
```

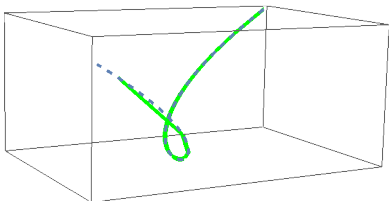
Out[280]=
```
{{-0.632381, 0.399905, -0.252893}, {-0.706213, 0.498737, -0.352214},
 {-0.780865, 0.609751, -0.476133}, {-0.857674, 0.735604, -0.630908},
 {-0.938997, 0.881715, -0.827928}, {-1.02819, 1.05717, -1.08697},
 {-1.12999, 1.27687, -1.44284}, {-1.2516, 1.56651, -1.96065}, {-1.40504, 1.97415, -2.77377},
 {-1.61257, 2.60039, -4.19331}, {-1.92136, 3.69163, -7.09296}, {-2.45237, 6.01413, -14.7489},
 {-3.63434, 13.2084, -48.0038}, {7869.23, 6.24219×10^7, 2.47594×10^11}}
```

Out[281]=
```
{{-0.632381, 0.399905, -0.252893}, {-0.559109, 0.312603, -0.174779},
 {-0.486932, 0.237103, -0.115453}, {-0.416013, 0.173067, -0.0719979},
 {-0.34561, 0.119447, -0.041282}, {-0.274075, 0.075117, -0.0205877},
 {-0.199084, 0.0396346, -0.00789063}, {-0.117642, 0.0138397, -0.00162813},
 {-0.025692, 0.000660079, -0.0000169587}, {0.0827633, 0.00684977, 0.00056691},
 {0.217483, 0.047299, 0.0102867}, {0.396062, 0.156865, 0.0621282},
 {0.654299, 0.428107, 0.28011}, {1.07795, 1.16198, 1.25256},
 {1.93303, 3.7366, 7.22294}, {7869.23, 6.24219×10^7, 2.47594×10^11}}
```

getting some points with large coordinates. That is expected since the third critical point lifts to the infinite plane. So we discard these points while plotting.

In[295]:=
```
Show[
  Graphics3D[{{Green, Thick, Line[Take[Path1, 8]]}, {Green, Thick, Line[Take[Path2, 14]]}}],
  ParametricPlot3D[{t, t^2, t^3}, {t, -1.25, 1.15}, PlotStyle → Dashed]]
```

Out[295]=



Here are other examples, for display we will not re-run nsQSIC3D.

In[184]:=
```
Q0 = {1 - y^2 + z^2 - 4 x y, -3 + y^2 + z^2};
```

By inspection {0,Sqrt[2],1} is a point on this curve.

In[222]:=
```
Q0 /. Thread[{x, y, z} → {0, Sqrt[2], 1}]
```

Out[222]=
```
{0, 0}
```

In[223]:=
```
{h0, Ω0, ʊ0} = nsQSIC3D[Q0, {0, Sqrt[2], 1}, {x, y, z}];  (*non-evaluatable*)
```

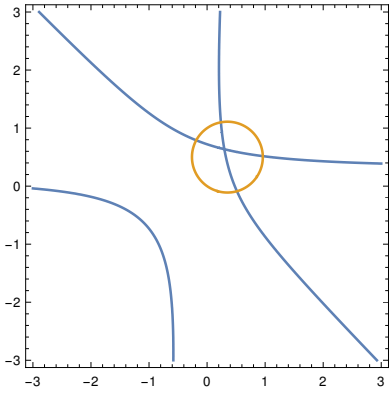
In[266]:=
```
h0 = 2.945656140191897` - 5.217838216917842` x -
     1.9646988724373289` x^2 - 0.07585852051106767` x^3 - 4.806985694928316` y -
     0.20976724723500872` x y + 8.782381871328202` x^2 y +
     1.478912121204738` y^2 + 7.944951077631805` x y^2 - 0.6576260858236058` y^3
```

Out[266]=
$$2.94566 - 5.21784\,x - 1.9647\,x^2 - 0.0758585\,x^3 - 4.80699\,y - 0.209767\,x\,y + 8.78238\,x^2\,y + 1.47891\,y^2 + 7.94495\,x\,y^2 - 0.657626\,y^3$$

Plotting

In[230]:=
```
ContourPlot[{h0 == 0, x^2 + y^2 - .7 x - y == 0}, {x, -3, 3}, {y, -3, 3}, MaxRecursion → 4]
```

Out[230]=



we see h0 is a singular cubic, therefore a rational curve. It follows from the discussion in 3.2.1 that Q0 is a rational curve, further the singular point of Q0 is {1, 0, 0, 0} which is an infinite singularity of Q0. We will leave it as an exercise to plot this curve as above.

We can find 4 real points on the curve as follows

```
In[233]:= pts = {x, y} /. NSolve[{h0, x^2 + y^2 - .7 x - y}, {x, y}, Reals]
Out[233]= {{0.960084, 0.517238}, {0.238219, 1.1}, {0.514636, -0.087703}, {-0.186962, 0.790124}}

In[234]:= Pts = Ʊ0/@ pts
Out[234]= {{-3.25431, -0.294004, 1.70692}, {-6.89478, -0.143543, -1.72609},
           {2.62763, 0.356401, -1.69499}, {2.85034, 0.331553, 1.70002}}

In[235]:= linearSetMD[Pts, {x, y, z}]
Out[235]= {}
```

Thus this is again a non-planar QSIC.

The next example requires luck to get a nice picture, so the following is only for show

```
In[226]:= Q4 = {x^2 + z^2 - 2 y, -3 x^2 + y^2 - z^2}
Out[226]= {x^2 - 2 y + z^2, -3 x^2 + y^2 - z^2}

In[248]:= {h4, Ω4, Ʊ4} = nsQSIC3D[Q5, {0, 0, 0}, {x, y, z}]; (* Non evaluatable *)

In[249]:= h5 (* non evaluatible *)
Out[252]= 1.02149 + 1.37722 x - 1.48595 x^2 - 0.382509 x^3 - 3.52956 y +
           6.36564 x y - 0.769861 x^2 y - 0.224828 y^2 - 1.39048 x y^2 + 1.73501 y^3

In[254]:= Ω4[{x, y, z}]  (* non evaluatible *)
Out[254]= {(0.0401846 x - 0.580388 y - 0.813348 z)/(0.579156 x + 0.676849 y - 0.454372 z),
           (-0.814226 x + 0.452797 y - 0.363334 z)/(0.579156 x + 0.676849 y - 0.454372 z)}

          Simplify[Ʊ4[{x, y}][[1]]]
          Simplify[Ʊ4[{x, y}][[2]]]  (*non-evaluatable*)
          Simplify[Ʊ4[{x, y}][[3]]]
Out[147]= -(0.0703392 (14.4124 + 1. x - 20.2621 y) (-1.1662 + 1. x - 0.780161 y))/
           (0.817124 + 1.18476 x + 1. x^2 - 0.924301 y + 0.792575 x y + 1.19879 y^2)

Out[148]= (1.01591 (1.1662 - 1. x + 0.780161 y)^2)/
           (0.817124 + 1.18476 x + 1. x^2 - 0.924301 y + 0.792575 x y + 1.19879 y^2)

Out[149]= (1.42368 (-1.1662 + 1. x - 0.780161 y) (0.558644 + 1. x + 0.446715 y))/
           (0.817124 + 1.18476 x + 1. x^2 - 0.924301 y + 0.792575 x y + 1.19879 y^2)
```

We notice the following linear factor appears in each numerator, so Ʊ is identically {0,0,0} on this line!

In[267]:= **line4 = −1.1661999857316967` + 1.` x − 0.7801613607079093` y (∗ evaluatable∗)**

Out[267]= −1.1662 + 1. x − 0.780161 y

In fact, this line is a factor of h4

In[152]:= **qf = nDivideMD[h4, line, {x, y}, dTol] (∗ non−evaluatable ∗)**

In[268]:= **qf = −0.875916120629441` − 1.9320362297177929` x − 0.38250867953478784` x^2 +**
3.6125172138379646` y − 1.068279503647881` x y − 2.223905683299877` y^2
(∗ this is evaluatable compare with the above ∗)
h4 = Expand[qf ∗ line4]

Out[268]= −0.875916 − 1.93204 x − 0.382509 x^2 + 3.61252 y − 1.06828 x y − 2.22391 y^2

Out[269]= 1.02149 + 1.37722 x − 1.48595 x^2 − 0.382509 x^3 − 3.52956 y +
6.36564 x y − 0.769861 x^2 y − 0.224828 y^2 − 1.39048 x y^2 + 1.73501 y^3

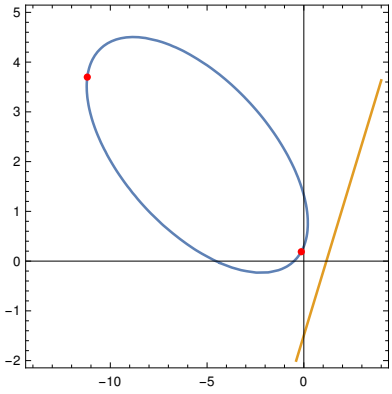
In[270]:= **cpqf = criticalPoints2D[qf, x, y]**

Out[270]= {{−11.1898, 3.69873}, {−0.131493, 0.188522}}

The contour plot of h4 is then

In[271]:= **ContourPlot[{qf == 0, line4 == 0}, {x, −14, 4}, {y, −2, 5},**
Axes → True, Epilog → {Red, PointSize[Medium], Point[cpqf]}]

Out[271]=



We can path trace qf

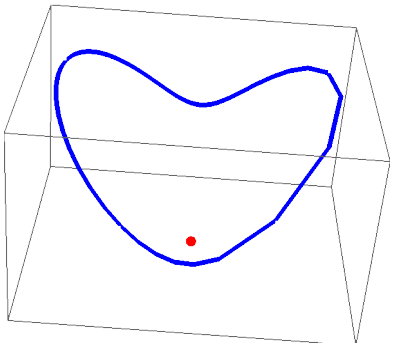
In[278]:= **pth1 = pathFinder2D[qf, cpqf[[1]], cpqf[[2]], .25, x, y, maxit → 70];**
pth2 = pathFinder2D[−qf, cpqf[[1]], cpqf[[2]], .25, x, y, maxit → 50];
pth4 = Join[pth1, Reverse[pth2]];

Now we lift to Q4

In[281]:= **PTH4 = ℧4/@ pth4; (∗ non−evaluatable ∗)**

In[282]:=
```
Graphics3D[{{Blue, Thick, Line[PTH4]}, {Red, PointSize[Large], Point[{0, 0, 0}]}}]
(* non evaluatable *)
```

Out[282]=



Thus we get an oval with an isolated point for this QSIC.

3.2.3 Plotting by projection

Often the easiest way to identify and plot QSIC is simply to project to a plane quartic, path trace the plane curve and use **fFiberMD** to lift back to $\mathbb{R}^3$. The latter only works with affine projections so the previous method is preferable, assuming it works, if you want to capture some feature on the infinite plane. Here is one of my favorite QSIC

In[201]:=
```
Q5 = {x^2+y^2+z^2-16, 57-12x+4x^2+y^2-64z+16z^2};
```

We first project.

In[202]:=
```
h5 = FLTMD[Q5, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

Out[202]=
$$1.-0.0344652\,x-0.033403\,x^2+0.00677934\,x^3+0.00134825\,x^4-$$
$$1.62713\,y-0.0245903\,x\,y+0.0241702\,x^2\,y-0.00181159\,x^3\,y+0.893879\,y^2+$$
$$0.0164066\,x\,y^2-0.00620129\,x^2\,y^2-0.208146\,y^3-0.000832304\,x\,y^3+0.0196645\,y^4$$

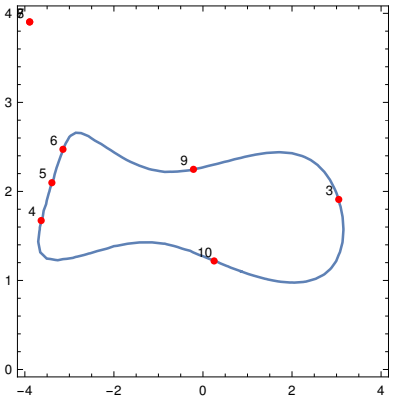
We first find and label critical points.

In[203]:=
```
cp5 = criticalPoints2D[h5, x, y];
ap5 = Association[Table[i → cp5[[i]], {i, 10}]]
```

Out[204]=
```
<| 1 → {-539.117, -251.121}, 2 → {-539.117, -251.121},
  3 → {3.04937, 1.90971}, 4 → {-3.63348, 1.67267}, 5 → {-3.39309, 2.09769},
  6 → {-3.14407, 2.47282}, 7 → {-3.89168, 3.90354}, 8 → {-3.89168, 3.90354},
  9 → {-0.213013, 2.24798}, 10 → {0.250554, 1.21911} |>
```

In[207]:=
```
Show[ContourPlot[h5 == 0, {x, −4, 4},
    {y, −0, 4}, Epilog → {Red, PointSize[Medium], Point[cp5]}],
  Graphics[Table[{PointSize[Medium], Text[i, ap5[i] + {−.2, .1}]}, {i, 10}]]]
```

Out[207]=



Note that there are two isolated singularities, points 1-2 and 7-8.

In[208]:=
```
fFiberMD[Q5, prd3D, cp5[[1]], {x, y, z}, 1.*^−6]
fFiberMD[Q5, prd3D, cp5[[7]], {x, y, z}, 1.*^−6]
```

» no point in fiber at {−539.117, −251.121}

Out[208]= {}

» no point in fiber at {−3.89168, 3.90354}

Out[209]= {}

These are artifactual isolated points, it should be noted that they must be here, since this is a quartic of genus 1, see section 3.3 or Plane Curve Book.

We now plot paths in the plane, output omitted, some trial and error was used.

In[215]:=
```
pth1 = pathFinder2D[−h5, cp5[[6]], cp5[[9]], .2, x, y];
pth2 = pathFinder2D[−h5, cp5[[9]], cp5[[3]], .2, x, y];
pth3 = pathFinder2D[−h5, cp5[[3]], cp5[[10]], .2, x, y];
pth4 = pathFinder2D[−h5, cp5[[10]], cp5[[4]], .14, x, y, maxit → 40];
pth5 = pathFinder2D[−h5, cp5[[4]], cp5[[6]], .05, x, y];
```

Then we lift

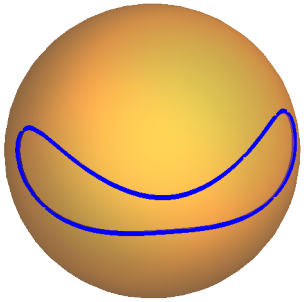
In[220]:=
```
Pth1 = Flatten[fFiberMD[Q5, prd3D, #, {x, y, z}, 1.*^−6], 1] &/@ pth1;
Pth2 = Flatten[fFiberMD[Q5, prd3D, #, {x, y, z}, 1.*^−8], 1] &/@ pth2;
Pth3 = Flatten[fFiberMD[Q5, prd3D, #, {x, y, z}, 1.*^−8], 1] &/@ pth3;
Pth4 = Flatten[fFiberMD[Q5, prd3D, #, {x, y, z}, 1.*^−8], 1] &/@ pth4;
Pth5 = Flatten[fFiberMD[Q5, prd3D, #, {x, y, z}, 1.*^−8], 1] &/@ pth5;
```

And now we can show our single oval with the first surface, a sphere, as the

background.

```
In[225]:= Show[ContourPlot3D[x^2+y^2+z^2 == 16, {x, -4, 4},
    {y, -4, 4}, {z, -4, 4}, Mesh → False, ContourStyle -> Opacity[.5]],
  Graphics3D[{Thick, Blue, Line[Pth1], Line[Pth2], Line[Pth3], Line[Pth4], , Line[Pth5]}],
  Boxed → False, Axes → False]
```

Out[225]=



3.3 .4 Some more examples from [TWMW].

We give some more examples from the classification of QSIC in [TWMW]. In some cases it will be enough just to project to the plane.

Example 6

```
In[127]:= Q6 = {x^2+y^2+z^2-1, x^2+2y^2};
```

We project with our pseudo-random projection.

```
In[128]:= h6 = FLTMD[Q6, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

$$\text{Out[128]}= 1. + 1.27881\,x^2 + 0.903815\,x^4 + 0.162081\,x\,y - 0.451232\,x^3\,y - 2.04542\,y^2 - 1.14578\,x^2\,y^2 - 0.165762\,x\,y^3 + 1.04594\,y^4$$

A contour plot with any scale gives nothing. But looking for critical points we get 4 distinct points of multiplicity 2.

```
In[130]:= cp6 = criticalPoints2D[h6, x, y]
Out[130]= {{-0.636105, -1.13489}, {-0.636105, -1.13489}, {0.636105, 1.13489},
    {0.636105, 1.13489}, {0., 0.988834}, {0., 0.988834}, {0., -0.988834}, {0., -0.988834}}
```

To show non-existence we use **fFiberMD** with a loose tolerance

In[132]:=
```
fFiberMD[Q6, prd3D, cp6[[1]], {x, y, z}, 1.*^-6]
fFiberMD[Q6, prd3D, cp6[[3]], {x, y, z}, 1.*^-6]
```

» no point in fiber at {-0.636105, -1.13489}

Out[132]= {}

» no point in fiber at {0.636105, 1.13489}

Out[133]= {}

The first two points are artifacts. For the second two we use **fFiberMD** with a tight tolerance to show existence.

In[137]:=
```
fFiberMD[Q6, prd3D, cp6[[5]], {x, y, z}, 1.*^-12]
fFiberMD[Q6, prd3D, cp6[[7]], {x, y, z}, 1.*^-12]
```

Out[137]= $\{\{5.78815\times 10^{-13}, 1.85504\times 10^{-13}, 1.\}\}$

Out[138]= $\{\{-4.94438\times 10^{-13}, -1.5847\times 10^{-13}, -1.\}\}$

So {0, 0, 1}, {0, 0, -1} are points on Q6.Since no other real critical points show up we conclude that there are no other real points. There are many complex points, remove the condition "Reals" from the critical point code

In[144]:=
```
criticalPoints3DC[{f_, g_}, {x_, y_, z_}] := Module[{J, ob},
    ob = RandomReal[{.7, 1.3}, 3].{x^2, y^2, z^2};
    J = D[{f, g, ob}, {{x, y, z}}];
    DeleteDuplicates[{x, y, z} /. NSolve[{f, g, N[Det[J]]}]]]
```

In[145]:=
```
criticalPoints3DC[Q6, {x, y, z}]
```

Out[145]= {{-1.41421, 0. + 1. i, 0.}, {-1.41421, 0. - 1. i, 0.},
{1.41421, 0. + 1. i, 0.}, {1.41421, 0. - 1. i, 0.}, {0., 0., 1.}, {0., 0., -1.}}

Thus this real QSIC is a two point set but the complex solution has non-isolated components. A similar example $\{y^2 - z^2 + 2\,z,\, x^2 + z^2\}$ has only one real point.

Example 7:

Here is a case where **nsQSIC3D** does not tell the whole story. We have a reducible curve consisting of a plane quadric and 2 lines, thus very definitely of degree 4 and not capable of being modelled by a plane cubic.

```
Q7 = {2 x y - y^2, y^2 + z^2 - 1};
```

We see that $\{x = 0,\ y^2 + z^2 = 0\}$ is a plane circle contained in Q7.

We project to the plane with our standard pseudo-random projection.

In[146]:=
```
h7 = FLTMD[Q7, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

Out[146]=
$$1. - 1.80651\,x^2 + 0.350558\,x^4 + 0.653265\,x\,y - 1.44199\,x^3\,y - 2.04542\,y^2 + 1.56429\,x^2\,y^2 - 0.668101\,x\,y^3 + 1.04594\,y^4$$

The result is a circle and two lines in the plane.

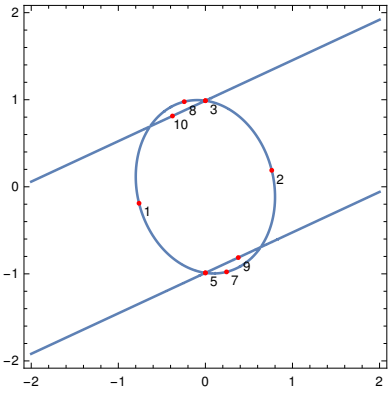
In[149]:=
```
cp7 = criticalPoints2D[h7, x, y];
acp7 = <|Table[i → Chop[cp7[[i]]], {i, 10}]|>
```

Out[150]=
```
<| 1 → {-0.761959, -0.189212}, 2 → {0.761959, 0.189212}, 3 → {0, 0.988834},
  4 → {0, 0.988834}, 5 → {0, -0.988834}, 6 → {0, -0.988834}, 7 → {0.242741, -0.977522},
  8 → {-0.242741, 0.977522}, 9 → {0.378051, -0.813048}, 10 → {-0.378051, 0.813048} |>
```

We see points 3, 5 are singular critical points but surprisingly the other two apparent intersection points were not picked up as critical points.

In[154]:=
```
Show[ContourPlot[h7 == 0, {x, -2, 2},
    {y, -2, 2}, MaxRecursion → 4, Epilog → {Red, Point[cp7]}],
  Graphics[{Table[Text[i, acp7[i] + {.1, -.1}], {i, {1, 2, 3, 5, 7, 8, 9, 10}}]}]]
```

Out[154]=



We lift points on the lines to $\mathbb{R}^3$.

In[156]:=
```
p1=fFiberMD[Q7,prd3D,acp7[3],{x,y,z},1.*^-12][[1]]
p2=fFiberMD[Q7,prd3D, acp7[10],{x,y,z},dTol][[1]]
p3=fFiberMD[Q7,prd3D, acp7[5],{x,y,z},dTol][[1]]
p4=fFiberMD[Q7,prd3D, acp7[9],{x,y,z},dTol][[1]]
```

Out[156]=
$$\{5.82867 \times 10^{-16},\ 1.8735 \times 10^{-16},\ 1.\}$$

Out[157]=
$$\{1.23871,\ 5.55112 \times 10^{-16},\ 1.\}$$

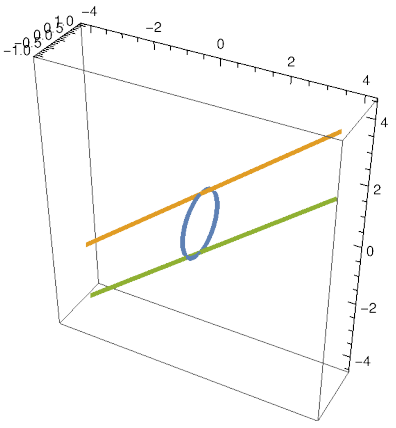
Out[158]=
$$\{-5.82867 \times 10^{-16},\ -1.8735 \times 10^{-16},\ -1.\}$$

Out[159]=
$$\{-1.23871,\ -4.996 \times 10^{-16},\ -1.\}$$

We can now plot in $\mathbb{R}^3$

In[160]:=
```
ParametricPlot3D[{{0, Cos[t], Sin[t]}, p1 + t * p2, p3 + t * p4}, {t, -Pi, Pi}]
```

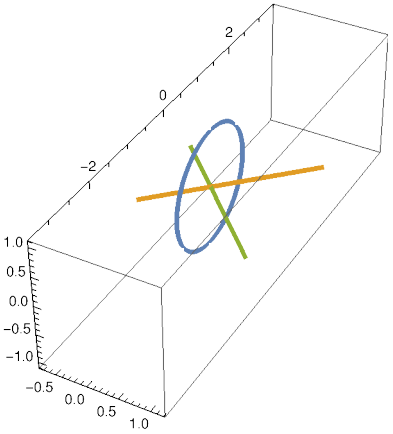
Out[160]=



Comment : In this example there is a circle and two lines through a common infinite point, each line intersecting the circle. Suppose instead the two lines do not touch the circle, for example the curve in $\mathbb{R}^3$ looks like

In[187]:=
```
ParametricPlot3D [{{0, Cos[t], Sin[t]}, {t, t, 0}, {t, -t, 0}}, {t, -Pi, Pi}]
```

Out[187]=



This is no longer a QSIC. Using the method of section 2.7 we see this configuration requires 4 equations, one of degree 2 but 2 of degree 3 and one of degree 4.

Out[188]=
$$\{1.\,x\,z, -1.\,x^3 + 1.\,x\,y^2, -1.\,z + 1.\,y^2\,z + 1.\,z^3, 1.\,x^2 - 1.\,x^4 - 1.\,y^2 + 1.\,y^4 + 1.\,y^2\,z^2\}$$

There are, according to[TWMW] 8 cases with the QSIC a union of 2, 3 or 4 lines.In Chapter 4 I plan to cover unions of lines in $\mathbb{R}^3$ more thoroughly, in particular where situations as in the comment are more common.

3.3 Birational equivalence and Genus

In the plane curve book we gave little emphasis to the idea of genus. For most of the results there the important number was the degree of a curve. But more importantly we viewed the genus from the standpoint of the Clebsch-Noether formula which, in fact, is not numerically stable. A perturbation could drastically change this, in fact every numerical curve is only a small perturbation away from being non-singular.

However, for space curves things are different. The degree is not the best parameter, especially when we have curves defined by an over-determined system. Even in section 3.2 where we had naive curves the degree was 4 but we saw these curves tended to be related to plane cubics. We will see the explanation is the genus. We will find, instead of Clebsch-Noether a more numerically stable way to calculate genus. But, as we also saw in this last section the role which we had previously given to FLT is now taken up with *birational equivalence.*

3.3.1 Elliptic Curves and functions.

Historically the formalization of the notion of genus began with Riemann and the Riemann-Roch Theorem (1857-1865). But some of the ideas surfaced as early as the early 1800. At that point a main interest was working out closed form integration formulas. In particular the integral

$$\int_0^{\phi} \frac{du}{\sqrt{1-\kappa \, \text{Sin}^2 \, u}}, \; 0 \le \kappa < 1$$

attracted special attention as it required new functions to give a closed form. These functions became known as *elliptic functions.* (For an elementary account see Chapter 6 of my *Theory of Equations* book https://barryhdayton.space/theoryEquations/theq6.pdf). Using these and then standard methods of integration indefinite integrals of the form

$$\int \frac{dx}{\sqrt{x^4+a\,x^2+b\,x+c}}, \int \frac{dx}{\sqrt{x^3+a\,x+b}}$$

could be expressed in terms of these elliptic functions. This suggested that the equations defining the denominators

$$y^2-(x^4+a\,x^2+b\,x+c), \; y^2-(x^3+a\,x+b)$$

could be called *elliptic curves.* From our study of QSIC we can show how these are related. So we can use our numerical methods let us take an explicit example:

```
In[129]:= f = y^2-(x^4+3x^2-2x+2);
```

We form a QSIC by adding a new variable $z = x^2$ getting

In[125]:=
```
q1 = y^2 - (z^2 + 3 z - 2 x + 2);
q2 = z - x^2;
Q = {q1, q2}
```

Out[127]= $\{-2 + 2x + y^2 - 3z - z^2, -x^2 + z\}$

We note the following simple algebraic maps between the curve f and the QSIC Q.

In[121]:=
```
Φ = {#[[1]], #[[2]], #[[1]]^2} &;
Θ = Take[#, 2] &;
```

where Θ is actually the projection on the first 2 coordinates.

In[130]:=
```
cpf = criticalPoints2D[f, x, y]
```

Out[130]= {{0.24284, 1.30181}, {0.24284, -1.30181}}

Then note that as claimed q is a point on Q.

In[132]:=
```
q = Φ[cpf[[1]]]
Q /. Thread[{x, y, z} → q]
```

Out[132]= {0.24284, 1.30181, 0.0589711}

Out[133]= $\{5.05151 \times 10^{-15}, 0.\}$

So we can now use

In[134]:=
```
{h, Ω, ʊ} = nsQSIC3D[Q, q, {x, y, z}];
```

We get a cubic

In[135]:=
```
h
```

Out[135]= $-1.01523 + 3.51199\,x - 0.395834\,x^2 - 0.451825\,x^3 + 0.960383\,y - 1.54767\,x\,y + 0.825717\,x^2\,y - 0.639259\,y^2 + 1.8056\,x\,y^2 - 0.154608\,y^3$

Let p2 be the point on h given by

In[145]:=
```
q2 = Φ[cpf[[2]]]
p2 = Ω[q2]
h /. Thread[{x, y} → p2]
```

Out[145]= {0.24284, -1.30181, 0.0589711}

Out[146]= {2.70196, 0.619418}

Out[147]= 1.77636×10^{-15}

Putting this cubic in Weierstrass form

In[137]:= **afl = allInflectionPoints2D[h, x, y]**

Out[137]= {{0.327046, 1.79307}, {0.235602, −2.95013}, {0.293491, 0.052556}}

In[138]:= **{wh, Awh} = weierstrassNormalForm2D[h, afl[[1]], x, y]**

Out[138]= $\{-0.776489 - 2.00209\,x + 1.\,x^3 - 1.\,y^2, \{\{-0.899271, -0.305709, 0.842261\},$
$\{0.0938218, 0.814567, 0.587697\}, \{0.986819, -0.156009, -0.0430005\}\}\}$

Note a we get point of wh which is in the image of our combined map fltMD[Ω[Φ[{x,y}]]]

In[154]:= **wp2 = fltMD[p2, Awh]**
wh /. Thread[{x, y} → wp2]
fltMD[Ω[Φ[cpf[[2]]]], Awh]

Out[154]= {−0.703244, 0.532612}

Out[155]= -3.59712×10^{-14}

Out[156]= {−0.703244, 0.532612}

This combined map can be simplified to

In[162]:= **α = Simplify[fltMD[Ω[Φ[{x, y}]], Awh]]**

Out[162]= $\left\{\frac{-1.27575 + 1.68777\,x - 0.851591\,x^2 + 0.703722\,y}{1.23811 + 0.0231189\,x + 1.\,x^2 - 1.00068\,y}, \frac{0.723771 + 0.276694\,x - 1.6471\,x^2 - 0.532974\,y}{1.23811 + 0.0231189\,x + 1.\,x^2 - 1.00068\,y}\right\}$

which is a rational algebraic function.

Going the other way we get a rational algebraic function

In[169]:= **β = Simplify[Θ[Ʊ[fltMD[{x, y}, Inverse[Awh]]]]]**

Out[169]= $\left\{\frac{421.658 + 250.845\,x - 326.455\,x^2 + 37.0921\,y - 22.2616\,x\,y + 2.99488\,y^2}{181.37 - 74.9655\,x + 1.\,x^2 + 426.963\,y - 5.93584\,x\,y + 0.266356\,y^2},\right.$
$\left.\frac{-359.535 - 45.8446\,x^2 + 206.315\,y + 34.05\,y^2 + x\,(357.182 + 241.8\,y)}{181.37 - 74.9655\,x + 1.\,x^2 + 426.963\,y - 5.93584\,x\,y + 0.266356\,y^2}\right\}$

In[165]:= **p3 = β /. Thread[{x, y} → wp2]**
f /. Thread[{x, y} → p3]

Out[165]= {0.24284, −1.30181}

Out[166]= 0.

Thus we have a *birational equivalence* between the quartic curve f and the cubic curve wh.

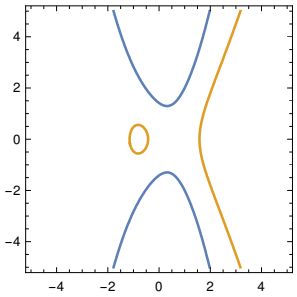
In the plane curve book we noted the non-singular cubic curve was of genus

1, because of this we claim the quartic curve is also of genus 1.

We end this discussion with a little geometry.

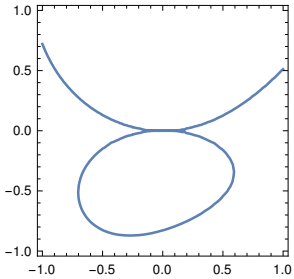
```
In[168]:= ContourPlot[{f == 0, wh == 0}, {x, -5, 5}, {y, -5, 5}, ImageSize → Small]
```

Out[168]=



In the affine plane both f and wh have two components. In fact further experimentation with these birational maps the reader can check that the smaller component of wh maps by β to the negative component of f while the large component of wh maps to the positive component of f.

However in the projective plane there is a difference, f is connected as these two affine components share the same infinite point {0,1,0}. Using ip2z in the plane curve book we get a plot near this infinite point which is a non-ordinary singularity of f.



In fact from the Clebsh-Noether formula f must have Clebsh number 2 in order to arrive at genus 1. So the birational map α actually breaks this singularity into two pieces. Birational maps have denominators so are not defined everywhere and α cannot be defined at this singularity because wh is non-singular.

For the convenience of the reader who wants to experiment with these maps here are the full precision expressions for wh, α and β.

```
wh = -0.7764892315302467` - 2.0020871428383487` x + 1.0000000000000004` x^3 - 1.` y^2;
α = {(-1.2757508990903774` + 1.6877669409285232` x -
        0.8515908479957427` x^2 + 0.7037222750771311` y) / (1.2381110937061424` +
        0.023118907708649807` x + 1.` x^2 - 1.0006802451169017` y),
    (0.7237711271825419` + 0.27669402085278955` x - 1.6471028507460224` x^2 -
        0.5329744284257336` y) / (1.2381110937061424` +
        0.023118907708649807` x + 1.` x^2 - 1.0006802451169017` y)};
β = {(421.65792029050067` + 250.8446741231946` x - 326.45452844178726` x^2 +
        37.0921077782662` y - 22.26157450698797` x y + 2.994881185534921` y^2) /
      (181.37021636228292` - 74.96553026772098` x + 1.` x^2 + 426.9632597935131` y -
        5.935844588204833` x y + 0.26635570919983936` y^2),
    (-359.53452326731184` - 45.84457497214858` x^2 + 206.31534505924515` y +
        34.05004604309161` y^2 + x (357.1822933061392` + 241.80005157723164` y)) /
      (181.37021636228292` - 74.96553026772098` x + 1.` x^2 + 426.9632597935131` y -
        5.935844588204833` x y + 0.26635570919983936` y^2)};
```

3.3.2 Blowing Up plane curves without exceptional curves

This is an important classical idea used to remove singularities by going up a dimension. Because of the limiting classical techniques, eg. no numerics, this becomes quite hard and the blown up curve has an extra component called the *exceptional curve*. Classical algebraic geometers leave this in and are able to make good use of it. However we can remove this exceptional curve which makes things cleaner and more understandable.

Given a plane curve f(x,y) with singularities at various points $p_1, \ldots, p_k$ we construct a rational function g(x,y) in x, y with denominator vanishing at the singular points and set $z_i = g_i(x, y)$, a different variable for each singular point. We get a curve $F = \{f, z_1 - g_1, \ldots, z_k - g_k\}$. In general the inverse image, fiber, of a particular singular point is itself a curve. We get a rational map $\Phi : f \longrightarrow F$ with projection on the x,y plane a left inverse. We use dual interpolation to remove these exceptional curves and make ϕ a birational isomorphism. Note below that dual interpolation works best with only a few random points and the lowest m possible. Randomness of the points is important and we can get a good random set by using randomRealRegular - Points2D from the plane curve book (see Global Functions 71).

Before starting we mention that one measure of a plane singularity that we can easily deal with is the multiplicity. This concept has been recently clarified by Araceli Bonifant and John Milnor in a long article on plane curve theory (mostly complex) in the AMS Bulletin, Volume 57, Number 2, April 2020 page 235. They define the *multiplicity of a plane singularity at p* to be the intersection multiplicity at p of the curve and a *generic* line

through p. For us a generic line is a random line. Here is some code to do this calculation in the plane case. Here $f = 0$ is a plane curve and p is a point, possibly complex but not infinite, on f.

```
singPointMult2D[f_, p_, x_, y_, tol_] := Module[{l},
    l = line2D[p, p + RandomReal[{-.2, .2}, 2], x, y];
    multiplicityMD[{f, l}, p, {x, y}, tol]]
```

Here is our first example.

3.3.2.1 The node

Consider the basic plane nodal cubic. It has a double point at the origin.

In[115]:=
```
f1 = y ^ 2 - x ^ 3 - x ^ 2;
```

We add a new variable z and set it equal to $z = \frac{y}{x}$ getting the new equation $y - x\,z$. We now consider the curve in $\mathbb{R}^3$

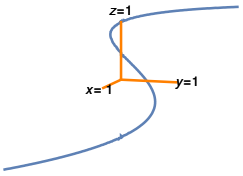
In[116]:=
```
F1 = {f1, y - x z};
```

We note that the entire z - axis is contained in F, in fact it is a double line which is invisible in a contour plot. This is our exceptional curve.

In[117]:=
```
showProjection3D [F1, fprd3D, 6, {x, y, z}, {x, y}, 2]
```

» projection Function $\{1.\,x^2 - 2.37355\,x^3 + 0.0574214\,x^4 + 0.000243617\,x^5 - 2.18663\,x^2\,y + 0.955595\,x^3\,y + 0.0153028\,x^4\,y - 1.02271\,x^2\,y^2 + 0.320413\,x^3\,y^2 + 2.2363\,x^2\,y^3\}$

Out[117]=



We see the equation of our pseudo - random projection is divisible by x^2 which is what makes it double and invisible.

An important thing for us is the rational maps between f and F.

In[118]:=
```
Φ := Append [#, #[[2]] / #[[1]]] &
Θ := Take[#, 2] &
```

At this point we have Θ as a left inverse of Φ. We need to remove the exceptional curve to get the birational equivalence.

In[120]:=
```
Θ[Φ[{x, y}]]
```

Out[120]=
```
{x, y}
```

We now, somewhat by trial and error choose a small number of points on f and lift those to the curve F by Φ.

In[121]:=
```
L = randomRealRegularPoints2D [f1, {{−2, 5}, {−5, 5}}, x, y, 5]
P = Φ /@ L
F1 /. Thread[{x, y, z} → #] & /@ P
```

Out[121]= {{1.10454, −1.60237}, {1.72728, 2.85251}, {−0.477635, −0.34521}, {1.36756, −2.10425}, {1.71515, −2.82616}}

Out[122]= {{1.10454, −1.60237, −1.4507}, {1.72728, 2.85251, 1.65145},
{−0.477635, −0.34521, 0.722748}, {1.36756, −2.10425, −1.53869}, {1.71515, −2.82616, −1.64777}}

Out[123]= $\left\{\left\{1.59872 \times 10^{-14}, 0.\right\}, \left\{3.55271 \times 10^{-15}, 0.\right\}, \left\{-8.32667 \times 10^{-17}, 5.55112 \times 10^{-17}\right\},$
$\left\{-1.24345 \times 10^{-14}, 0.\right\}, \left\{-1.45661 \times 10^{-13}, 4.44089 \times 10^{-16}\right\}\right\}$

```
B1 = dualInterpolationMD [F, P, 4, {x, y, z}, 1.*^−7]
```

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

» Final Hilbert Function {1, 3, 3, 3, 3}

Out[149]= $\left\{-1.\, y + 1.\, x\, z, -1.\, x - 1.\, x^2 + 1.\, y\, z, -1. - 1.\, x + 1.\, z^2\right\}$

Note G contains the image of Φ even though the original equation f is not present.

In[126]:=
```
p = randomRealRegularPoints2D [f1, {{−2, 5}, {−5, 5}}, x, y, 1][[1]]
B1 /. Thread[{x, y, z} → Φ[p]]
```

Out[126]= {−0.554447, 0.370092}

Out[127]= $\left\{-1.22125 \times 10^{-15}, -4.00863 \times 10^{-12}, 8.18939 \times 10^{-12}\right\}$

However a typical point on the exceptional curve is not in G so, with a little more effort we see that Φ,Θ are inverse functions from f, away from {0,0} and G.

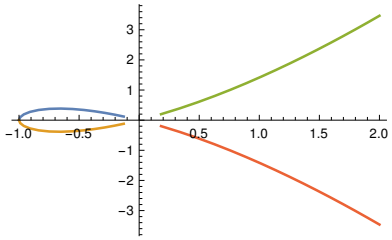
In[128]:=
```
B /. Thread[{x, y, z} → {0, 0, 3.13}]
```

Out[128]= {0., 0., 8.7969}

Finally we can plot B, the *blowup* of f using 2 dimensional path tracing and lifting by Φ.

In[134]:=
```
pth1 = Drop[pathFinder2D [f1, {−1, 0}, {0, 0}, .1, x, y], −1];
pth2 = Drop[Reverse[pathFinder2D [−f1, {−1, 0}, {0, 0}, .1, x, y]], 1];
pth3 = Drop[pathFinder2D [f1, {2, N[Sqrt[2 ^ 2 + 2 ^ 3]]}, {0, 0}, .25, x, y], −1];
pth4 = Drop[pathFinder2D [−f1, {2, −N[Sqrt[2 ^ 2 + 2 ^ 3]]}, {0, 0}, .25, x, y], −1];
ListLinePlot[{pth2, pth1, pth3, pth4}]
```

Out[138]=



In[139]:=
```
Pth1 = Φ /@ pth1;
Pth2 = Φ /@ pth2;
Pth3 = Reverse[Φ /@ pth3];
Pth4 = Φ /@ pth4;
```

Before plotting we want to add in our exceptional line. We can find out where it intersects B1

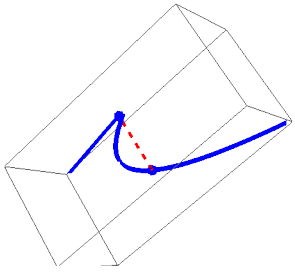
In[143]:=
```
excpt1 = fFiberMD[B1, {{1, 0, 0}, {0, 1, 0}}, {0, 0}, {x, y, z}, dTol]
```

» multiple fiber points {0, 0}

Out[143]=
```
{{0., 0., 1.}, {0., 0., -1.}}
```

In[144]:=
```
Graphics3D[{{Blue, Thick, Line[Join[Pth4, Pth2]]}, {Blue, Thick, Line[Join[Pth1, Pth3]]},
    {Blue, PointSize[Large], Point[excpt1]}, {Red, Thick, Dashed, Line[excpt1]}}, ImageSize → Small]
```

Out[144]=



Comment: We could handle the node $y^2 - x^3$ similarly but this curve only goes through the singularity {0,0} once (eg: as the parametric curve $\{t^2, t^3\}$) so there is only one point in the blow up over the singularity. In this case the blow-up is tangent to the exceptional line. We leave it for the reader to plot this.

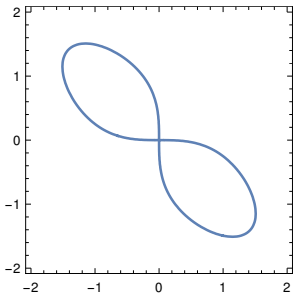
3.3.2.2 A lemniscate

Consider the lemniscate

In[282]:=
```
f2 = x^4 + 4 x y + y^4;
```

In[146]:=
```
ContourPlot[f2 == 0, {x, -2, 2}, {y, -2, 2}, ImageSize → Small]
```

Out[146]=



This is similar to the node above but brings up several issues not present in the node since this is a bounded curve and should have a bounded blow-up. Our method calls for a rational function with the denominator vanishing at

the singular point {0,0}. In particular the denominator and curve intersect in a multiple point of multiplicity greater than 1 because of the singularity of f. We should choose this denominator so that the multiplicity of the intersection of the denominator is the multiplicity of the singularity In the case of the node the multiplicity is calculated by

```
In[284]:= singPointMult2D[f2, {0, 0}, x, y, dTol]
Out[284]= 2
```

But if we attempt to use the rational function $z = \frac{y}{x}$ here

```
In[149]:= multiplicityMD[{f2, x}, {0, 0}, {x, y}, dTol]
Out[149]= 4
```

This could introduce infinite points above the singularity. Therefore we use, instead, the rational function $z = \frac{x+y}{x-y}$. Then we are back to

```
In[150]:= multiplicityMD[{f2, x - y}, {0, 0}, {x, y}, dTol]
Out[150]= 2
```

Another consideration in this bounded case is that to avoid infinite points in the blow-up then the curve of the denominator should not intersect our curve f in a real point other than the singularity. This is not a problem:

```
NSolve[{f2, x - y}]
Out[129]= {{x → 0. - 1.41421 i, y → 0. - 1.41421 i},
  {x → 0. + 1.41421 i, y → 0. + 1.41421 i}, {x → 0., y → 0.}, {x → 0., y → 0.}}
```

So we proceed

```
In[151]:= F2 = {f2, z (x - y) - (x + y)};
          Φ = Append[#, (#[[1]] + #[[2]]) / (#[[1]] - #[[2]])] &
          Θ = Take[#, 2] &
Out[152]= Append[#1, (#1[[1]] + #1[[2]]) / (#1[[1]] - #1[[2]])] &
Out[153]= Take[#1, 2] &
```

We may need several attempts before finding a suitable system eliminating the exceptional component.

In[160]:=
```
L = randomRealRegularPoints2D[f2, {{-2, 2}, {-2, 2}}, x, y , 5];
P = Φ /@ L
F2 /. Thread[{x, y, z} → #] &/@ P
```

Out[161]=
$$\{\{-0.81389, 0.134885, 0.715664\},\ \{0.963838, -0.224506, 0.622153\}, \{0.784433, -0.12074, 0.733222\},\ \{-1.43947, 0.826858, 0.270311\}, \{-0.900132, 0.182639, 0.662645\}\}$$

Out[162]=
$$\{\{9.01348\times 10^{-14}, 0.\}, \{-9.0847\times 10^{-14}, 0.\}, \{3.3185\times 10^{-15}, 0.\},\ \{-1.249\times 10^{-14}, 0.\}, \{-1.02562\times 10^{-12}, 1.11022\times 10^{-16}\}\}$$

```
B2 = Chop[dualInterpolationMD[F, P, 4, {x, y, z}, 1.*^-7], 1.*^-8]
```

» Initial Hilbert Function {1, 3, 5, 7, 6}

» Final Hilbert Function {1, 3, 5, 7, 6}

Out[144]=
$$\{1.\,x + 1.\,y - 1.\,x\,z + 1.\,y\,z, -2.\,x - 1.\,x^2\,y - 1.\,x\,y^2 - 1.\,y^3 + 2.\,x\,z + 1.\,x^3\,z,\ -2. + 3.\,x^2 + 4.\,x\,y + 2.\,y^2 - 2.\,x^2\,z + 2.\,z^2 + 1.\,x^2\,z^2, 1.\,x^4 + 4.\,x\,y + 1.\,y^4\}$$

Testing at random points is sufficient as above

In[164]:=
```
p = randomRealRegularPoints2D[f2, {{-2, 2}, {-2, 2}}, x, y , 1][[1]]
B2 /. Thread[{x, y, z} → Φ[p]]
B2 /. Thread[{x, y, z} → {0, 0, RandomReal[{-4, 4}]}]
```

Out[164]=
$$\{0.911205, -0.189496\}$$

Out[165]=
$$\{-2.62457\times 10^{-13}, 4.54738\times 10^{-10}, 4.12434\times 10^{-9}, 6.21173\times 10^{-12}\}$$

Out[166]=
$$\{0., 0., 12.3511, 0.\}$$

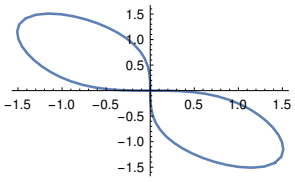
Thus the blowup contains the image of Φ but not other points on the exceptional line. We can plot our blow-up B.

In[168]:=
```
cpf2 = criticalPoints2D[f2, x, y]
```

Out[168]=
$$\{\{1.41421, -1.41421\}, \{-1.41421, 1.41421\},\ \{1.90519\times 10^{-175}, 1.24893\times 10^{-175}\}, \{0., 0.\}, \{0., 0.\}, \{0., 0.\}\}$$

In[169]:=
```
pth1 = Drop[pathFinder2D[f2, cpf2[[1]], {0, 0}, .15, x, y], −1];
pth2 = Reverse[Drop[pathFinder2D[−f2, cpf2[[1]], {0, 0}, .15, x, y], −1]];
pth3 = Reverse[Drop[pathFinder2D[f2, cpf2[[2]], {0, 0}, .15, x, y], −1]];
pth4 = Drop[pathFinder2D[−f2, cpf2[[2]], {0, 0}, .15, x, y], −1];
ListLinePlot[{Join[pth1, pth3, pth4, pth2]}, ImageSize → Small]
```

Out[173]=



Again we look at our exceptional line

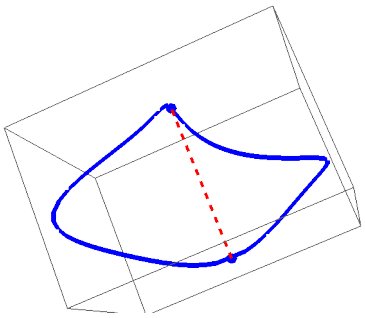
In[174]:=
```
excpt2 = fFiberMD[B2, {{1, 0, 0}, {0, 1, 0}}, {0, 0}, {x, y, z}, dTol]
```

» multiple fiber points {0, 0}

Out[174]= {{0., 0., −1.}, {0., 0., 1.}}

In[175]:=
```
Pth = Φ /@ Join[pth1, pth3, pth4, pth2];
Graphics3D[{{Blue, Thick, Line[Pth]},
    {Blue, PointSize[Large], Point[excpt2]}, {Red, Thick, Dashed, Line[excpt2]}}]
```

Out[176]=



3.3.2.3 The Bow Curve

In[178]:=
```
f3 = x^4 − x^2 y + y^3;
```

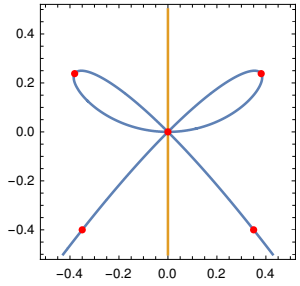
In[187]:=
```
cpf3 = DeleteDuplicates[Chop[criticalPoints2D[f3, x, y]]]
pts3 = {x, y} /. NSolve[{f3, y + .4}, {x, y}, Reals]
```

Out[187]= {{0.380892, 0.237985}, {−0.380892, 0.237985}, {0, 0}}

Out[188]= {{−0.349986, −0.4}, {0.349986, −0.4}}

In[192]:=
```
ContourPlot[{f3 == 0, x == 0}, {x, -.5, .5}, {y, -.5, .5}, MaxRecursion → 6,
  Epilog → {Red, PointSize[Medium], Point[Join[cpf3, pts3]]}, ImageSize → Small]
```

Out[192]=



In[191]:=
```
multiplicityMD[{f3, x}, {0, 0}, {x, y}, dTol]
```

Out[191]= 3

So x is a good denominator.

```
F3 = {f3, z x - y};
```

In[196]:=
```
Φ := Append[#, #[[2]]/#[[1]]] &
```

```
Θ := Take[#, 2] &
```

In[223]:=
```
L = randomRealRegularPoints2D[f3, {{-.5, .5}, {-.5, .5}}, x, y, 6]
P3 = Φ /@ L
```

Out[223]= {{−0.303216, −0.341592}, {0.226936, −0.249278}, {0.288911, 0.093154}, {−0.252178, −0.279407}, {−0.380898, 0.237976}, {0.312038, 0.111669}}

Out[224]= {{−0.303216, −0.341592, 1.12657}, {0.226936, −0.249278, −1.09845}, {0.288911, 0.093154, 0.322432}, {−0.252178, −0.279407, 1.10797}, {−0.380898, 0.237976, −0.624776}, {0.312038, 0.111669, 0.357871}}

In[225]:=
```
B3 = dualInterpolationMD[F3, P3, 6, {x, y, z}, 1.*^-8]
```

» Initial Hilbert Function {1, 3, 5, 4, 4, 4, 4}

» Final Hilbert Function {1, 3, 5, 4, 4, 4, 4}

Out[225]= $\{-1.\,y+1.\,x\,z,\ 1.\,x^3-1.\,x\,y+1.\,y^2\,z,\ 1.\,x^2-1.\,y+1.\,y\,z^2,\ 1.\,x-1.\,z+1.\,z^3,$
$1.\,x\,y-1.\,y\,z+1.\,y\,z^3,\ 1.\,y-1.\,z^2+1.\,z^4,\ 1.\,x^2-1.\,y+1.\,y^2+1.\,y\,z^4,$
$1.\,x-1.\,z+1.\,y\,z+1.\,z^5,\ -1.\,x^3+2.\,x\,y-1.\,y\,z+1.\,y\,z^5,\ -1.\,x^2+2.\,y-1.\,z^2+1.\,z^6\}$

In[226]:=
```
p = randomRealRegularPoints2D[f3, {{-10, 20}, {-20, 10}}, x, y, 1][[1]]
B3 /. Thread[{x, y, z} → Φ[p]]
```

Out[226]= {6.99065, −14.5823}

Out[227]= $\{1.06581\times10^{-14},\ -1.49726\times10^{-10},\ 5.4257\times10^{-11},\ -4.44835\times10^{-11},\ 1.28503\times10^{-9},$
$3.63109\times10^{-10},\ -6.06094\times10^{-9},\ -1.29319\times10^{-9},\ 1.39917\times10^{-8},\ 5.40972\times10^{-9}\}$

In[229]:=
```
excp3 = fFiberMD[B3, {{1, 0, 0}, {0, 1, 0}}, {0, 0}, {x, y, z}, 1.*^-9]
```

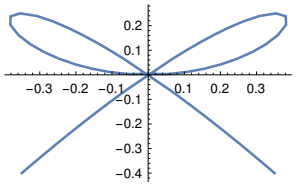
» multiple fiber points {0, 0}

Out[229]=
```
{{0., 0., 1.}, {0., 0., 0.}, {0., 0., -1.}}
```

So now we plot

In[242]:=
```
pth1 = pathFinder2D[f3, cpf3[[2]], {0, 0}, .05, x, y];
pth2 = pathFinder2D[-f3, cpf3[[2]], {0, 0}, .05, x, y];
pth3 = pathFinder2D[f3, cpf3[[1]], {0, 0}, .05, x, y];
pth4 = pathFinder2D[-f3, cpf3[[1]], {0, 0}, .05, x, y];
pth5 = pathFinder2D[-f3, pts3[[1]], {0, 0}, .05, x, y];
pth6 = pathFinder2D[f3, pts3[[2]], {0, 0}, .05, x, y];
ListLinePlot[Join[Drop[pth5, -1], Drop[Reverse[pth3], 1], Drop[pth4, -1],
    Drop[Reverse[pth1], 1], Drop[ pth2, -1], Drop[Reverse[pth6], 1]], ImageSize → Small]
```

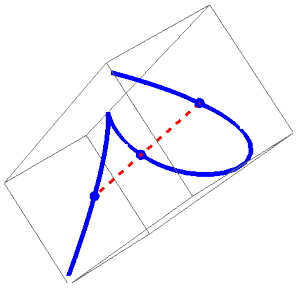
Out[247]=



In[248]:=
```
Pth = Φ /@ Join[Drop[pth5, -1], Drop[Reverse[pth3], 1], Drop[pth4, -1],
        Drop[Reverse[pth1], 1], Drop[ pth2, -1], Drop[Reverse[pth6], 1]];
```

In[251]:=
```
Graphics3D[{{Blue, Thick, Line[Pth]}, {Red, Thick, Dashed, Line[excp3]},
    {Blue, PointSize[Large], Point[excp3]}}, ImageSize → Small]
```

Out[251]=



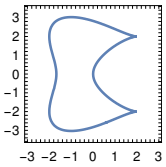
3.3.2.4 The Bicuspid

The bicuspid will present new challenges.

In[152]:=
```
f4 = 16 x - 4 x^3 + x^4 - 8 y^2 + y^4;
```

In[118]:=
```
ContourPlot[f4 == 0, {x, -3, 3}, {y, -3.5, 3.5}, ImageSize → Tiny]
```

Out[118]=



There are two cusps as singularities at $\{2, 2\}$ and $\{2, -2\}$. Our strategy will be to handle the two singularities simultaneously but separately in two new dimensions. To have denominators meet the singularity in a low multiplicity and miss the real part of the curve we construct the following lines

In[162]:=
```
l1 = line2D[{2, 2}, {3, 0}, x, y];
l1 = Expand[l1 / Coefficient[l1, y]]
```

Out[163]=
```
-6. + 2. x + 1. y
```

In[164]:=
```
l2 = line2D[{2, -2}, {3, 0}, x, y];
l2 = Expand[l2 / Coefficient[l2, y]]
```

Out[165]=
```
6. - 2. x + 1. y
```

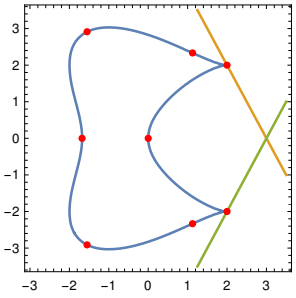
The critical points of the bicuspid are

In[144]:=
```
cpf4 = DeleteDuplicates[criticalPoints2D[f4, x, y]]
```

Out[144]=
```
{{-1.55139, -2.9125}, {2., 2.}, {1.12457, 2.33407},
 {-1.55139, 2.9125}, {1.12457, -2.33407}, {-1.67857, 0.}, {2., -2.}, {0., 0.}}
```

In[126]:=
```
ContourPlot[{f4 == 0, l1 == 0, l2 == 0}, {x, -3, 3.5}, {y, -3.5, 3.5},
 Epilog → {Red, PointSize[Medium], Point[cpf4]}, ImageSize → Small]
```

Out[126]=



We now define our blowup and rational functions

In[168]:=
```
F4 = {f4, z l1 - (x - y), w l2 - (x + y)}
```

Out[168]=
```
{16 x - 4 x^3 + x^4 - 8 y^2 + y^4, -x + y + (-6. + 2. x + 1. y) z, -x - y + w (6. - 2. x + 1. y)}
```

```
In[142]:= Φ := Join[#, {(#[[1]] - #[[2]])/(2 #[[1]] + #[[2]] - 6), (#[[1]] + #[[2]])/(-2 #[[1]] + #[[2]] + 6)}] &
          Θ := Take[#, 2] &
```

To check compatibility

```
In[166]:= p = randomRealRegularPoints2D[f4, {{-4, 4}, {4, 4}}, x, y, 1][[1]]
          F4 /. Thread[{x, y, z, w} → Φ[p]]
Out[166]= {-1.5978, 2.88581}

Out[167]= {-8.52814 × 10^-9, 0., 2.22045 × 10^-16}
```

We can calculate the exceptional curve by

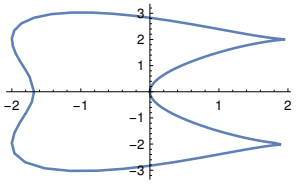
```
In[141]:= Chop[F4 /. Thread[{x, y, z, w} → {2, 2, z, w}]]
          F4 /. Thread[{x, y, z, w} → {2, -2, z, w}]
Out[141]= {0, 0, -4 + 4. w}

Out[142]= {0, -4 - 4. z, 0.}
```

Since these evaluations should give 0 on the curve the exceptional curve is the union of two lines in $\mathbb{R}^4$ given by $\{2, 2, z, 1\}$,and $\{2, -2, -1, w\}$ for param - eters z, w. Since we have cusps the actual blow-up without exceptional lines will meet the exceptional lines tangentially at one double point. We need to calculate these points but this will be hard as the equation of the exception free blow-up B4 will be a system of degree 6 in 4 variables which is beyond the capability of our **dualInterpolation** function.

But using our standard plotting method which involves path tracing f4 and lifting by Φ we can "plot" B4 in $\mathbb{R}^4$ by giving a large list of points. We can actually see the plot by projecting down on $\mathbb{R}^3$.

In[145]:=
```
pth1 = pathFinder2D[f4, {0, 0}, {2, 2}, .1, x, y, maxit → 40];
pth2 = pathFinder2D[–f4, {0, 0}, {2, –2}, .1, x, y, maxit → 40];
pth3 = pathFinder2D[–f4, cpf4[[3]], {2, 2}, .03, x, y, maxit → 40];
pth4 = pathFinder2D[f4, cpf4[[3]], cpf4[[6]], .3, x, y];
pth5 = pathFinder2D[f4, cpf4[[6]], cpf4[[5]], .4, x, y];
pth6 = pathFinder2D[f4, cpf4[[5]], {2, –2}, .07, x, y];
ListLinePlot[{Join[Drop[pth1, –1], Reverse[Drop[pth3, –1]], pth4,
      pth5, Drop[pth6, –1], Reverse[Drop[pth2, –1]]]}, ImageSize → Small]
```

Out[151]=



In[152]:=
```
pth = Join[Drop[pth1, –1], Reverse[Drop[pth3, –1]],
      pth4, pth5, Drop[pth6, –1], Reverse[Drop[pth2, –1]]];
Pth =
   Φ /@
     pth;
```

In[154]:=
```
Length[Pth]
```

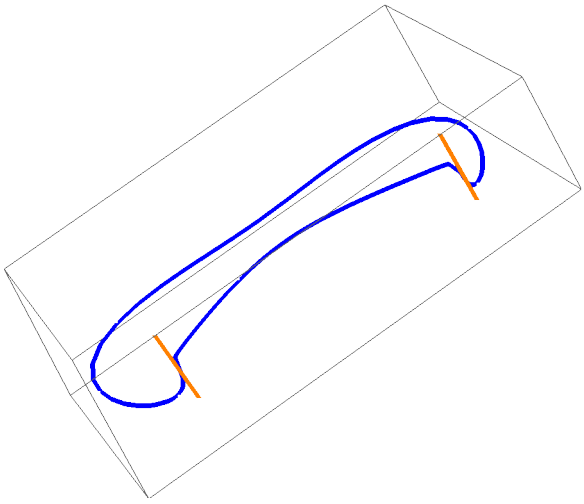
Out[154]= 138

To get an idea of what this looks like we can project down to $\mathbb{R}^3$. We can include the exceptional lines.

In[184]:=
```
proj4 = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, –1, 1}};
Pth3 = Pth.Transpose[proj4];
```

In[168]:=
```
Graphics3D[{{Blue, Thick, Line[Pth3]}, {Orange, Thick,
    Line[{{2, 2, 1}, {2, 2, 0}}], Line[{{2, −2, 0}, {2, −2, 1}}]}}, ImageSize → Medium]
```

Out[168]=



Our problem with **dualInterpolation** is two fold. First it will take far to long to run, the sizes of the matrices will be enormous. Second using only machine numbers these calculations will have small numerical errors, but using more precision will take even longer. We can somewhat fix the first problem is that most of the time will be used in the last step of finding the H-basis. Leaving out that step will give us a much quicker algorithm but the output will consist of a very large number of equations. But these should all, at least approximately, contain our B4. We use option **hBasis→False**

We choose 8 points

```
pts = RandomChoice[Pth, 8];
```

In[173]:=
```
pts
```

Out[173]=
```
{{1.12457, 2.33407, 0.853685, 0.568394}, {1.17864, 2.30806, 0.846226, 0.585924},
 {−1.67853, 2.82846, 0.690347, 0.0943692}, {−0.547515, 2.98335, 0.858741, 0.241689},
 {1.50526, −2.15674, −0.711592, −0.782327}, {1.50526, −2.15674, −0.711592, −0.782327},
 {0.719194, 1.26876, 0.166895, 0.340965}, {1.3413, 2.23097, 0.818889, 0.64384}}
```

In[161]:=
```
B4 = dualInterpolationMD[F4, pts, 6, {x, y, z, w}, 1.*^−8, hBasis → False]
```

Out[161]=
```
{−0.0110519+ ··· 323 ··· + 0.103655 y z^5 + 0.155079 z^6,
 ··· 176 ···, ··· 324 ··· + ··· 22 ··· ··· 1 ···}
```

large output | show less | show more | show all | set size limit...

There are 178 equations, each of which have 210 terms! So we will merely

sample B4. Our goal is to find where B4 intersects the exceptional lines. We are looking for multiple solutions. First we look at the line through $\{2, 2\}$.

In[189]:=
```
RandomChoice[Table[i, {i, 178}], 3]
```

Out[189]= {55, 128, 61}

In[190]:=
```
g55 = B4[[55]] /. {x → 2, y → 2, w → 1}
NSolve[g55]
```

Out[190]= $-0.195394 + 0.957736\,z - 1.56741\,z^2 + 1.03747\,z^3 - 0.351087\,z^4 + 0.00708735\,z^5 + 0.0265151\,z^6$

Out[191]= {{z → −5.0055}, {z → 0.500067 − 0.00212638 i}, {z → 0.500067 + 0.00212638 i},
{z → 0.888658 − 1.48755 i}, {z → 0.888658 + 1.48755 i}, {z → 1.96075}}

In[192]:=
```
g128 = B[[128]] /. {x → 2, y → 2, w → 1}
NSolve[g128]
```

Out[192]= $0.0708331 - 0.303968\,z + 0.340114\,z^2 + 0.00558316\,z^3 - 0.0424019\,z^4 - 0.0567799\,z^5 - 0.00900801\,z^6$

Out[193]= {{z → −5.69544}, {z → −1.32082 − 1.89438 i}, {z → −1.32082 + 1.89438 i},
{z → 0.500703 − 0.00700209 i}, {z → 0.500703 + 0.00700209 i}, {z → 1.03239}}

In[194]:=
```
g61 = B[[61]] /. {x → 2, y → 2, w → 1}
NSolve[g61]
```

Out[194]= $-0.0907389 + 0.331602\,z - 0.202743\,z^2 - 0.253104\,z^3 + 0.0994685\,z^4 + 0.0238999\,z^5 + 0.0193771\,z^6$

Out[195]= {{z → −1.31375}, {z → −1.20845 − 2.84028 i},
{z → −1.20845 + 2.84028 i}, {z → 0.496376}, {z → 0.50328}, {z → 1.49758}}

In each of these case there are two solutions, possibly complex, very close to $z = .5$ So we will suggest that $z = .5$ is at least a good approximation for the intersection of the exceptional line through $\{2, 2\}$ and B4. We do this again for $\{2, -2\}$

In[215]:=
```
RandomChoice[Table[i, {i, 178}], 3]
```

Out[215]= {130, 73, 50}

In[216]:=
```
g130 = B[[130]] /. {x → 2, y → −2, z → −1}
NSolve[g130]
```

Out[216]= $-0.589159 - 2.18272\,w - 1.91965\,w^2 + 0.112316\,w^3 - 0.190499\,w^4 + 0.0251769\,w^5 - 0.0164628\,w^6$

Out[217]= {{w → −1.01985 − 3.03538 i}, {w → −1.01985 + 3.03538 i}, {w → −0.515133 − 0.0412104 i},
{w → −0.515133 + 0.0412104 i}, {w → 2.29965 − 2.78939 i}, {w → 2.29965 + 2.78939 i}}

In[218]:=
```
g73 = B[[73]] /. {x → 2, y → -2, z → -1}
NSolve[g73]
```

Out[218]= $0.507827 + 1.23116\,w - 0.252052\,w^2 - 0.64533\,w^3 + 1.52396\,w^4 - 0.166313\,w^5 + 0.0242305\,w^6$

Out[219]= $\{\{w \to -0.527983 - 0.0480729\,i\}, \{w \to -0.527983 + 0.0480729\,i\}, \{w \to 0.76187 - 0.813969\,i\},$
$\{w \to 0.76187 + 0.813969\,i\}, \{w \to 3.19801 - 7.05408\,i\}, \{w \to 3.19801 + 7.05408\,i\}\}$

In[222]:=
```
g50 = B[[50]] /. {x → 2, y → -2, z → -1}
NSolve[g50]
```

Out[222]= $0.0140237 - 0.256541\,w - 0.491441\,w^2 +$
$0.0285561\,w^3 - 0.0145955\,w^4 + 0.0403071\,w^5 + 0.0151122\,w^6$

Out[223]= $\{\{w \to -3.64706\}, \{w \to -0.546267\}, \{w \to -0.287786 - 2.11328\,i\},$
$\{w \to -0.287786 + 2.11328\,i\}, \{w \to 0.049907\}, \{w \to 2.0518\}\}$

This is not so clear but it seems that we are getting solutions near $-.5$ This is somewhat consistent with the point {1.90006,-2.01554,-0.928878,-0.626432} which is seeming closest to the line $\{2, -2, -1, w\}$.

Unfortunately we are close to the limits of what we can do with our methodology.

3.3.2.5 A compound example

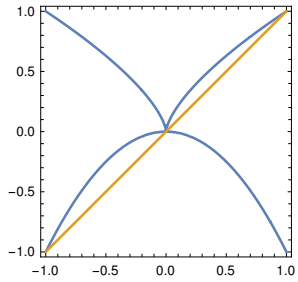
We consider the singularity at {0, 0} of

In[285]:=
```
f5 = Expand[(y^3 - x^2) (y + x^2)]
```

Out[285]= $-x^4 - x^2\,y + x^2\,y^3 + y^4$

In[834]:=
```
ContourPlot[{f == 0, x - y == 0}, {x, -1, 1}, {y, -1, 1}, MaxRecursion → 4, ImageSize → Small]
```

Out[834]=

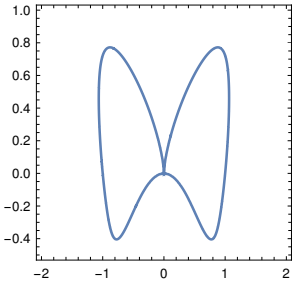


This is technically a reducible curve and we only discuss genus for irreducible curves, however we can still blow up. This will be essentially the singularity of a higher degree irreducible curve such as

In[849]:=
```
h = f5 + x^8 + y^8;
```

In[850]:=
```
ContourPlot[h == 0, {x, -2, 2}, {y, -.5, 1}, MaxRecursion → 4, ImageSize → Small]
```

Out[850]=



so it is worth studying this sort of singularity.

The multiplicity of our singularity of f5 is

In[851]:=
```
singPointMult2D[f5, {0, 0}, x, y, dTol]
```

From the first contour plot above we see the line $x - y$ is as good a choice as any but we we restrict our blow up to the region $-1 < x, y < 1$ because there will be infinite points above $\{-1, 1\}$ and $\{1, 1\}$. This has the right multiplicity.

In[853]:=
```
multiplicityMD[{f5, x - y}, {0, 0}, {x, y}, dTol]
```

Out[853]=
```
3
```

We obtain the equation of the blow-up

In[6]:=
```
F5 = {f5, z (x - y) - (x + y)}
```

Out[6]=
```
{-x^4 - x^2 y + x^2 y^3 + y^4, -x - y + (x - y) z}
```

dualInterpolation will work in default mode and degree 5 but needs a large set of random points not near the origin. But it returns a large system even after reducing to something like a H-basis. We throw out most of the equations to get a reasonable basis for the blow-up.

In[18]:=
```
B5 = {-x^4 - x^2 y + x^2 y^3 + y^4, -x - y + (x - y) z,
      1 - 8 x - x^2 - 8 y + z + 8 x z + 3 x^2 z - z^2 - 8 x z^2 - 3 x^2 z^2 - z^3 + x^2 z^3,
      1 - 8 x + 4 x^2 - 8 y + 6 x y + y^2 + z + 8 x z - 5 x^2 z - z^2 - 8 x z^2 + x^2 z^2 - z^3 + y^2 z^3};
```

We then calculate where the blow-up hits the exceptional line for F.

In[863]:=
```
Bo = B5 /. {x → 0, y → 0}
NSolve[Bo]
```

Out[863]=
```
{0, 0, 1 + z - z^2 - z^3, 1 + z - z^2 - z^3}
```

Out[864]=
```
{{z → -1.}, {z → -1.}, {z → 1.}}
```

These intersections are at $\{0, 0, \pm 1\}$. Note these points are regular.

In[865]:= **tangentVectorMD[B5, {0, 0, 1}, {x, y, z}]**

» Hilbert Function {1, 1, 1, 1, 1}

Out[865]= {0.447214, 0., −0.894427}

This is otherwise known as {1,0,-2}.

In[135]:= **tangentVectorMD[B5, {0, 0, −1}, {x, y, z}]**

» Hilbert Function {1, 1, 1, 1, 1}

Out[135]= {0., 0., 1.}

We plot the blow up using our rational function and the fact that both components are parametric curves:

In[2]:= **Φ := Append[#, (#[[1]] + #[[2]]) / (#[[1]] − #[[2]])] &**

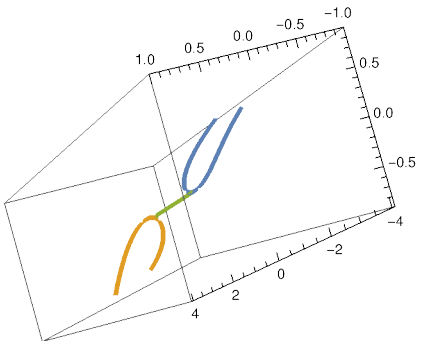
Note that

In[14]:= **F5 /. Thread[{x, y, z} → Φ[{t^3, t^2}]]**
F5 /. Thread[{x, y, z} → Φ[{t, −t^2}]]

Out[14]= {0, 0}

Out[15]= {0, 0}

In[16]:= **ParametricPlot3D[{Φ[{t^3, t^2}], Φ[{t, −t^2}], {0, 0, t}}, {t, −.9, .9}]**



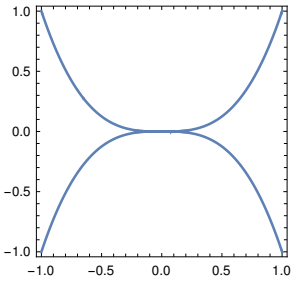
3.3.2.5 A harder compound example

Our final example is

In[125]:= **f6 = y^2 − x^6;**

In[204]:=
```
ContourPlot[f6 == 0, {x, -1, 1}, {y, -1, 1}, MaxRecursion → 4, ImageSize → Small]
```

Out[204]=



We see the multiplicity is smaller

```
l = RandomReal[{-2, 2}, 2].{x, y}
```

In[205]:=
```
multiplicityMD[{f6, l}, {0, 0}, {x, y}, dTol]
```

Out[205]= 2

We will not actually try to find the blow-up but just look at the plots. First

In[126]:=
```
F6 = {f6, z x - y};
Φ := Append[#, #[[2]] / #[[1]]] &;
```

Note that

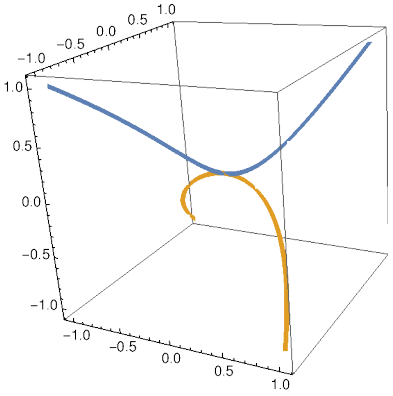
In[208]:=
```
F6 /. Thread[{x, y, z} → Φ[{t, t^3}]]
F6 /. Thread[{x, y, z} → Φ[{t, -t^3}]]
```

Out[208]= {0, 0}

Out[209]= {0, 0}

In[210]:=
```
ParametricPlot3D[{Φ[{t, t^3}], Φ[{t, -t^3}]}, {t, -1, 1}]
```

Out[210]=



We see that we still have a singularity at $\{0, 0, 0\}$ over $\{0, 0\}$. In fact we can generalize the multiplicity of a singularity to higher dimension

In[211]:=
```
pl = RandomReal[{-1, 1}, 3].{x, y, z}
```

Out[211]=
$$0.385291\,x + 0.571885\,y - 0.097667\,z$$

In[212]:=
```
multiplicityMD[Append[F, pl], {0, 0, 0}, {x, y, z}, 1.*^-10]
```

Out[212]= 4

So our singularity is actually worse in some sense. So we blow this up.

In[128]:=
```
G6 = Append[F6, w x - (x - y + z)]
Λ := Append[#, (#[[1]] - #[[2]] + #[[3]]) / (#[[1]])] &
```

Out[128]=
$$\{-x^6 + y^2, -y + x\,z, -x + w\,x + y - z\}$$

In[130]:=
```
G6 /. Thread[{x, y, z, w} → Λ[Φ[{t, t^3}]]]
G6 /. Thread[{x, y, z, w} → Λ[Φ[{t, -t^3}]]]
```

Out[130]= {0, 0, 0}

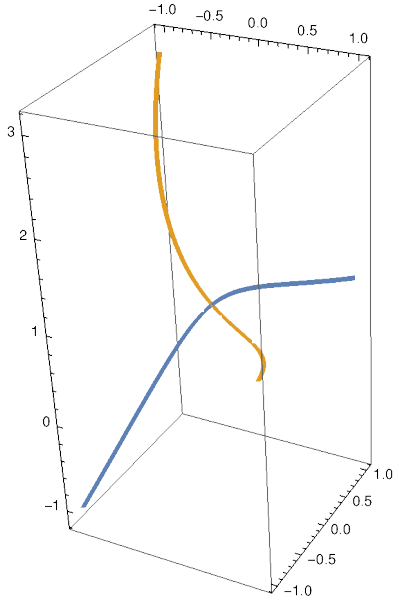
Out[131]= {0, 0, 0}

To plot we project back to $\mathbb{R}^3$

In[118]:=
```
Λ3 := (Λ[Φ[#]].{{1, 0, 0}, {0, 1, 0}, {0, 0, 0}, {0, 0, 1}}) &
```

In[226]:=
```
ParametricPlot3D[{Λ3[{t, t^3}], Λ3[{t, -t^3}]}, {t, -1, 1}]
```

Out[226]=



Our singularity looks better but is still there over {0, 0}. In fact looking at the vertical scale in this plot we can guess correctly that the singularity is actually at {0, 0, 0, 1} in $\mathbb{R}^4$.

```
In[243]:= hp4 = RandomReal[{-1, 1}, 4].{x, y, z, w - 1}
          multiplicityMD[Append[G, hp4], {0, 0, 0, 1}, {x, y, z, w}, 1.*^-9]
```

Out[243]= $-0.628167\,(-1+w)-0.441975\,x+0.453086\,y-0.545897\,z$

Out[244]= 6

So we blow up once more

```
In[132]:= H6 = Append[G, u (w - 1) - x]
          Γ := Append[#, #[[1]] / (#[[4]] - 1)] &
```

Out[132]= $\{-x^6+y^2,\ -y+x\,z,\ -x+w\,x+y-z,\ u\,(-1+w)-x\}$

```
In[134]:= H6 /. Thread[{x, y, z, w, u} → Γ[Λ[Φ[{t, t^3}]]]]
          H6 /. Thread[{x, y, z, w, u} → Γ[Λ[Φ[{t, -t^3}]]]]
```

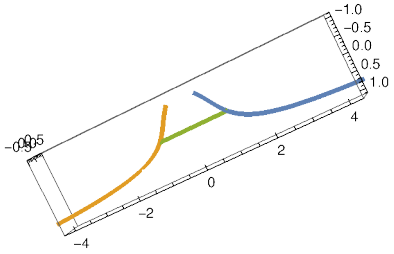
Out[134]= {0, 0, 0, 0}

Out[135]= {0, 0, 0, 0}

So H6 is compatible with the composition $\Gamma[\Lambda[\Phi[\#]]]$. Now project

```
In[137]:= Γ3 := (Γ[Λ[Φ[#]]].{{1, 0, 0}, {0, 1, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 1}}) &
In[294]:= ParametricPlot3D[{Γ3[{t, t^3}], Γ3[{t, -t^3}], {0, 0, t}}, {t, -1, 1}]
```

Out[294]=



Where the green segment is again the exceptional line over $\{0, 0\}$.

So this takes 3 blow-ups to accomplish the job.

3.3.2.7 Conclusion on blowing-up

We have seen that *given a square free algebraic plane curve f with only affine singularities we can find, by a sequence of blowing up, a non-singular algebraic curve F in $\mathbb{R}^n$ for some n that projects to f using the projection taking a point $\{x_1, x_2, \ldots, x_n\}$ to $\{x_1, x_2\}$.*

We should compare this with Abhyankar's *Theorem of resolution of singularities of plane curves* in Lecture 18 of his book. Our theorem is a little more explicit than his as it actually produces such a curve with no exceptional lines and projecting on the first two coordinates. We also explicitly give the

equation of this plane curve, at least in theory, and the rational function from f to F.

Of course we already saw in Chapter 6 of the Plane Curve Book that given any plane curve we can move all the projective singularities to the affine plane so the requirement that all singularities be affine is not really a restriction.

An important point about blowing-up is that it is numerically stable. We saw in the examples of this subsection that choice of the linear function in the denominator has few restrictions, only that the multiplicity at the point of the intersection of the denominator with the curve is the multiplicity of the singularity. Since this multiplicity is numerically stable under small perturbations even a slight error in identifying the singularity will not materially effect the blow-up.

3.3.3 Genus of curves

Barry Mazur, in his famous 1986 paper *Arithmetic on Curves* (Reprinted in the AMS Bulletin, Vol 55, No.3, July 2018) states on page 219

[A non-singular space curve] under a generic projection to a 2-dimensional projective space yields a plane curve with at worst nodal (or ordinary double point) singularities.

This is not quite right. At the end of section 2.4 I give the correct version

Under any projection of a space curve to the plane, the projection of a singular point will still be singular. For generic projections, with high probability, the only artifactual singularities will be normal crossings (nodes), cusps or isolated points.

Recall that artifactual singularities are those that do not come from singularities of the original space curve, so for a non-singular space curve all singularities of the projection are artifactual. In the generic case artifactual singular points are double points, they have multiplicity 2. Nodes are ordinary, in the sense of Section 3.4 of my Plane Curve Book, cusps and isolated points (which arise only in the real case) are not. But these do still have Clebsch number 1, the same as ordinary double points, so Mazur's formula below still works. Note that Example 3.3.2.6 is a double point but not a node or cusp.

Mazur's Formula: [Mazur page 220] *Let ν be the number of singular points of a generic (random) projection of a non-singular space curve. Then the genus g of the space curve and its plane projection of degree d is given by*

$$g = \frac{(d-1)(d-2)}{2} - \nu$$

We can use this formula to calculate the genus. But note that this should not be taken as a definition of *genus* but the consequence of the formal study of genus by algebraic geometers.

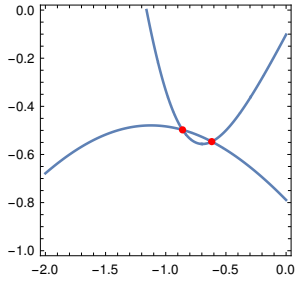
Example 3.3.3.1: A nice example is the bow curve 3.3.2.3. We found the non-singular blow-up to be curve

```
In[188]:= B3 = {-1. y + 1. x z, 1. x^3 - 1. x y + 1. y^2 z, 1. x^2 - 1. y + 1. y z^2, 1. x - 1. z + 1. z^3,
              1. x y - 1. y z + 1. y z^3, 1. y - 1. z^2 + 1. z^4, 1. x^2 - 1. y + 1. y^2 + 1. y z^4,
              1. x - 1. z + 1. y z + 1. z^5, -1. x^3 + 2. x y - 1. y z + 1. y z^5, -1. x^2 + 2. y - 1. z^2 + 1. z^6};

In[123]:= bbc = FLTMD[B3, A, 6, {x, y, z}, {x, y}, 1.*^-9][[1]]

Out[123]= 1. - 7.64881 x - 14.4732 x^2 - 9.41023 x^3 - 2.81002 x^4 + 11.0999 y + 4.72927 x y - 0.912787 x^2 y +
          1.42335 x^3 y + 12.5987 y^2 + 10.9587 x y^2 + 3.71029 x^2 y^2 + 0.216945 y^3 - 0.478956 x y^3 + 0.0523491 y^4

In[138]:= csp = complexProjectiveSingularPoints2D [bbc, x, y, 1.*^-9]

Out[138]= {{-1.98573, -11.9106}, {-0.617785, -0.546592}, {-0.860395, -0.497789}}
```

Take a generic projection from $\mathbb{P}^3$ to $\mathbb{P}^1$

```
In[121]:= A = Orthogonalize[RandomReal[{-1, 1}, {3, 4}]]

Out[121]= {{0.084272, 0.846137, 0.364511, 0.379582},
           {-0.591153, 0.464949, -0.517101, -0.408617}, {-0.632973, -0.163991, 0.730846, -0.195745}}

In[146]:= bbc = FLTMD[B3, A, 6, {x, y, z}, {x, y}, 1.*^-9][[1]]

Out[146]= 1. - 7.64881 x - 14.4732 x^2 - 9.41023 x^3 - 2.81002 x^4 + 11.0999 y + 4.72927 x y - 0.912787 x^2 y +
          1.42335 x^3 y + 12.5987 y^2 + 10.9587 x y^2 + 3.71029 x^2 y^2 + 0.216945 y^3 - 0.478956 x y^3 + 0.0523491 y^4

In[143]:= csp = complexProjectiveSingularPoints2D [bbc, x, y, 1.*^-8]

Out[143]= {{-0.860395, -0.497789}, {-0.617785, -0.546592}, {-1.98573, -11.9106}}

In[151]:= ContourPlot[bbc == 0, {x, -2, 0}, {y, -1, 0}, MaxRecursion → 5,
            Epilog → {Red, PointSize[Medium], Point[Take[csp, 2]]}, ImageSize → Small]
```

Out[151]=

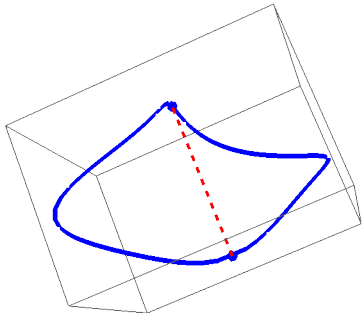


The third singular point {-1.98573, -11.9106} is an isolated singularity. Since $\nu = 3$ and $d = 4$ then $g = 0$ which is what we expect given this is a parameterized curve.

Example 3.3.3.2: The lemniscate.

We calculated the blow-up of the lemniscate as

In[292]:=
```
B2 = {1. x + 1. y − 1. x z + 1. y z, −2. x − 1. x ^ 2 y − 1. x y ^ 2 − 1. y ^ 3 + 2. x z + 1. x ^ 3 z,
    −2. + 3. x ^ 2 + 4. x y + 2. y ^ 2 − 2. x ^ 2 z + 2. z ^ 2 + 1. x ^ 2 z ^ 2, 1. x ^ 4 + 4. x y + 1. y ^ 4};
```



Since the lemniscate is a bounded we let the random projection be

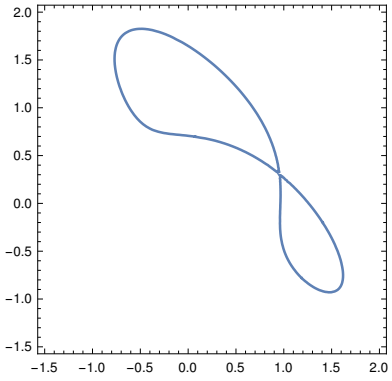
In[312]:=
```
A2 = Append [Orthogonalize [RandomReal [{−1, 1}, {2, 4}]], {0, 0, 0, 1}]
h2 = FLTMD [B2, A2, 6, {x, y, z}, {x, y}, 1.*^−7 ][[1]]
```

Out[312]=
```
{{0.0648747 , −0.730778 , 0.479363 , 0.481628 },
  {−0.844846 , 0.0847465 , −0.232867 , 0.474159 }, {0, 0, 0, 1}}
```

Out[313]=
$$1. + 0.809265\ x - 1.59889\ x^2 - 2.62573\ x^3 + 2.89494\ x^4 - 0.69337\ x^5 + 0.169007\ x^6 - 0.662574\ y - 1.32448\ x\,y + 1.10846\ x^2\,y + 1.91681\ x^3\,y - 1.61182\ x^4\,y + 0.748308\ x^5\,y - 0.00606811\ y^2 - 0.0856423\ x\,y^2 - 1.72749\ x^2\,y^2 + 0.734084\ x^3\,y^2 + 0.945916\ x^4\,y^2 - 2.06329\ y^3 - 1.4528\ x\,y^3 + 2.94894\ x^2\,y^3 + 0.322685\ x^3\,y^3 + 0.925493\ y^4 + 1.1306\ x\,y^4 + 0.334215\ x^2\,y^4 - 0.452467\ y^5 + 0.576751\ x\,y^5 + 0.400899\ y^6$$

In[316]:=
```
ContourPlot[h2 == 0, {x, −1.5, 2}, {y, −1.5, 2}]
```

Out[316]=



For finding all singular points we find a very large tolerance works best, although it is recommended that this be checked carefully.

In[317]:=
```
csp = complexProjectiveSingularPoints2D  [h2, x, y, .01]
```

Out[317]=
```
{{-0.432648 + 0.415856 i, -0.643362 + 0.829752 i},
  {-0.432648 - 0.415856 i, -0.643362 - 0.829752 i}, {-1.23861 - 0.922593 i, 1.04178 + 1.87947 i},
  {-1.23861 + 0.922593 i, 1.04178 - 1.87947 i}, {2.41939 + 1.30732 i, -0.394761 - 2.28109 i},
  {2.41939 - 1.30732 i, -0.394761 + 2.28109 i}, {0.954581 , 0.306315 }, {1., -0.485784 , 0}}
```

In[318]:=
```
Length [csp]
```

Out[318]=
```
8
```

So we have 6 complex singular points, one real affine singular point and one affine infinite singular points. Since the degree of the projection is 6 Mazur's formula gives $g = 10 - 8 = 2$. Note that it is impossible for a non-singular plane curve to have genus 2.

3.3.3.3 Example. This example is different in that we start with a non-singular space curve and don't blow up. The example is a case of Exercise IV 5.2.2 from Hartshorne's *Algebraic Geometry* book.

We take a naive intersection of a quadric and cubic surface in $\mathbb{R}^3$.

In[288]:=
```
f1 = x^2 + y^2 + z^2 - 25;
f2 = -51 + 3 x - 3 x^2 + x^3 - 3 y - 3 y^2 - y^3 + 14 z - z^2;
```

We will use a random affine projection

In[291]:=
```
A = {{-0.163999, 0.250186, -0.294883, -0.138623},
     {-0.609386, 0.427477, -0.396493, -0.530766}, {0, 0, 0, 1}};
```

In[292]:=
```
g4 = FLTMD[{f1, f2}, A, 6, {x, y, z}, {x, y}, 1.*^-10][[1]]
```

Out[292]=
```
1. + 1.73533 x + 1.55993 x^2 - 0.656023 x^3 - 2.60795 x^4 - 2.7081 x^5 + 2.70678 x^6 -
  0.00233382 y - 1.17272 x y + 0.266015 x^2 y + 4.17205 x^3 y + 7.69862 x^4 y -
  8.26483 x^5 y + 0.365579 y^2 - 0.122065 x y^2 - 2.71565 x^2 y^2 - 8.65234 x^3 y^2 +
  10.9325 x^4 y^2 + 0.0466165 y^3 + 0.792389 x y^3 + 4.87301 x^2 y^3 - 7.91459 x^3 y^3 -
  0.0857802 y^4 - 1.37971 x y^4 + 3.2871 x^2 y^4 + 0.156806 y^5 - 0.739525 x y^5 + 0.0701592 y^6
```

As usual we need to fiddle with complexProjectiveSingularPoints to get a reliable answer but we come up with one answer we can verify

```
{{1.289, 0.0206694}, {3.56225, 7.6008}, {-1.54168 + 0.346882 I, -2.16869 + 0.495606 I},
  {-1.54168 - 0.346882 I, -2.16869 - 0.495606 I},
  {-1.74148, -3.95367}, {-2.04685, -4.49551}}
```

Since g4 is of degree 6 Mazur's formula shows the genus $g = 10 - 6 = 4$ agreeing with Hartshorne's claim. Note that the first two real singular points are actually isolated points from the projection.

3.3.4 Examples of non-singular Curves of genus 0 - 6

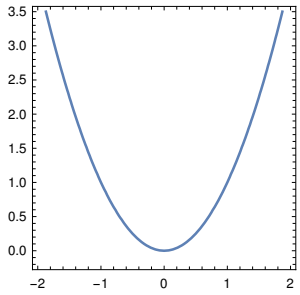
Now that we have developed our software and theory I end by plotting an example of a curve of each genus from 0 to 6. We don't show work but we

use the methods we have developed. Some of these examples have appeared before in this book or my plane curve book. We give a plane model on the left and, where the plane model is singular, a non-singular model in $\mathbb{R}^3$ on the right.

Genus 0, Rational curve, parabola $y = x^2$

In[128]:=
```
ContourPlot[y == x^2, {x, -2, 2}, {y, -.2, 3.5}, ImageSize → Small]
```

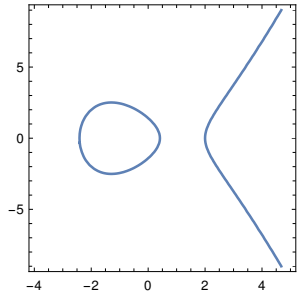
Out[128]=



Genus 1, Elliptic curve $y^2 = x^3 - 5x + 2$

In[126]:=
```
ContourPlot[y^2 == x^3 - 5 x + 2, {x, -4, 5}, {y, -9, 9}, ImageSize → Small]
```

Out[126]=

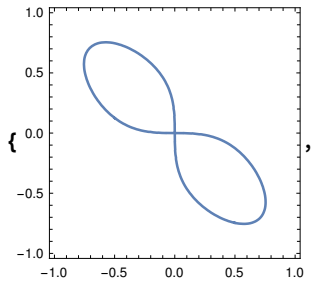


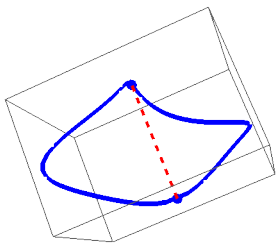
Genus 2, Lemniscate $x^4 + xy + y^4$

In[175]:=
```
ContourPlot[x^4 + x y + y^4 == 0, {x, -1, 1}, {y, -1, 1}, ImageSize → Small]
```

(Red dashed line is exceptional line over {0,0})

Out[175]=

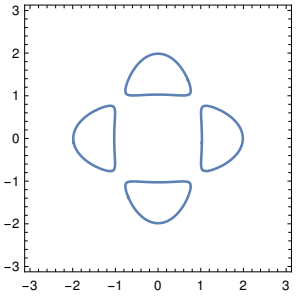
{ 

,  }

Genus 3, Klein Curve $\left(x^2 + \frac{y^2}{4} - 1\right)\left(\frac{x^2}{4} + y^2\right) = -.04$

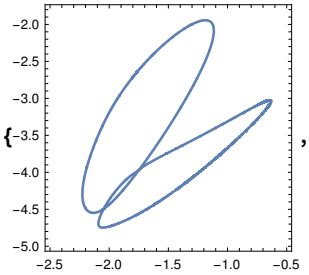
In[122]:=
```
ContourPlot[(x^2+y^2/4-1) (x^2/4+y^2-1) == -.04,
  {x, -3, 3}, {y, -3, 3}, ImageSize→ Small]
```

Out[122]=

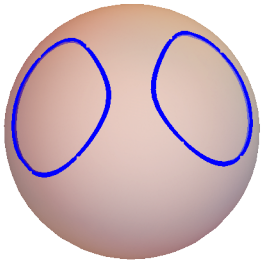


Genus 4 (See Example 3.3.3.3) g4 on the left, {f1, f2} plotted on f1 on the right.

{

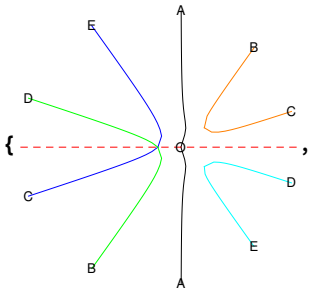


,

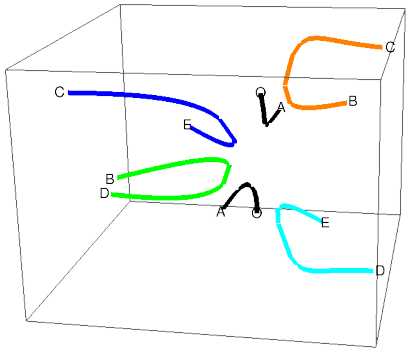


}

Genus 5: Gauss' curve $g5 = -5x^2 + 9x^3 - 5x^4 + x^5 + 5y^2 - 27xy^2 + 30x^2y^2 - 10x^3y^2 - 5y^4 + 5xy^4$
(Dashed red line is blowing-up denominator, A,B,C,D,E,0 infinite points.)



,



}

Genus 6

In[173]:= `g6 = 1 - 10 x^2 + 5 x^4 - 3 y + 18 x^2 y - 3 x^4 y - 5 y^2 + 15 x^2 y^2 + 15 y^3 - 15 x^2 y^3 + 4 y^4 - 12 y^5;`

In[174]:= `ContourPlot[g6 == 0, {x, -5, 5}, {y, -2, 3}, ImageSize → Small]`

Out[174]=