9/2020

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

A Numerical Approach to Real Algebraic Curves

with the Wolfram Language, Part II Space Curves.

https: // 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 and Appendix I on numerical linear algebra or the Mathematica Journal article and prior familiarity with numerical linear algebra. All the code is in the Mathematica notebook GlobalFunctionsMD.nb available at my website listed above.

Table of Contents

We recommend that the reader be familiar with our book *A Numerical Approach to Real Algebraic Curves with the Wolfram Language,* henceforth known as "my Plane Curve book", or at least with the *Mathematica Journal* summary of this book (2018). And the reader should have some familiarity with the Wolfram Language.

Note the naming conventions: All global functions defined in this Space Curve Book begin with a lowercase letter, compound names will capitalize first letters of subsequent words, (camel casing). This avoids confusion with built in Mathematica functions. Also functions with polynomial and/or point arguments will end in 2D, 3D, or MD depending on whether they work in 2,3 or all dimensions. This makes clear what the arguments are and distinguishes these functions from my Plane Curve functions so both sets can be initialized together without conflict, however most plane curve functions that you may need are contained here with 2D designation. Note that functions with suffix 3D or MD take variables as a list, but members of the list should be atomic variables, e.g. not X[[2]] but possibly x[2].

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

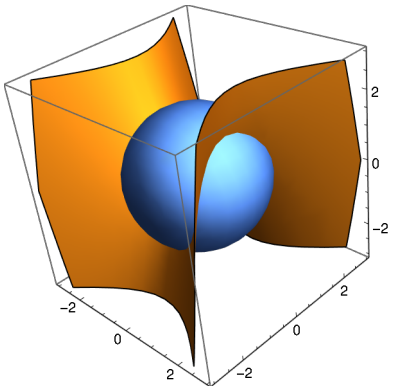
1.1 Emulating Plane Curves

As a seemingly simple example consider the curve produced by intersecting a hyperboloid and an ellipsoid.

1.1.1 Example

In[239]:=
```
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[239]=



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

In[148]:=
```
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[148]= {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[244]:=
```
nv1 = Grad[f11, {x, y, z}] /. Thread[{x, y, z} → p1]
nv2 = Grad[f12, {x, y, z}] /. Thread[{x, y, z} → p1]
```

Out[244]= {2.30826, −2.89231, −1}

Out[245]= {2.30826, 2.89231, −1.5187}

The tangent vector is simply the cross product

In[246]:=
```
tv1 = Cross[nv1, nv2]
```

Out[246]= {7.28487, 1.19729, 13.3524}

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

In[6]:=
```
tangentVector3D[{f_, g_}, p_, {x_, y_, z_}] := Module[{n1, n2, bi},
    If[Norm[{f, g} /. Thread[{x, y, z} → p]] > 1.*^-8, Echo[p, "not a point "];
      Return[Fail]];
    n1 = {D[f, x], D[f, y], D[f, z]} /. Thread[{x, y, z} → p];
    n2 = {D[g, x], D[g, y], D[g, z]} /. Thread[{x, y, z} → p];
    bi = N[Cross[n1, n2]];
    If[Norm[bi] < .0001, Echo[p, "No tangent vector at "]; bi, Normalize[bi]]]
```

In[152]:=
```
tangentVector3D[{f11, f12}, p1, {x, y, z}]
```

Out[152]= {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 definition that will be discussed later.

In this naive case we can get critical points just as for plane curves.

In[8]:=
```
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[9]:=
```
critpts = criticalPoints3D[{f11, f12}, {x, y, z}]
```

Out[9]= {{1.45718, -1.25159, 0.556915}, {1.24962, 0., 1.56155}, {0., 1.24962, -1.56155},
{-1.45718, 1.25159, 0.556915}, {1.45718, 1.25159, 0.556915},
{0., -1.24962, -1.56155}, {-1.24962, 0., 1.56155}, {-1.45718, -1.25159, 0.556915}}

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.

```
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, N[Det[J]]}, {x, y, z}, Reals];
    MinimalBy[sol, Norm[# - p] &][[1]]
  ]
```

There may be infinitely many closest points.

In[128]:=
```
p2 = closestPoint3D[{f11, f12}, {1, 1, 1}, {x, y, z}]
```

Out[128]= {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. We use an analog of pathFinderT from my Plane Curve book.

In our code p, q will be the start and end points of the path and s will be the desired step size. One may choose this by trial.

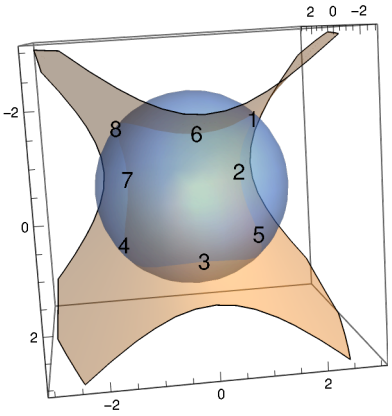
In[20]:=
```
Options[pathFinder3D] = {maxit → 30};
pathFinder3D[{f_, g_}, p_, q_, s_, {x_, y_, z_}, OptionsPattern[]] :=
  Module[{k, p0, p1, tv1, tv, L},
    p0 = p;
    L = Reap[Sow[p];
      k = 0;
      While[Norm[q – p0] > 2 s && k < OptionValue[maxit],
        tv1 = tangentVector3D[{f, g}, p0, {x, y, z}];
        If[tv1.(q – p0) > 0, tv = tv1, tv = –tv1];
        p0 = closestPoint3D[{f, g}, p0 + s * tv, {x, y, z}];
        Sow[p0];
        k++];
      If[k ≥ OptionValue[maxit], Print["Warning, iteration limit reached"]];
      Sow[q]];
    L[[2, 1]];
```

The reader is cautioned that in $\mathbb{R}^3$ we don't have a canonical direction of travel on curves, unlike $\mathbb{R}^2$. Therefore tracing in $\mathbb{R}^3$ is somewhat different. This tracing function takes what appears to be the shortest Euclidean distance to the end point. If the intended path does not go directly to the desired end the trace may fail, so one should trace short or relatively straight paths only. Also replacing the order of $\{f, g\}$ or their signs makes no difference. In particular tracing around a closed bounded component requires at least 3 paths. Finally, in the unlikely event of a singular point then you can trace into this point, but not out. By default the procedure will stop after 30 steps, this can be changed to a different number n by the option maxit→n. If the maximum number of iterations is reached, the path will jump to the indicated end point as in the plane case.

In Example 1.1.1 we plot

In[144]:=
```
Show[ContourPlot3D [{f11 == 0, f12 == 0}, {x, −3, 3},
  {y, −3, 3}, {z, −3, 3}, Mesh → None, ContourStyle → Opacity[0.4]],
 Graphics3D [Table[{Text[Style[i, FontSize → 14], critpts⟦i⟧]}, {i, 8}]]]
```

Out[144]=



This shows that our curve will be closed and bounded, in principle we can have a path from any critical point to any other. But applying pathFinder3D to get from critical point 4 to critical point 6 we get

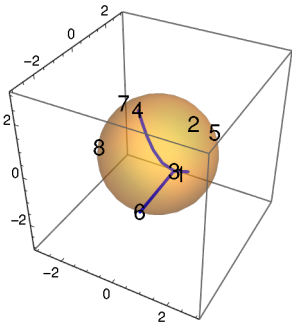
In[192]:=
```
pth = pathFinder3D [{f11, f12}, critpts⟦4⟧, critpts⟦6⟧, .6, {x, y, z}, maxit → 15]
```

Warning, iteration limit reached

Out[192]=
```
{{−1.45718, 1.25159, 0.556915}, {−1.41441, 1.41401, 0.00113412},
 {−1.25246, 1.45722, −0.554849}, {−0.960051, 1.40464, −1.05132}, {−0.535693, 1.30548, −1.41731},
 {−0.0167815, 1.24968, −1.56142}, {0.502491, 1.29911, −1.4352}, {−0.0196231, 1.2497, −1.56137},
 {0.499898, 1.29863, −1.43654}, {−0.0224445, 1.24972, −1.56131}, {0.49732, 1.29815, −1.43787},
 {−0.0252454, 1.24975, −1.56124}, {0.494759, 1.29768, −1.43918}, {−0.0280255, 1.24978, −1.56117},
 {0.492214, 1.29721, −1.44048}, {−0.0307847, 1.24982, −1.56109}, {0., −1.24962, −1.56155}}
```

In[193]:=
```
Show[ContourPlot3D[{f12 == 0}, {x, −3, 3},
  {y, −3, 3}, {z, −3, 3}, Mesh → None, ContourStyle → Opacity[0.4]],
 Graphics3D[{Table[{Text[Style[i, FontSize → 14], critpts⟦i⟧]}, {i, 8}], {Blue, Thick, Line[pth]}}],
 ImageSize → Small]
```

Out[193]=



In this attempt we find that the tracing gets hung up at critical point 3 and doesn't know how to get to point 6 from there.

We could however find intermediate points and do

In[249]:=
```
pth1 = pathFinder3D [{f11, f12}, critpts⟦4⟧, critpts⟦3⟧, .3, {x, y, z}];
pth2 = pathFinder3D [{f11, f12}, critpts⟦3⟧, critpts⟦5⟧, .3, {x, y, z}];
pth3 = pathFinder3D [{f11, f12}, critpts⟦5⟧, critpts⟦1⟧, .3, {x, y, z}];
pth4 = pathFinder3D [{f11, f12}, critpts⟦1⟧, critpts⟦6⟧, .3, {x, y, z}];
```

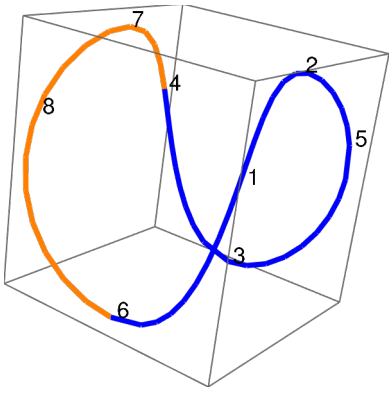
Or, if we are only interested in getting from 4 to 6 we could simply do

In[257]:=
```
pth5 = pathFinder3D [{f11, f12}, critpts⟦6⟧, critpts⟦4⟧, .4, {x, y, z}];
```

But by now we have gone all around the curve so we can plot the curve only

In[190]:=
```
Graphics3D [{{Blue, Thick, Line[{pth1, pth2, pth3, pth4}]}, {Orange, Thick, Line[pth5]},
    Table[{Text[Style[i, FontSize → 14], critpts⟦i⟧ + {.1, .1, .1}]}, {i, 8}]}]
```

Out[190]=



As with plane curves we can find infinite points of space curves. We need forms which can just as easily be defined in any number of variables.

```
formMD[f_, k_, X_] :=
    FromCoefficientRules[Select[CoefficientRules[f, X], Total[#⟦1⟧] == k &], X];
maxFormMD[f_, X_] := formMD[f, tDegMD[f, X], X];
```

```
infiniteRealPoints3D[{f_, g_}, {x_, y_, z_}] := Module[{sol},
    sol = {x, y, z} /. NSolve[{maxFormMD[f, {x, y, z}],
            maxFormMD[g, {x, y, z}], x^2 + y^2 + z^2 - 1}, {x, y, z}, Reals];
    Append[#, 0] &/@ Tally[sol, Norm[#1 - #2] < .0001 &][All, 1]]
```

1.1.2 Our simple example is

In[233]:=
```
F2 = {x^2 - y^2 - 1, x + y + z - 1};
infiniteRealPoints3D [F2, {x, y, z}]
```

Out[234]=
```
{{-0.707107 , 0.707107 , 0., 0}, {-0.408248 , -0.408248 , 0.816497 , 0}}
```

1.2 Projection

1.2.1 Linear 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.

Example 1.2.1.1: Projection Pxy

In[269]:=
```
Pxy = {{1, 0, 0}, {0, 1, 0}};
```

Given a point, say $p = \{1, 2, 3\}$, in $\mathbb{R}^3$ we can project it onto $\mathbb{R}^2$ by

In[271]:=
```
p = {1, 2, 3};
Pxy.p
```

Out[272]=
```
{1, 2}
```

Here Mathematica treats, by context, p as a column vector, that is, takes its transpose. But typically we have a list of points, for instance

In[274]:=
```
pts = {{1, 2, 3}, {0, 1, 4}, {0, 0, 3}}
```

Out[274]=
```
{{1, 2, 3}, {0, 1, 4}, {0, 0, 3}}
```

it is easiest to implement the projection function given by Pxy as

In[275]:=
```
pts.Transpose[Pxy]
```

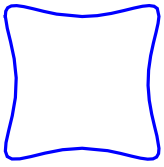
Out[275]=
```
{{1, 2}, {0, 1}, {0, 0}}
```

A better example using Example 1.1.1

In[276]:=
```
Pth = Join[pth1, pth2, pth3, pth4, pth5];
pth = Pth.Transpose[Pxy];
```

In[279]:=
```
Graphics[{Blue, Thick, Line[pth]}, ImageSize → Tiny]
```

Out[279]=



So if we path trace a curve in $\mathbb{R}^3$ we can plot its projection in $\mathbb{R}^2$. However the main technique in this book is to find the equation of a space curve after projection to the plane. In Chapter 2 we will learn how to do this algebraically from the equations but for now we can simply project a sufficient number of sufficiently random points and reconstruct an equation interpolating by my plane interpolation function acurve. Here it is as a 2D function in our Space Curve global functions:

```
aCurve2D [pts _, x_, y_] := Module [{d, P, M, B, n, c, pow},
      pow [a_, n_] := If[n == 0, 1, a ^ n];
      d = Switch [Length [pts], 2, 1, 5, 2, 9, 3, 14, 4, 20,
          5, 27, 6, _, Return ["number of points must be 2,5,9,14,20,27"]];
      P = exps [2, d];
      n = Length [P];
      M = Table [If[Length [p] == 2, pow [p⟦1⟧, e⟦1⟧] * pow [p⟦2⟧, e⟦2⟧],
            pow [p⟦1⟧, e⟦1⟧] * pow [p⟦2⟧, e⟦2⟧] * pow [p⟦3⟧, d – e⟦1⟧ – e⟦2⟧]], {p, pts}, {e, P}];
      AppendTo [M, RandomReal [{–1, 1}, n]];
      B = Append [Table [0, {n – 1}], 1];
      c = LinearSolve [M, B];
      FromCoefficientRules  [Table [P⟦i⟧ → c⟦i⟧, {i, n}], {x, y}]
    ];
```

Note from my plane curve book that the number of points to use to get a polynomial of degree d is $\binom{d+2}{2} - 1 = \frac{(d+2)(d+1)}{2} - 1$.

One difficult issue with space curves is calculating the degree of a projection. This depends on both the equations and the projection matrix. But generically in the case of a naive curve given by equations of degrees d_1, d_2 the degree of a reasonably random plane projection is $d_1 * d_2$.

For Example 1.1.1 both equations are quadratics so the degree of the curve is 4. By interpolation we need $6 * 5 / 2 - 1 = 14$ points. It turns out, relative to these specific equations that **Pxy** is sufficiently random. We can get 14 points easily from our projected path tracing.

In[295]:=
```
pts2 = RandomSample[pth, 14];
g = aCurve2D[pts2, x, y]
```

Out[296]=
$$3.35997 - 6.01438 \times 10^{-13}\, x - 0.839992\, x^2 + 3.99152 \times 10^{-13}\, x^3 - 0.839992\, x^4 - 4.91802 \times 10^{-13}\, y - 5.67127 \times 10^{-13}\, x\, y + 4.48573 \times 10^{-13}\, x^2\, y + 1.99789 \times 10^{-13}\, x^3\, y - 0.839992\, y^2 + 3.16171 \times 10^{-13}\, x\, y^2 + 1.67998\, x^2\, y^2 + 1.96307 \times 10^{-13}\, y^3 + 3.63534 \times 10^{-13}\, x\, y^3 - 0.839992\, y^4$$

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

In[297]:=
```
pf1 = Chop[g, 1.*^–9]
```

Out[297]=
$$3.35997 - 0.839992\, x^2 - 0.839992\, x^4 - 0.839992\, y^2 + 1.67998\, x^2\, y^2 - 0.839992\, y^4$$

In fact, this looks like an exact polynomial, so divide by the smallest coefficient

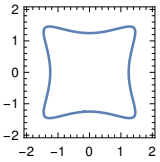
In[298]:=
```
pf1 = Expand[pf1 / Coefficient[pf1, y ^ 4]]
```

Out[298]=
$$-4. + 1.\, x^2 + 1.\, x^4 + 1.\, y^2 - 2.\, x^2\, y^2 + 1.\, y^4$$

The plot is the same as above.

In[300]:=
```
ContourPlot[pf1 == 0, {x, -2, 2}, {y, -2, 2}, ImageSize → Tiny]
```

Out[300]=



But notice instead if we use a different projection we get a badly contitioned matrix

In[318]:=
```
Pyz = {{0, 1, 0}, {0, 0, 1}};
pts3 = RandomSample[Pth.Transpose[Pyz], 14];
pf2 = aCurve2D[pts3, x, y]
```

LinearSolve : Result for LinearSolve of badly conditioned matrix {{1., 1.25159 , 0.556915 , 1.56647 , ≪7≫, 1.09187 , 0.485846 , 0.216185 , 0.0961955 }, ≪13≫, {-≪20≫ , ≪14≫}} may contain significant numerical errors .

Instead we can suspect the possibility of a degree 2 projection and use 5 points

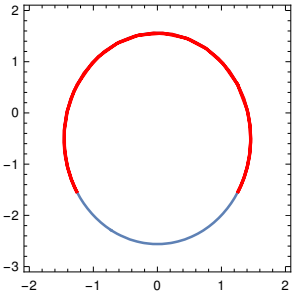
In[326]:=
```
pts3 = RandomSample[Pth.Transpose[Pyz], 5];
pth2dyz = Pth.Transpose[Pyz];
pf2 = Chop[aCurve2D[pts3, x, y], 1.*^-9]
```

Out[328]=
$$-0.9141 + 0.45705\,x^2 + 0.228525\,y + 0.228525\,y^2$$

This is just a circle, due partly because our curve lies on a sphere in $\mathbb{R}^3$.

In[335]:=
```
Show[ContourPlot[pf2 == 0, {x, -2, 2}, {y, -3, 2}, ImageSize → Small],
  Graphics[{Red, Thick, Line[pth2dyz]}]]
```

Out[335]=



In fact, the actual point projection is only part of a circle! This is an important lesson, *the point projection of an algebraic space curve will lie in an algebraic curve but may not be the entire curve.* The smallest algebraic curve containing the point projection is known to algebraic geometers as the *Zariski Closure* of the projection.

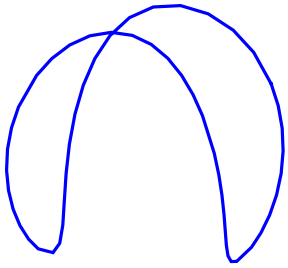
So this is why it is important to use generic, that is, random projections. Sometimes it is useful, for replication, to have only a pseudo-random projection that we will use over and over. The one I have chosen is known as **prd3D** and given by

In[336]:=
```
prd3D
```

Out[336]=
```
{{-0.305198, 0.952289, 0.}, {-0.141911, -0.0454808, 0.988834}}
```

In[341]:=
```
pth2dr = Pth.Transpose[prd3D];
Graphics[{Blue, Thick, Line[pth2dr]}, ImageSize → Small]
```

Out[342]=



This brings up another issue. When curves are projected the projection may have singular points even though the original curve did not have a singular point or at least not one that projects to this singularity. I will call such points, non-standardly, *artifactual*. In fact, for many curves, including this one, generic projections **must** include artifactual points, although very possibly complex or infinite. We will discuss this at the end of this book when considering *genus*. In addition to ordinary crossings these artifactual singularities may be cusps or isolated points.

For an example of an artifactual cusp we introduce the famous *twisted cubic* to be discussed at the beginning of Chapter 2. This is a curve generally given parametrically as $t \mapsto \{t, t^2, t^3\}$. As we will explain in Chapter 3 such curves are algebraic, although even in $\mathbb{R}^3$not necessarily naive. In fact this curve is the poster child for non-naive curves but is contained in the naive curve

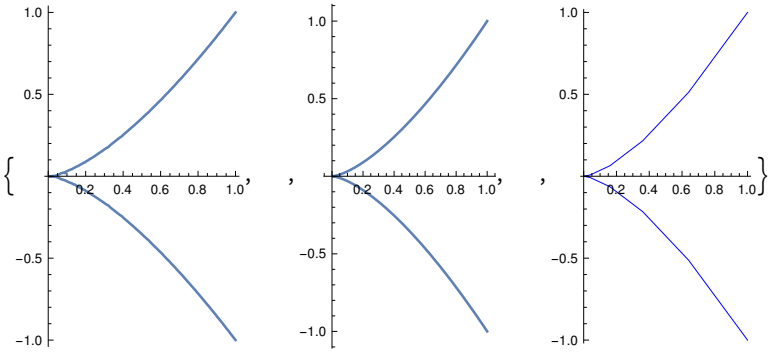
In[120]:=
```
F2 = {y - x ^ 2, z - x y};
```

where the extra component lies in the infinite plane so won't influence this discussion. If we project to the plane with Pyz which sends the first component to 0 then from the parametric expression we get the parametric plane curve $t \mapsto \{t^2, t^3\}$ which we recognize as a cusp. Or we can easily describe a set of points plotting the curve

In[181]:=
```
twcpts = Table[{t, t ^ 2, t ^ 3}, {t, -1, 1, .2}];
ptwcpts = twcpts.Transpose[Pyz];
```

In[183]:=
```
{ContourPlot[y^2 == x^3, {x, 0, 1}, {y, -1, 1}, ImageSize → Small,
   Axes → True, Frame → False, AspectRatio → 1.75], Invisible["xxx"],
 ParametricPlot[{t^2, t^3}, {t, -1, 1}, ImageSize → Small], Invisible["xxx"] ,
 Graphics[{Blue, Line[ptwcpts]}, Axes → True, ImageSize → Small]}
```

Out[183]=



As for the possibility of the projection having isolated artifactual singular points the easiest example is projecting the z-axis, that is the naive space curve $\{x = 0,\ y = 0\}$ with Pxy.

One can certainly find non-singular curves and projections giving more complicated artifactual singularities. For example see the section on *blowing-up* in Chapter 3 to see how to make any plane singularity artifactual. But for this to happen with a truly generic projection generated independently from the curve is very unlikely.

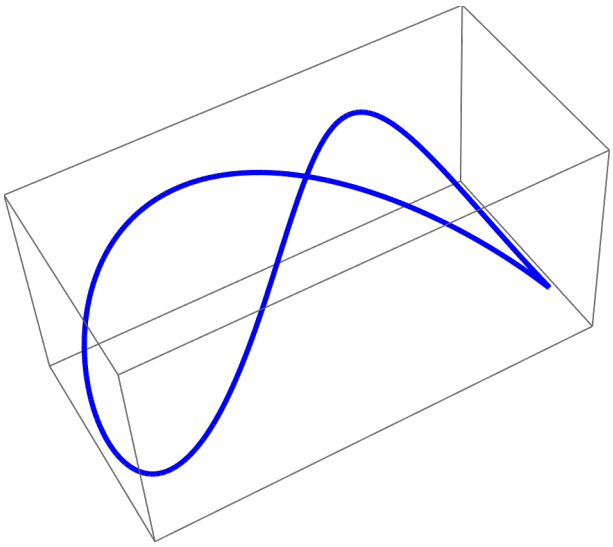
1.2.1 Nice Example: Viviani Curve

The Viviani Curve [see https://www.wolframalpha.com/input/?i=Vivian i+Curve] 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}

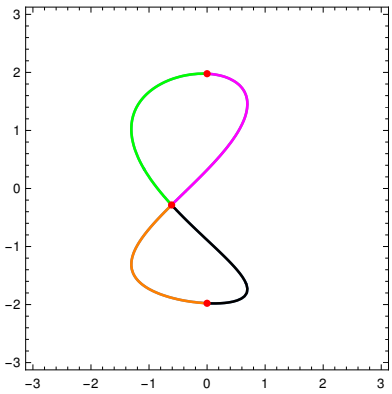
Out[287]= {0., 0., 0.}

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

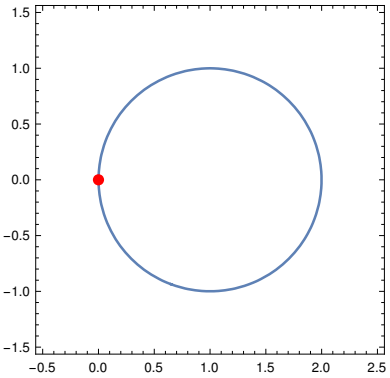
circle

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]=



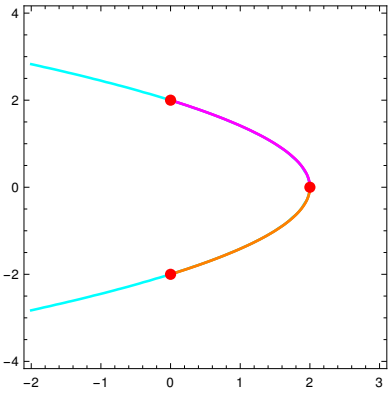
where again the red point is the image of the singular point. Each other point of the circle has a 2 point fiber. This is an example of how a non-generic projection can take a singular point to a non-singular point.

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

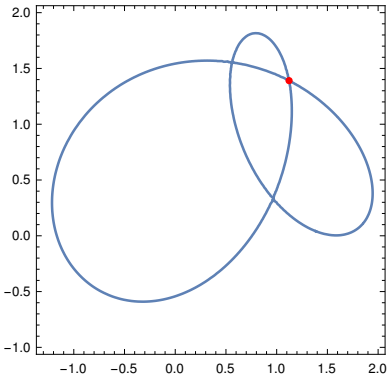
example of where a space curve may not map onto the FLT projection curve, particularly in the case of a non-random projection.

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.

In the plane curve book I defined Fractional Linear Transformations in Chapter 6 and use them heavily there. On the point level these are given in the Wolfram Language under the name **TransformationFunction**. My abbrevia - tion for **TransformationFunction** was flt. Since **TransformationFunction** works in all dimensions this appears here as **fltMD[p,A]**. Note neither the curve we are working with nor the variables matter so we need to know only the point p and the transformation matrix A which needs to be neither square nor invertible. However, in the affine case the number of columns needs to be 1 more than the length of p and the length of the output will be one less than the number of rows. That is, a $(n+1)\times(m+1)$ transformation matrix takes a point in $\mathbb{R}^m$ to a point in $\mathbb{R}^n$. If p is an infinite point then **TransformationFunction** should be replaced by either matrix multiplication $A.p$ or **fltiMD[p,A]**. Then a $(n+1)\times(m+1)$ transformation matrix takes projective $\mathbb{P}^n$ to projective $\mathbb{P}^m$. Actually **fltiMD[p,A]** will accept either an affine point of length m or an infinite point of length $m+1$ and if the result is not an infi - nite point it will be represented as an affine point of length n.

However an important observation was that invertible Transformation Matrices actually take curves to curves on the equation level. In the plane case this was simple as each curve is given by a single bivariate polynomial. This plane case is represented here by **FLT2D[f,A,x,y]**. This is easily extended to the naive case and given by **FLT3D[F,A,X]**.

```
FLT3D[F_, A_, X_] := Module[{B, d, g, h, t, n},
    n = Length[X];
    If[Dimensions[A] ≠ {n + 1, n + 1}, Echo[{n + 1, n + 1}, "need A to be of size"];
      Abort[]];
    If[MatrixRank[A] ≠ n + 1, Echo["A must be invertible"]; Abort[]];
    B = Inverse[A].Append[X, t];
    Reap[Do[
        d = tDegMD[f, X];
        g = Expand[t^d (f /. Thread[X → X/t])];
        h = Expand[g /. Thread[Append[X, t] → B]];
        Sow[Chop[h /. {t → 1}, dTol]], {f, F}]][[2, 1]]
```

Although we will keep the name **FLT3D** to distinguish this version from the much more complicated general **FLTMD**, the main workhorse and contribu - tion of this book, we note that **FLT3D** actually works in all dimensions and for systems of any number of equations as long as the transformation matrix is square and invertible. Unlike the more general **FLTMD** this works separately on each equation so the number of equations returned is the same as the

number entered.

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}]
```

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

so to translate the curve given by

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

use

```
In[114]:= FLT3D[F, ( 1  0  0 | 3
                     0  1  0 | -3
                     0  0  1 | 2
                     ------- -
                     0  0  0 | 1 ), {x, y, z}]
```

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

Wolfram also has rotations, reflections and scaling (homothety) transforms in n dimensions.

In addition we import **klRotation2D** and **ip2z2D** from our plane curve book (code in GlobalFunctions.nb) but note that the latter does not need the dummy variables x, y entered, the syntax is simply **ip2z2D[ip]** where ip is the infinite point.

For 3 dimensions we have a generalization of **klRotation2D** , **uvRotationMD[u,v]** which takes the vector u and rotates it about the origin until it is in the direction of v and a transformation matrix **ip2z3D[ip]** which takes the infinite point ip and places it at the origin.

```
In[125]:= ip2z3D[ip_] := Module[{p, A}, p = Take[ip, 3];
            A = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}, {1, 1, 1, 0}};
            If[Norm[Take[p, 2]] < 1.*^-6, A, A.uvRotationMD[p, {0, 0, 1}]]]
```

Example 1.4.1:

```
In[197]:= F = {z (x ^ 2 + y ^ 2) - 1, x + y};
          ips = infiniteRealPoints3D [F, {x, y, z}]
```

```
Out[198]= {{0., 0., -1., 0}, {0., 0., -1., 0}, {-0.707107 , 0.707107 , 0., 0}}
```

In[199]:= **Aip1 = ip2z3D [{0, 0, 1}]**

Out[199]= {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}, {1, 1, 1, 0}}

In[201]:= **F1 = FLT3D [F, Aip1 , {x, y, z}]**

Out[201]= $\{x^2 - x^3 - x^2\, y + y^2 - x\, y^2 - y^3 - z^3,\ x + y\}$

In[217]:= **tangentVector3D[F1, {0, 0, 0}, {x, y, z}]**

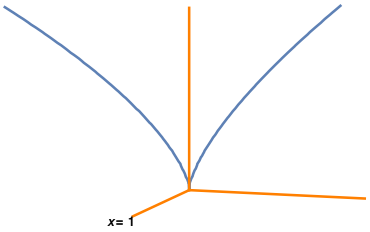
» No tangent vector at {0, 0, 0}

Out[217]= {0., 0., 0.}

In[202]:= **showProjection3D [F1, fprd3D , 4, {x, y, z}, {x, y}, 1]**

» projection Function $\{-1.22291\ x^2 - 0.00045095\ x^3 + 0.0176417\ x^2\, y - 0.230054\ x\, y^2 + 1.\ y^3\}$

Out[202]=



So this infinite point has a cusp-like singularity at {0, 0, 1, 0}.

In[213]:= **Aip2 = ip2z3D [ips⟦3⟧]**
F2 = FLT3D [F, Aip2 , {x, y, z}]

Out[213]= {{0.5, 0.5, 0.707107 , 0.}, {0.5, 0.5, −0.707107 , 0.}, {0., 0., 0., 1.}, {0.292893 , 1.70711 , 0., 0.}}

Out[214]= $\{0.707107\ x - 1.41421\ x^2 + 1.06066\ x^3 - 0.707107\ y + 1.06066\ x^2\, y + 1.41421\ y^2 - 1.06066\ x\, y^2 - 1.06066\ y^3 - 1.\ z^3,\ 1.\ x + 1.\ y\}$

In[215]:= **tangentVector3D [F2, {0, 0, 0}, {x, y, z}]**

Out[215]= {0., 0., 1.}

So this other infinite point is non-singular.

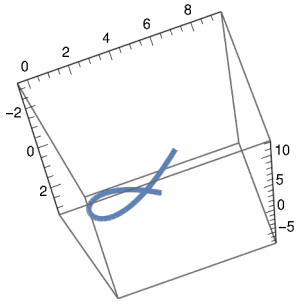
2 | General Case

We now treat the general case of a curve in $\mathbb{R}^n$, $n \geq 3$ with $k \geq n-1$ polynomial equations $F = \{f_1, f_{2,\cdots,} f_k\}$in the n variables. But first, some more numerical linear algebra.

2.1 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 **ip2z3D[{0,0,1,0}]**

In[243]:=
```
A = ip2z3D[{0, 0, 1, 0}];
eq = FLT3D[{y - x^2, z - x y}, A, {x, y, z}]
```

Out[244]=
```
{-x^2 + y z, -x y + z - x z - y z}
```

In[245]:=
```
{x, y, z} /. NSolve[Append[eq, l]]
```

Out[245]=
```
{{1.80204 + 1.60657 i, -2.37285 + 0.126197 i, -0.150579 - 2.4482 i},
 {1.80204 - 1.60657 i, -2.37285 - 0.126197 i, -0.150579 + 2.4482 i},
 {0., 2.16, 0.}, {0.384404, 0.330846, 0.446632}}
```

Again we get 4, not 3 solutions. We conclude it takes all 3 equations to define the twisted cubic! We will see this curve several more times.

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, F is a list of polynomials in the n variables X.

In[178]:=
```
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[p, "No unique tangent vector"]];
  Table[0, {Length[X]}]]
```

If a non-zero list of length n is returned then it is a tangent vector and p is called a *regular* point. Otherwise p is called a *non-regular point* point.

Example 2.2.1:

It is easy to see that the twisted cubic

In[171]:=
```
twCubic = {x z - y ^ 2, y - x ^ 2, z - x y};
```

is parameterized by $t \mapsto \{t,\ t^2,\ t^3\}$

In[172]:=
```
twCubic /. Thread [{x, y, z} → {t, t ^ 2, t ^ 3}]
```

Out[172]=
```
{0, 0, 0}
```

Picking a random point on this curve

In[173]:=
```
p = {t, t ^ 2, t ^ 3} /. {t → RandomReal [{-3, 3}]}
```

Out[173]=
```
{-1.05995 , 1.12349 , -1.19085 }
```

In[183]:=
```
tv1 = tangentVectorJ3D [twCubic , p, {x, y, z}]
```

Out[183]=
```
{0.243583 , -0.516372 , 0.820992 }
```

Note that in calculus or differential geometry the tangent vector of the curve at $t = p⟦1⟧$ would be defined by

In[182]:=
```
tv2 = D[{t, t ^ 2, t ^ 3}, t] /. {t → p⟦1⟧}
```

Out[182]=
```
{1, -2.1199 , 3.37048 }
```

But tangent vectors are defined only up to a non - zero constant

In[184]:=
```
Evaluate [tv1⟦1⟧ * tv2]
```

Out[184]=
```
{0.243583 , -0.516372 , 0.820992 }
```

which is tv1 so the classical definition of a tangent vector to a curve agrees with ours!

Example 2.2.2: We consider the apparent naive curve

In[185]:=
```
G = {x z, y z};
```

If $p = \{0,\ 0,\ r\}$ where r is random

In[190]:=
```
p1 = {0, 0, RandomReal [{−5, 5}]};
tangentVectorJMD [G, p1, {x, y, z}]
```

Out[191]= {0., 0., 1.}

so this is a regular point. But if $p = \{a, b, 0\}$ then it is a point on algebraic set G but is not regular

In[192]:=
```
p2 = Append [RandomReal [{−5, 5}, 2], 0];
tangentVectorJMD [G, p2, {x, y, z}]
```

» no unique tangent vector at {4.52064 , −4.0333 , 0}

Out[193]= {0, 0, 0}

Finally, any other point is not on the set.

In[194]:=
```
p3 = RandomReal [{−5, 5}, 3];
tangentVector3D [{−5, 5}, p3, {x, y, z}]
```

» not a point {−4.90776 , −2.23185 , 4.7079}

Out[195]= Fail

So this set has infinitely many regular points but also infinitely many non-regular points, so it is not a curve so even though it is given by 2 equations in 3 unknowns it is not a curve.

Example 2.2.3 Cyclic 4

Here is well studied curve in $\mathbb{R}^4$, the Cyclic 4. We will examine this further later on in this Chapter.

In[168]:=
```
C4 = {w + x + y + z, w x + x y + y z + z w, w x y + x y z + y z w + z w x, w x y z − 1};
```

We note that for any random number r, possibly complex, the points $\{r, 1/r, -r, -1/r\}$ and $\{r, -1/r, -r, 1/r\}$ are solutions.

In[178]:=
```
Clear [r]
C4 /. Thread [{w, x, y, z} → {r, 1 / r, −r, −1 / r}]
C4 /. Thread [{w, x, y, z} → {r, −1 / r, −r, 1 / r}]
```

Out[179]= {0, 0, 0, 0}

Out[180]= {0, 0, 0, 0}

But not all these points are regular

In[174]:=
```
r = RandomReal [{−4, 4}]
```

Out[174]= −0.30827

In[176]:= **tangentVectorJMD [C4, {r, 1/r, −r, −1/r}, {w, x, y, z}]**
tangentVectorJMD [C4, {r, −1/r, −r, 1/r}, {w, x, y, z}]

Out[176]= {0.0668952 , −0.703935 , −0.0668952 , 0.703935 }

Out[177]= {−0.0668952 , −0.703935 , 0.0668952 , 0.703935 }

But if $r = \pm 1$ or $r = \pm i$ then

In[181]:= **r = 1;**

In[182]:= **tangentVectorJMD [C4, {r, 1/r, −r, −1/r}, {w, x, y, z}]**
tangentVectorJMD [C4, {r, −1/r, −r, 1/r}, {w, x, y, z}]

» no unique tangent vector at {1, 1, −1, −1}

Out[182]= {0, 0, 0, 0}

» no unique tangent vector at {1, −1, −1, 1}

Out[183]= {0, 0, 0, 0}

In[184]:= **r = −1;**

In[185]:= **tangentVectorJMD [C4, {r, 1/r, −r, −1/r}, {w, x, y, z}]**
tangentVectorJMD [C4, {r, −1/r, −r, 1/r}, {w, x, y, z}]

» no unique tangent vector at {−1, −1, 1, 1}

Out[185]= {0, 0, 0, 0}

» no unique tangent vector at {−1, 1, 1, −1}

Out[186]= {0, 0, 0, 0}

In[187]:= **r = *i*;**

In[188]:= **tangentVectorJMD [C4, {r, 1/r, −r, −1/r}, {w, x, y, z}]**
tangentVectorJMD [C4, {r, −1/r, −r, 1/r}, {w, x, y, z}]

» no unique tangent vector at {*i*, −*i*, −*i*, *i*}

Out[188]= {0, 0, 0, 0}

» no unique tangent vector at {*i*, *i*, −*i*, −*i*}

Out[189]= {0, 0, 0, 0}

And similarly for $-i$. Thus this curve has 8 complex singular points. It can be shown that all solutions are of this form and only these 8 are singular so the cyclic 4 is a curve.

A strange fact, discussed later in this chapter, is that the parametric curves $\{r, 1/r, -r, -1/r\}$, $\{r, -1/r, -r, 1/r\}$ comprising the solution set of C4 are non-singular as parametric curves. So *singularity is based on the equation system rather than the geometry of the point set.* We have actually seen this before with plane curves $x + y = 0$ is non-singular but the same solution set is given by $(x + y)^2 = 0$ where every point is singular.

One other general algorithm we can give at this point is a general critical point finder. It does not work as well as criticalPoint3D for naive curves but

it will return some critical points using standard optimization techniques. It does not give isolated points but it may identify possibly singular points by repeated solutions. This method was suggested by the paper by [Wang, Bican] but since the methods are not specifically related to this paper we just give the code. The objective function is random so you might run this several times.

```
In[8]:= Options[criticalPointsMD ] = {solutions → Reals};
criticalPointsMD [F_, X_, OptionsPattern []] := Module[{uv, n, k, wbg, b},
    n = Length[X];
    k = Length[F];
    uv = Table[u[i], {i, k}];
    b = RandomReal [{−1, 1}, {n, 1}];
    Echo[X.b − RandomReal [{−1, 1}], "Objective Function "];
    wbg = Flatten[Expand[uv.Grad[F, X] − b]];
    X /. NSolve[Join[F, wbg], Join[X, uv], OptionValue [solutions ]]]
```

2.2.4 Example 2.2.3 continued.

```
In[144]:= criticalPointsMD [C4, {w, x, y, z}]
» Objective Function {−0.00220762 + 0.730347 w − 0.564975 x − 0.446484 y + 0.993358 z}
Out[144]= {w, x, y, z}

In[148]:= ccp4 = criticalPointsMD [C4, {w, x, y, z}, solutions → Complexes ]
» Objective Function {−0.57865 + 0.596792 w − 0.160889 x + 0.100207 y − 0.981373 z}
Out[148]= {{9.93591 × 10^−8 − 1. i, −6.69459 × 10^−8 + 1. i, 6.69474 × 10^−8 + 1. i, −9.93606 × 10^−8 − 1. i},
  {1. − 1.01799 × 10^−7 i, 1. + 1.01799 × 10^−7 i, −1. − 5.92361 × 10^−7 i, −0.999999 + 5.92361 × 10^−7 i},
  {−1. − 6.95247 × 10^−7 i, −1. + 6.95247 × 10^−7 i, 1. − 2.01104 × 10^−7 i, 0.999999 + 2.01104 × 10^−7 i},
  {1. + 4.11075 × 10^−7 i, 1. − 4.11076 × 10^−7 i, −1. + 6.44503 × 10^−7 i, −0.999999 − 6.44503 × 10^−7 i}}

In[149]:= Chop[ccp4, 1.*^−6]
Out[149]= {{0. − 1. i, 0. + 1. i, 0. + 1. i, 0. − 1. i}, {1., 1., −1., −0.999999},
  {−1., −1., 1., 0.999999}, {1., 1., −1., −0.999999}}
```

2.3 Macaulay and Sylvester Matrices

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.

2.3.1 Construction of Macaulay and Sylvester Matrices.

As mentioned above these matrices can be large, therefore it is important to have efficient methods for constructing these. Fortunately Mathematica has adequate data manipulation methods which allow us to do that.

I generally use m to represent the *order* of a Macaulay or Sylvester Matrix, this is the largest total degree of a monomial to be considered. The columns of both types represent the monomials in given variables up to total degree m. One can get a list of the monomials in the ordering used by the routine mE**xpsMD**. For example the columns of order 3 with 3 variables $\{x, y, z\}$ correspond to the following list.

```
In[120]:= mExpsMD[3, {x, y, z}]
Out[120]= {1, x, y, z, x^2, x y, x z, y^2, y z, z^2, x^3, x^2 y, x^2 z, x y^2, x y z, x z^2, y^3, y^2 z, y z^2, z^3}
```

For Sylvester matrices the rows represent the coefficients of monomials in this list of the multivariate polynomials defining a curve, or other algebraic set, together with multiples of these polynomials by monomials of degree small enough that the product is of degree less than or equal to m. For

Macaulay matrices we apply a change of variables sending the given point to the origin and then allow multiplication by all monomials of order less than m but then truncating the result by dropping all terms of total degree greater than m. If this truncating results in the zero row we do not add this row to the Macaulay matrix.

Note that if m is smaller than the largest total degree of the equations the Sylvester Matrix would be empty or will ignore some equations, so our routine **sylvesterMD** will refuse a result returning only an error message. On the other hand, **macaulayMD** will return a result for any $m \geq 1$. As you will notice in the applications we generally use small orders for the Macaulay matrix but need larger orders for the Sylvester matrix. Although for the same m the Macaulay matrix will have far more rows than the Sylvester matrix it is misleading to say Macaulay matrices are larger since we use smaller orders for the Macaulay matrices than the degrees of the equations.

In the rest this subsection I will explain the details of the construction, the reader uninterested in these will skip to the next subsection.

Rather than using the actual variables, since only coefficients appear in these matrices we use integer lists corresponding to the exponents of the monomials, so, for example, if our variables are given as $\{x, y, z, w\}$ then instead of the monomial $x^2 z w^3$ we will use {2,0,1,3}. Note that our variables are used in the order indicated in the last argument X. Here n is the number of variables.

Our first task is to create the list of possible exponents. We do this one degree at a time with a recursive routine, essentially getting homogeneous monomials hence the "h".

```
hExpsMD [n_, d_] := Module [{hps},
      hps[0] = {Table [0, {n}]};
      hps[m_] := hps[m] = DeleteDuplicates [
            Flatten [Table [ReplacePart [p, i → (p[[i]] + 1)], {p, hps[m – 1]}, {i, n}], 1]];
      hps[
        d]];
```

For instance

```
In[131]:= hExpsMD [4, 3]

Out[131]= {{3, 0, 0, 0}, {2, 1, 0, 0}, {2, 0, 1, 0}, {2, 0, 0, 1}, {1, 2, 0, 0}, {1, 1, 1, 0},
    {1, 1, 0, 1}, {1, 0, 2, 0}, {1, 0, 1, 1}, {1, 0, 0, 2}, {0, 3, 0, 0}, {0, 2, 1, 0}, {0, 2, 0, 1},
    {0, 1, 2, 0}, {0, 1, 1, 1}, {0, 1, 0, 2}, {0, 0, 3, 0}, {0, 0, 2, 1}, {0, 0, 1, 2}, {0, 0, 0, 3}}
```

To get the list for all degrees up to d we use the trick

```
expsMD [n_, d_] := Drop [hExpsMD [n + 1, d], None , 1];
```

In[133]:=
```
Timing[expsMD[4, 3]]
```

Out[133]=
```
{0.001096, {{0, 0, 0, 0}, {1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}, {2, 0, 0, 0}, {1, 1, 0, 0},
  {1, 0, 1, 0}, {1, 0, 0, 1}, {0, 2, 0, 0}, {0, 1, 1, 0}, {0, 1, 0, 1}, {0, 0, 2, 0}, {0, 0, 1, 1},
  {0, 0, 0, 2}, {3, 0, 0, 0}, {2, 1, 0, 0}, {2, 0, 1, 0}, {2, 0, 0, 1}, {1, 2, 0, 0}, {1, 1, 1, 0},
  {1, 1, 0, 1}, {1, 0, 2, 0}, {1, 0, 1, 1}, {1, 0, 0, 2}, {0, 3, 0, 0}, {0, 2, 1, 0}, {0, 2, 0, 1},
  {0, 1, 2, 0}, {0, 1, 1, 1}, {0, 1, 0, 2}, {0, 0, 3, 0}, {0, 0, 2, 1}, {0, 0, 1, 2}, {0, 0, 0, 3}}}
```

constructing this list in about .001 seconds. Note, as an aside, that we can use this to get all the monomials of degree less than or equal to d in an arbitrary set of variables.

In[119]:=
```
mExpsMD [d_, X_] := Module [{n},
    n = Length [X];
    Table [FromCoefficientRules  [{p → 1}, X], {p, expsMD [n, d]}]];
```

In[120]:=
```
mExpsMD [3, {x[1], x[2], x[3]}]
```

Out[120]=
```
{1, x[1], x[2], x[3], x[1]^2, x[1] × x[2], x[1] × x[3], x[2]^2, x[2] × x[3], x[3]^2, x[1]^3, x[1]^2 x[2],
  x[1]^2 x[3], x[1] x[2]^2, x[1] × x[2] × x[3], x[1] x[3]^2, x[2]^3, x[2]^2 x[3], x[2] x[3]^2, x[3]^3}
```

Next we convert the built in CoefficientRules to an association adding missing monomials. As an extra we calculate the total degree.

```
fAssocMD [f_, X_] := Module [{A, d, n, FA},
  n = Length [X];
  A = Association [CoefficientRules  [f, X]];
  d = Max [Table [Total [p], {p, Keys [A]}]];
  FA = Association [Table [If [MissingQ [A[p]], p → 0, p → A[p]], {p, exps [n, d]}]];
  {FA, d}]
```

In[145]:=
```
FA = fAssocMD [1 + 3 x – 2 x y, {x, y}]
```

Out[145]=
```
{⟨| {0, 0} → 1, {1, 0} → 3, {0, 1} → 0, {2, 0} → 0, {1, 1} → −2, {0, 2} → 0 |⟩ , 2}
```

We perform multiplication by a monomial by shifting and adding in missing terms.

In[24]:=
```
shiftFAMD [FA_, q_, d_] := Module [{S, K, n},
    K = Keys [FA];
    n = Length [K⟦1⟧];
    S = Association [Table [p + q → FA[p], {p, K}]];
    Association [Table [If [MissingQ [S[p]], p → 0, p → S[p]], {p, exps [n, d]}]]];
```

In the above example we multiply by monomial x^2 for use with order 4.

In[149]:=
```
sFA = shiftFAMD [FA⟦1⟧, {2, 0}, 4]
```

Out[149]=
```
⟨| {0, 0} → 0, {1, 0} → 0, {0, 1} → 0, {2, 0} → 1, {1, 1} → 0, {0, 2} → 0, {3, 0} → 3,
  {2, 1} → 0, {1, 2} → 0, {0, 3} → 0, {4, 0} → 0, {3, 1} → −2, {2, 2} → 0, {1, 3} → 0, {0, 4} → 0 |⟩
```

Note we can recover our product $x^2 (1 + 3\, x\, - 2\, x\, y)$

In[150]:=
```
FromCoefficientRules [Normal [sFA], {x, y}]
```

Out[150]=
$$x^2 + 3\,x^3 - 2\,x^3\,y$$

Now we treat the special case of one equation

In[25]:=
```
sylMD [f_, m_, X_] := Module [{FA, d},
      n = Length [X];
      {FA, d} = fAssocMD [f, X];
      If[d > m, Print["Degree error in syl"]; Abort []];
      Table [Values [shiftFAMD [FA, q, m]], {q, exps [n, m – d]}]];
```

In[152]:=
```
sylMD [1 + 3 x – 2 x y, 4, {x, y}] // MatrixForm
```

Out[152]//MatrixForm=
$$\begin{pmatrix} 1 & 3 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 3 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & -2 & 0 \end{pmatrix}$$

For the general Sylvester matrix case we apply the above equation by equation.

In[26]:=
```
sylvesterMD [F_,m_,X_]:=Flatten [Table [sylMD [F⟦i⟧,m,X],{i,Length [F]}],1];
```

The Macaulay matrix is similar with exception of using the following instead of sylMD in the one equation case:

```
macaMD [f_, m_, p_, X_] := Module [{M, fp, FA, d, n},
    fp = Expand [f /. Thread [X → X + p]];
    n = Length [X];
    {FA, d} = fAssocMD [fp, X];
    M = Table [Values [shiftFAMD [FA, q, m]], {q, expsMD [n, m – 1]}];
    Select [M, AnyTrue [#, Abs [###] > 0 &] &]]
```

In[154]:=
```
macaMD [1 + 3 x – 2 x y, 3, {–1 / 3, 0}, {x, y}] // MatrixForm
```

Out[154]//MatrixForm=
$$\begin{pmatrix} 0 & 3 & \frac{2}{3} & 0 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & \frac{2}{3} & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & \frac{2}{3} & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & \frac{2}{3} \end{pmatrix}$$

To finish

In[28]:=
```
macaulayMD [F_, m_, p_, X_] := Flatten [Table [macaMD [F⟦i⟧, m, p, X], {i, Length [F]}], 1];
```

2.3.2 Application of Sylvester Matrices.

Sylvester matrices will play a large role below. For those readers familiar with contemporary algebraic geometry they essentially replace the concept of *ideal*. So we give only two simple applications here which are multivariate extensions procedures in Appendix 1 of our plane curve book.

2.3.2.1 Numerical Division of multivariate polynomials.

Given polynomials f, g of degrees d_1, d_2 in variables X we note that we can use **sylMD** to do matrix multiplication with the formulas

sylMD[f ∗ g, $d_1 + d_2$, X] = syl[f, d_1, X].syl[g, $d_1 + d_2$, X]

h = f ∗ g = sylMD[h, $d_1 + d_2$, X].mExpsMD[$d_1 + d_2$, X]

Of course this will be about 100 times slower than the built-in **Expand[f∗g]** but it gives us a suggestion for undoing this multiplication : recover f by multiplying **sylMD[h, $d_1 + d_2$, X]** on the right by $\mathbf{syl}[g, d_1 + d_2, X]^{-1}$. Of course this rarely would be an invertible matrix but we can use the pseudo-inverse instead. This gives us the procedure, using the faster **FromCoefficientRules** instead of multiplying by **mExps**:

```
nDivideMD [h_, g_, X_, tol_] := Module [{n, l, m, d1, d2, P, S, f, ex},
      n = Length [X];
      d1 = tDegMD [g, X];
      d2 = tDegMD [h, X];
      If [d1 > d2, Print ["Does Not Divide "]; Return [Fail]];
      P = PseudoInverse [N[sylMD [g, d2, X]], Tolerance → tol];
      S = Chop [sylMD [h, d2, X].P];
      ex = expsMD [n, d2 – d1];
      l = Length [ex];
      f = FromCoefficientRules [Table [ex⟦i⟧ → S⟦1, i⟧, {i, l}], X];
      If[Norm [Flatten [sylMD [f ∗ g – h, d2, X]]] > d2 ∗ tol,
        Print ["Does not divide at this tolerance "];
        Return [Fail]];
      f];
```

Of course you cannot use this on arbitrary h, g but, especially with numerical polynomials, even if h does factor we probably need to use a looser tolerance than **dTol**.

Example 2.3.2.1.1 We divide three variable polynomials h by g

```
In[149]:= h = 10 + 36 x + 38 x^2 + 4 x^3 – 28 x^4 – 28 y – 70 x y – 16 x^2 y + 92 x^3 y – 19 y^2 – 64 x y^2 – 57 x^2 y^2 + 40 y^3 –
           5 x y^3 + 25 y^4 – 34 z – 76 x z – 2 x^2 z – 6 x^3 z – 21 y z – 98 x y z + 69 x^2 y z + 92 y^2 z – 14 x y^2 z +
           65 y^3 z + 47 z^2 + 94 x z^2 – 5 x^2 z^2 + 2 y z^2 + 58 x y z^2 – 14 y^2 z^2 – 40 z^3 – 35 x z^3 – 65 y z^3 + 25 z^4;

In[150]:= g = 5 + 8 x – 7 x^2 – 4 y + 9 x y – 5 y^2 – 2 z – 5 x z – 4 y z + 5 z^2;
```

In[152]:= **nDivideMD [h, g, {x, y, z}, dTol]**

Out[152]= $2. + 4.\, x + 4.\, x^2 - 4.\, y - 8.\, x\, y - 5.\, y^2 - 6.\, z - 2.\, x\, z - 9.\, y\, z + 5.\, z^2$

This idea can be extended to finding the greatest common divisor of 2 n-variable polynomials. In my plane curve book I give the code in the case of 2 variables but it is easily extended to the general n-variable case. The code is in my GlobalFunctionsMD notebook. For more information on this algorithm see our paper [Zeng,Dayton 2004].

2.3.2.2 The Membership Problem

When using more than 2 variables a more common and important question than GCD finding is the ideal membership problem *given polynomial g in n-variables X is it a polynomial combination of n-variable polynomials* $\{f_1, \ldots, f_k\}$?

The easiest way to handle this in general is as follows. Set a tolerance τ which could be **dTol** for or weaker for numerical systems. Suppose **tDeg[g,X]** = d_g. Then we calculate the ranks by

S = sylvesterMD[{f_1, ..., f_k}, m_i, X];
r1 = Length[SingularValueList[S, Tolerance → τ]]
r2 = Length[SingularValueList[Append[S, sylMD[g, m_i, X]⟦1⟧], Tolerance → τ]]

starting with $m_1 = \text{Max}\left[\{f_1, \ldots, f_k, d_{g.}\}\right]$ If these are equal then g is a member. If not then let $m_2 \geq m_1 + 1$ and try again. We continue this way for a few more tries but give up after about 3 or 4 tries concluding that g is probably not a polynomial combination of $\{f_1, \ldots, f_k\}$.

2.3.2.2.1 Example:

In[172]:= **f1 = x + y − 2 z + y z^2 − z^4;**
f2 = −x^2 + y − x y + 2 x z − z^2 − x y z^2 + x z^4;
g = x + y − 2 z;

We can take m_1 = 5

In[175]:= **S = sylvesterMD [{f1, f2}, 5, {x, y, z}];**
r1 = Length [SingularValueList [S, Tolerance → dTol]]
r2 = Length [SingularValueList [Append [S, sylMD [g, 5, {x, y, z}]⟦1⟧], Tolerance → dTol]]

Out[176]= 5

Out[177]= 6

These are not equal. Trying m_2 = 6 they still are not equal, but

```
In[178]:= S = sylvesterMD [{f1, f2}, 7, {x, y, z}];
          r1 = Length [SingularValueList [S, Tolerance → dTol]]
          r2 = Length [SingularValueList [Append [S, sylMD [g, 7, {x, y, z}][[1]], Tolerance → dTol]]
Out[179]= 30
Out[180]= 30
```

So g is a polynomial combination of {f1, f2}.

Things can be much worse, that is the final m_i could be much bigger than m_1 and as formulated there is no stopping criterion in this method to conclude that g is not a polynomial combination of the f_i. In the next section 2.4 we will see that there is a defect in the curve system $\{f_1, f_2\}$, we should have g and one more polynomial in our system and then the first try will be definitive.

2.3.3 Applications of Macaulay Matrices

2.3.3.1 Intersection Multiplicity

The main application of Macaulay matrices is Macaulay's original application, the computation of intersection multiplicity. For this application we will have a system of n or more equations in n variables, $n \geq 2$, and an isolated solution. In our case *isolated* means a solution point p such that there is no other solution point q such that **Norm[p−q]<ϵ** for some $\epsilon > 0$. In particular p will not be a regular point of a curve.

A full description of this algorithm is given in [Dayton-Li- Zeng] where we emphasize that *multiplicity* is not just a number. This was known but not well known previously. We describe this concept with a sequence called, historically, the *Hilbert Function* although the reader is forewarned that there are other different sequences with this name in the literature and even in this book where in addition to this *local* Hilbert Function there is a global Hilbert Function. Essentially this measures the change in dimension of the null space of the Macaulay matrix of order m as m increases. The first term $(m = 0)$ of the Hilbert Function should be 1 indicating that point p is a zero of our system. The second term $(m = 1)$ we call the *breadth* which is also known as the *embedding dimension* by algebraic geometers, it is always less than n. The fact that p is isolated implies that the numbers in this Hilbert Function become zero at some point, once this happens it will continue to happen if we calculated further. The order of the last non-zero number in the Hilbert function we call the *depth* which should not be confused with Macaulay's notion of depth. Finally the sum of all non-zero numbers in the Hilbert Function is simply called the *intersection multiplicity,* or just

multiplicity.

As mentioned above in section 2.3.1 the Macaulay matrix for large m, n can be very large already when m or n is greater than 3. Thus calculating null spaces can be time consuming. Therefore I give several algorithms for multiplicity.

The first is our original which requires the user to guess an upper bound for the depth. It is the only version that gives the Hilbert Function. Usually this is a small number so the calculation will be quick. If the depth turns out to be large this version stops before termination so as not to force the user to wait. On the other hand this version does not halt at the first occurrence of 0 in the Hilbert function. In order to make this as fast as possible we calculate only the final Macaulay Matrix and deduce the Hilbert function from that.

We need two subroutines. The first, **nrref** is simply a numerical version of the reduced row echelon form, the code, in GlobalFunctions.nb will be discussed in the next subsection. This **nrref** does return a sequence which allows computation of the Hilbert function. Here is that algorithm.

```
Options[hilbertFunctionMD ] = {diff → False};
hilbertFunctionMD [p_, m_, n_, OptionsPattern []] :=
  Module[{h}, h = Table[Binomial[d + n - 1, n - 1] -
          Length[Select[p, Binomial[d + n - 1, n] < # ≤ Binomial[d + n, n] &]], {d, 0, m}];
    If[OptionValue[diff], Differences[Prepend[h, 0]], h]]
```

Then the multiplicity algorithm is

In[128]:=
```
multiplicity0MD [F_, m_, p_, X_, tol_] := Module[{M, n, l, A, h},
    n = Length[X];
    M = macaulayMD[F, m, p, X];
    {l, A} = nrref[M, tol];
    h = hilbertFunctionMD [l, m, n];
    Echo[h, "hilbert Function "];
    Echo[Length[Select[h, # > 0 &]] - 1, "Depth "];
    If[h⟦m + 1⟧ > 0, Echo[h⟦n + 1⟧, "Warning: use higher m"]];
    Total[h]]
```

Here F is the equation system, m is the maximum order to compute, p is the isolated solution point, X is the set of variables and tol is a desired tolerance. For numerical systems, in particular, this can make a difference, for example a very loose tolerance can pick up nearby isolated points. But the reader should be aware that, for high depth, computation of intersection points accurately is a problem, see our paper [Dayton-Li-Zeng]. A loose tolerance

can make up for an inaccurately calculated intersection point. Note the built-in function **Timing** returns the time of execution and the value. If the last entry of the Hilbert function is not 0 a warning message is given.

Example 2.3.3.1.1: Consider the two variable system at {0,0}

In[125]:=
```
F = {x^2 - y^2 + x^3, x^2 - y^2 + y^3};
```

In[131]:=
```
Timing[multiplicity0MD[F, 6, {0, 0}, {x, y}, dTol]]
```

» hilbert Function {1, 2, 2, 1, 1, 0, 0}

» Depth 4

Out[131]= {0.038308, 7}

Our second version recalculates the Macaulay matrix at each step, but it stops at the first 0 in the Hilbert function so, since low multiplicities are the most common, is usually the fastest although this could run a long time if the depth is large. The user does not need to give an upper depth. The subroutines are not necessary.

```
multiplicityMD [F_, p_, X_, tol_] := Module[{ttd, svdl, cols, rnk, k, M, h, dh, hk},
    ttd = Total[tDegMD [#, X] &/@ F];
    k = tDegMD [F[[1]], X];
    dh = 1;
    h = 0;
    While[k ≤ ttd && dh > 0,
      M = macaulayMD [F, k, p, X];
      cols = Last[Dimensions [M]];
      rnk = Length[SingularValueList [N[M], Tolerance → tol]];
      hk = cols - rnk;
      dh = hk - h;
      h = hk;
      k++];
    h]
```

In[132]:=
```
Timing[multiplicityMD[{x^2 - y^2 + x^3, x^2 - y^2 + y^3}, {0, 0}, {x, y}, dTol]]
```

Out[132]= {0.033905, 7}

The final version assumes the maximal depth will be less than the sum of the total degrees of the equations. This seems to be valid, although the author has no proof. The advantage is the code is short but the answer is not guaranteed unless the multiplicity is less than the sum of total degrees.

```
multiplicity2MD [F_, p_, X_, tol_] := Module [{ ttd, M, svdl, cols, rnk, h},
    ttd = Total [tDegMD [#, X] & /@ F];
    M = macaulayMD [F, ttd, p, X];
    cols = Last [Dimensions [M]];
    rnk = Length [SingularValueList [N[M], Tolerance → tol]];
    cols – rnk]
```

In[133]:=
```
Timing[multiplicity2MD[{x^2 – y^2 + x^3, x^2 – y^2 + y^3}, {0, 0}, {x, y}, dTol]]
```

Out[133]= {0.040724, 7}

Example 2.3.3.1.2 Here is a numerical example:

In[244]:=
```
{a, b, c} = N[{Sqrt[7], Sqrt[11], CubeRoot [29]}]
```

Out[244]= {2.64575 , 3.31662 , 3.07232 }

In[245]:=
```
F0 = Expand [
    {(x – a)^3 + (y – b)^2 + (z – c)^2, (x – a)^2 + (y – b)^3 + (z – c)^2, (x – a)^2 + (y – b)^2 + (z – c)^3}]
```

Out[245]=
$$\{1.91887 + 21.\, x - 7.93725\, x^2 + x^3 - 6.63325\, y + y^2 - 6.14463\, z + z^2,$$
$$-20.0437 - 5.2915\, x + x^2 + 33.\, y - 9.94987\, y^2 + y^3 - 6.14463\, z + z^2,$$
$$-11. - 5.2915\, x + x^2 - 6.63325\, y + y^2 + 28.3174\, z - 9.21695\, z^2 + z^3\}$$

In[257]:=
```
sol = {x, y, z} /. NSolve [F0];
p = sol[[16]]
```

Out[258]=
$$\{2.64575 + 8.33743 \times 10^{-8}\, i,\ 3.31662 + 4.84024 \times 10^{-8}\, i,\ 3.07232 - 2.09602 \times 10^{-15}\, i\}$$

We first try multiplicity0MD to actually see what is happening

In[264]:=
```
Timing [multiplicity0MD [F0, 4, p, {x, y, z}, dTol]]
```

» hilbert Function {1, 0, 3, 3, 1}

» Depth 3

» Warning : use higher m 3

Out[264]= {0.494936 , 8}

In[265]:=
```
Timing [multiplicity0MD [F0, 4, p, {x, y, z}, 1.*^–6 ]]
```

» hilbert Function {1, 3, 3, 1, 0}

» Depth 3

Out[265]= {0.263942 , 8}

In both cases we get the same multiplicity but with tighter tolerance the wrong Hilbert function.

Now using the other methods

In[268]:=
```
Timing[multiplicityMD[F0, p, {x, y, z}, 1.*^-6]]
Timing[multiplicity2MD[F0, p, {x, y, z}, 1.*^-6]]
```

Out[268]= {0.057562, 8}

Out[269]= {3.0274, 8}

In this case we see **multiplicityMD** is the fastest but if we tried it with **dTol** perhaps getting the correct answer was luck.

2.3.3.2 Tangent Vectors

Our function **tangentVectorJMD** works in simple cases but may not work in near singular cases. An alternate uses the local property of the Macaulay matrix and gives some information about singular points encountered. Unlike the multiplicity finders above we do not expect to apply this to an isolated point so we will use a global Hilbert function rather than the local one used above. These Hilbert functions are related in some sense as integrals or derivatives of each other. The discrete built-in functions **Accumulate** and **Differences** will connect these two Hilbert functions.

Our function **nrref** mentioned above is very important here so we give the code.

```
nrref[M_, eps_] := Module[{p, P, j, R, mn, n, r, s, U, S, V},
    {U, S, V} = SingularValueDecomposition [N[M], Tolerance → eps];
    r = Length[Select[Diagonal[S], # > 0 &]]; (* rank *)
    R = Take[Transpose[V], r];  (* row space of M *)
     mn = Dimensions[R];
    n = 1;
    While[Norm[Take[R, All, {n}]] < eps, n++];
    p = {n};
    For[j = n, j > 0, j++,
      If[mn[[1]] ≤ Length[p], Break[],
          p = Append[p, j];
          P = Check[R[[All, p]], Abort[]];
          s = Length[Select[SingularValueList [N[P], Tolerance → eps], # > 0 &]];
          If[s < Length[p],
            p = Drop[p, -1];, Null];
        ];
    ];
    P = R[[All, p]];
    {p, Chop[Inverse[P].R]}]
```

Example 2.3.3.2.1 We consider example 2.2.1 the twisted cubic at {2,4,8}.

In[307]:=
```
twCubic = {x z – y ^ 2, y – x ^ 2, z – x y};
p = {2, 4, 8};
```

We calculate the Macaulay matrix at p for $m = 2$ since we are basically only interested in the linear part.

In[316]:=
```
M = macaulayMD [twCubic , 2, {2, 4, 8}, {x, y, z}];
M // MatrixForm
```

Out[317]//MatrixForm=

$$\begin{pmatrix} 0 & 8 & -8 & 2 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & -8 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & -8 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & -8 & 2 \\ 0 & -4 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & 1 & 0 \\ 0 & -4 & -2 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -4 & 0 & -2 & 1 \end{pmatrix}$$

Next we apply **nrref**

In[318]:=
```
{pv, M2} = nrref [M, dTol];
pv
M2 // MatrixForm
```

Out[319]= {2, 3, 5, 6, 7, 8, 9}

Out[320]//MatrixForm=

$$\begin{pmatrix} 0 & 1. & 0 & -0.0833333 & 0 & 0 & 0 & 0 & 0 & 0.00347222 \\ 0 & 0 & 1. & -0.333333 & 0 & 0 & 0 & 0 & 0 & 0.00694444 \\ 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & -0.00694444 \\ 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & -0.0277778 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & -0.0833333 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & -0.111111 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & -0.333333 \end{pmatrix}$$

Columns $2, 3, 4$ give the linear span of these equations which, since we have a curve should be of dimension $n - 1 = 2$.

In[337]:=
```
nv1 = {1, 0, –0.08333333333333334` };
nv2 = {0, 1, –0.3333333333333334` };
```

Note that

In[339]:=
```
Normalize [N[tangentVectorJMD [twCubic , p, {x, y, z}]]]
Normalize [Cross[nv1, nv2]]
```

Out[339]= {0.078811 , 0.315244 , 0.945732 }

Out[340]= {0.078811 , 0.315244 , 0.945732 }

give the same result. In the case of general n the analog of the cross product of $n-1$ rows is the last row orthogonal completion of these rows

In[342]:=
```
Orthogonalize [{nv1, nv2, RandomReal [{-1, 1}, 3]}] // MatrixForm
```

Out[342]//MatrixForm=
$$\begin{pmatrix} 0.996546 & 0. & -0.0830455 \\ -0.0261796 & 0.949011 & -0.314155 \\ -0.078811 & -0.315244 & -0.945732 \end{pmatrix}$$

This is the idea behind our algorithm

```
Options [tangentVectorMD ] = {tol → 1.*^-7 , ord → 4, hilbertFunction → True};
tangentVectorMD [F_, p_, X_, OptionsPattern []] := Module [{M2, n, pv, orth, J, hf},
    If[OptionValue [ord] < 2, Echo["ord must be at least 2"]; Abort[]];
    n = Length [X];
    {pv, M2} = nrref [macaulayMD [F, OptionValue [ord], p, X], OptionValue [tol]];
    If[AnyTrue [Flatten [Take [M2, All, 1]], Abs[#] > OptionValue [tol] &],
      Echo [p, "Not a solution , p = "];
      Return []];
    hf = hilbertFunctionMD [pv, OptionValue [ord], n];
    If[OptionValue [hilbertFunction ], Echo [hf, "Hilbert Function "]];
    If[hf[[OptionValue [ord] + 1]] == 0, Echo [ "point may be isolated ", "Warning "]];
    If[hf[[2]] == 1, Return [Take [Orthogonalize [
                Append [M2[[1 ;; n - 1, 2 ;; n + 1]], RandomReal [{-1, 1}, n]]], -1][[1]],
      Echo [p, "No unique tangent vector at "]];
    Null];
```

Example 2.3.3.2.1 continued

In[343]:=
```
tangentVectorMD [twCubic , p, {x, y, z}, ord → 2]
```

» Hilbert Function {1, 1, 1}

Out[343]=
```
{-0.078811 , -0.315244 , -0.945732 }
```

Increasing the order of the Macaulay matrix gives more of the Hilbert function. Since this is the accumulation of the local Hilbert function this stabilizes at the multiplicity. In this example we had a regular point so the multiplicity is 1. We could use this to calculate 2-dimensional singularities, note the first argument of tangentVectorMD is a set so even with one equation we need set braces.

Examples 2.3.3.2.2

In[345]:=
```
tangentVectorMD [{x y (x - y)}, {0, 0}, {x, y}]
```

» Hilbert Function {1, 2, 3, 3, 3}

» No unique tangent vector at {0, 0}

Compare with

In[346]:= singPointMult2D [x y (x – y), {0, 0}, x, y, dTol]

Out[346]= 3

Applying to a system with only isolated solutions

In[348]:= tangentVectorMD [{x z – y, y z – x ^ 2 – x, z ^ 2 – x ^ 2 – 1}, {0, 0, 1}, {x, y, z}]

» Hilbert Function {1, 1, 0, 0, 0}

» Warning point may be isolated

Out[348]= {–0.707107 , –0.707107 , 0.}

we get a tangent vector but it has multiplicity 0.

Going back to example 2.3.3 the cyclic-4 curve

In[350]:= C4 = {w + x + y + z, w x + x y + y z + z w, w x y + x y z + y z w + z w x, w x y z – 1};

In[351]:= tangentVectorMD [C4, {1, –1, –1, 1}, {w, x, y, z}]

» Hilbert Function {1, 2, 1, 1, 1}

» No unique tangent vector at {1, –1, –1, 1}

we have a singular point of multiplicity 1. We will explain later.

2.4 H-bases

We saw in Example 2.3.2.2.1 that there is a lack of a stopping point in the membership problem but is was suggested that an equation system could be modified so that only one step is needed. This is the main thrust of this section is to describe a type of equation system where this is true.

However there are infinitely many equations that any given curve, or more generally algebraic set, will satisfy. Several algorithms we will see generate a large number of these and we want to pick a good, but relatively small, equation set. The equation sets that satisfy the previous paragraph are good candidates for this.

Fortunately Macaulay in the same 1916 book where he described his Macaulay matrix did come up with an answer. He was using homogeneous equations for projective space and so called this an *H-basis*. Some authors use the name *Macaulay basis* for this. In this book , even though we recog - nize that algebraic curves live in projective space, prefer working in affine space as it is more algorithm friendly. It turns out that H-bases work fine in affine space too.

A system of polynomial equations in n variables is a *H-basis* if the member - ship problem can always be solved in one step. Specifically a system $F = \{f_1, f_2, \ldots, f_k\}$ of polynomials in n-variables is an H-basis if given any n-

variable polynomial g of degree d it is a polynomial combination of the polynomials of F if and only if there exist polynomials $\{g_1, \ldots, g_k\}$ so that

$$g_1 f_1 + g_2 f_2 + \cdots + g_k f_k = h \text{ with each } g_i f_i \text{ of total degree } \leq d$$

In particular suppose $k = 3$, f_1 is linear, f_2 is quadratic and f_3 is cubic. If h is linear then to be a polynomial combination of H-basis $F = \{f_1, f_2, f_3\}$ then h must be a constant times f_1. If h is quadratic it can be a linear times f_1 plus a constant times f_2. If h is cubic it can be a quadratic times f_1 plus a linear times f_2 plus a constant times f_3. And so on.

H-bases do exist and every polynomial system is a subset of an H-basis. We will see that Mathematica has a built-in algorithm **GroebnerBasis** to find one. This algorithm uses abstract algebra so we will not try to explain here how it works. A simple introduction to Gröbner bases is given at the beginning of the book by [Cox,Little and O'Shea] but unfortunately I do not know of an elementary exposition of H-bases that does not require lots of algebra. My position in this book has always been that any algorithm of Mathematica does not require my explanation. The big problem using **GroebnerBasis** is that this algorithm is intended for integer systems only. Mathematica will try to handle numerical systems but we can't rely on this working.

If F is a H-basis any larger system is also an H-basis. The trick is to find a small H-basis and in the rest of this section I will concentrate on this.

2.4.1 The algorithm hBasisMD

This algorithm (revised 5/2020) which takes a large polynomial system and attempt to find a small H-basis. It will not give an H-basis if the argument m is not large enough, unfortunately one cannot know what m is large enough in advance. One can check with **hBasisMDQ** below. The big advantage of this version of hBasisMD is that it works fine with numerical systems which will occur in applications.

There are essentially three steps in this algorithm. The first step is to calcu - late the Sylvester Matrix for the user given m and use the singular value decomposition to find a full rank row space. For numerical systems this essentially replaces the possibly numerically inconsistent input system with a least squares approximation of a consistent system. We then apply a reverse row reduction to find polynomials of small degree among polynomi - als combinations of the now consistent input system. This is the essential reason for H-bases. These first two steps are contained in the more general matrix reduction procedure **arref** below. The final, third, step is to return the resulting Sylvester matrix back into a polynomial system and use the mem - bership problem solution to reject polynomials which are already polyno -

mial combinations inside the vector space of polynomials of degree m or less of preceding accepted polynomials. The output is what is left after the rejections.

```
arref[M_, eps_] := Module[{p, P, j, r, s, R, mn, n, U, S, V},
    {U, S, V} = SingularValueDecomposition [N[M], Tolerance → eps];
    r = Length[Select[Diagonal[S], # > 0 &]]; (* rank *)
    R = Take[Transpose[V], r]; (* row space of M *)
    mn = Dimensions[R];
    n = mn[[2]];
    While[Norm[Take[R, All, {n}]] < eps, n--];
    p = {n};
    For[j = n - 1, j > 0, j--,
      If[mn[[1]] ≤ Length[p], Break[],
          p = Prepend[p, j];
          P = Check[R[[All, p]], Abort[]];
          s = Length[Select[SingularValueList [N[P], Tolerance → eps], # > 0 &]];
          If[s < Length[p],
            p = Drop[p, 1];, Null];
        ];
    ];
    P = R[[All, p]];
    {p, Chop[Check[Inverse[P], Abort[]].R]}]
```

The idea is that **arref** will allow us to pick out polynomial combinations of our input of lowest degrees. We look at a previous example:

Example 2.4.1.1. We consider example 2.3.2.2.1 where we found a linear polynomial that was a polynomial combination of a fourth and fifth degree polynomial.

In[116]:=
```
f1 = x + y - 2 z + y z^2 - z^4;
f2 = -x^2 + y - x y + 2 x z - z^2 - x y z^2 + x z^4;
```

We start by picking $m = 6$ and calculating the Sylvester matrix and its **arref** decomposition.

In[120]:=
```
S6 = sylvesterMD [{f1, f2}, 6, {x, y, z}];
{p6, A6} = arref[S6, dTol];
Length[p6]
```

Out[122]= 14

This last number says we have created 14 polynomial combinations of {f1,f2}. Lets look at the first 5

In[123]:=
```
Take[A6, 5].mExpsMD [6, {x, y, z}]
```

Out[123]= $\{-1.\, y + 1.\, z^2,\ -1.\, x\, y + 1.\, x\, z^2,\ -1.\, y^2 + 1.\, y\, z^2,\ -1.\, y\, z + 1.\, z^3,\ -1.\, x - 1.\, y - 1.\, y^2 + 2.\, z + 1.\, z^4\}$

In[130]:=
```
S7 = sylvesterMD [{f1, f2}, 7, {x, y, z}];
{p7, A7} = arref [S7, dTol];
Length [p7]
Take[A7, 5].mExpsMD [7, {x, y, z}]
```

Out[132]= 30

Out[133]= $\{-0.5\,x - 0.5\,y + 1.\,z,\ -1.\,y + 1.\,z^2,\ -1.\,x\,y + 1.\,x\,z^2,\ -1.\,y^2 + 1.\,y\,z^2,\ -1.\,y\,z + 1.\,z^3\}$

So we have produced our linear polynomial and can hope that $m = 7$ is large enough, that is that from these 30 polynomial combinations we can get all polynomial combinations of {f1,f2} without relying on cancellation of terms to do our work.

So the algorithm **hBasisMD** creates a list of polynomial combinations $\mathcal{A}$ using **arref** that we hope is a building block for all polynomial combinations of our input system. The first entry of $\mathcal{A}$ is becomes an element of our proposed H-Basis H and we proceed to go down the list $\mathcal{A}$ using our membership problem method to test if it is a polynomial combination of the previous choices. If not we add it to our list H. So we hypothesize that every polynomial combination of our input system is an appropriate combination of polynomials in the list $\mathcal{A}$ which in turn are appropriate combinations of our list H. The code follows:

In[135]:=
```
hBasisMD[F_, m_, X_, tol_] := Module[{n, p, S, A, a, Sa, H, H1, r, s, s1, k, temp},
  n = Length[X];
  H = {};
  H1 = {};
  S = sylvesterMD[F, m, X];
  {p, A} = arref[S, tol];
  Echo[hilbertFunctionMD[p, m, n], "Initial Hilbert Function "];
  r = Length[p];
  H1 = {A[[1]].mExpsMD[m, X]};
  H = H1;
  S = sylvesterMD[H, m, X];
  s = Length[SingularValueList[S, Tolerance → tol]];
  k = 2;
  While[k ≤ r,
   H1 = Append[H, A[[k]].mExpsMD[m, X]];
   Sa = sylvesterMD[H1, m, X];
   s1 = Length[SingularValueList[Sa, Tolerance → tol]];
   If[s1 > s, s = s1; H = H1];
   If[r > 30 && Mod[k, 10] == 0,
    temp = PrintTemporary["hBasis :: At equation ", k, " of ", r];
    Pause[3];
    NotebookDelete[temp]];
   k++];
  S = sylvesterMD[H, m + 1, X];
  {p, A} = arref[S, tol];
  Echo[hilbertFunctionMD[p, m, n], "Final Hilbert Function "];
  H];
```

Although this code is fairly simple we are finding the rank of increasingly large matrices. This can take a long time. A new feature (5/2020) is if **A** has many rows then the procedure gives temporary output of progress. This could give the user a chance to abort the procedure if the user does not wish to wait. Unlike previous versions there are no options available. A global Hilbert function of the original system and the H-basis are given, ideally the second Hilbert function will stabilize. If not you may wish to try a larger m or to use the algorithm hBasisMDQ below to test the output of this algorithm.

Example 2.4.1.1 Continued. We use the algorithm to calculate a H-basis for {f1,f2}.

In[136]:=
```
Timing[hBasisMD[{f1, f2}, 7, {x, y, z}, dTol]]
```

» Initial Hilbert Function {1, 2, 5, 7, 9, 18, 22, 26}

» Final Hilbert Function {1, 2, 2, 2, 2, 2, 2, 2}

Out[136]= $\{44.5493, \{-0.5\,x - 0.5\,y + 1.\,z, -1.\,y + 1.\,z^2\}\}$

2.4.2 hBasisMDQ

Typically **hBasisMD** is used as a subroutine for other algorithms which return a large set of polynomials to get a smaller set, not necessarily an actual H-Basis. This will terminate with a warning message if some of the input polynomials have degree greater than m. Then the one thing that should always happen is that the set H returned will at least generate the input set, so even if one does not get an H-basis something useful is returned.

But one will not get an H-basis if two small an m is used. There is no easy a-priori method to guess a large enough m but the algorithm in this subsec - tion should be able to check to see if you do have an H-basis.

So you can use the output from **hBasisMD** in **hBasisMDQ**. If using **hBasisMD** as a stand-alone procedure you may wish to run **hBasisMDQ** first to see if you already have an H-Basis and to get a value of m that should work.

hBasisMDQ works by comparing the input system with a known H-Basis. By default this the output of the built-in Mathematica function **GroebnerBasis** with option **MonomialOrder→DegreeLexicographic.** As mentioned before this is not cheating the reader as I have never promised to explain built-in functions, only my own. Normally Gröbner Bases only work for systems with integer coefficients, Mathematica's will attempt numerical systems but I offer no guarantees. In particular **GroebnerBasis** will flag inconsistent systems and abort. An over-determined numerical system that may be fine in other places in this book may look inconsistent to **GroebnerBasis.**

The syntax is **hBasisMDQ[F,H,X,tol]** where F is your known system, H is the system you wish to check to see if it is an H-basis. As usual X is the variable set, and tol is desired tolerance. **Note that *m* is not used as input** so one does not need to know m beforehand. Here, especially, the order of the variables matter, *Lexicographic* is respect to the order in X for instance the built in **MonomialList** used in **GroebnerBasis** returns a different list depending on the way the variables are listed:

In[123]:=
```
MonomialList[(x+y+z)^3, {x, y, z}, "DegreeLexicographic"]
MonomialList[(x+y+z)^3, {z, y, x}, "DegreeLexicographic"]
```

Out[123]= $\{x^3, 3x^2y, 3x^2z, 3xy^2, 6xyz, 3xz^2, y^3, 3y^2z, 3yz^2, z^3\}$

Out[124]= $\{z^3, 3yz^2, 3xz^2, 3y^2z, 6xyz, 3x^2z, y^3, 3xy^2, 3x^2y, x^3\}$

As an option **hBasisMDQ** will treat the first argument **F** as a known H-basis and check the argument **H** against that. This could be useful, for example, if one has an H-basis but is concerned that it is not minimal. This may be, for instance, the case for the H-basis returned by **GroebnerBasis.**

Our function **hBasisMDQ** gives an information notice with the size of the Gröbner Basis and a list of total degrees of polynomials present. In the case above where the option **useF→True** this information refers to **F** rather than the Gröbner Basis which is not calculated. If it is determined that H is a Gröbner basis the the procedure returns only the value **True.** Otherwise it stops at the first instance an element of **F** is not expressible in terms of the polynomials in **H.** If the Gröbner Basis (or optionally **F**) has polynomials of degree 1 but **H** does not then **hBasisMDQ** flags that fact and stops, doing no calculations. Otherwise it gives the degree of the missing polynomial and which polynomial of the Gröbner Basis in that degree it is and halts. For the user's convenience this routine creates a global variable lastHBGroebner so this polynomial can be retrieved.

Using the last sentence above the user could use **hBasisMDQ** perhaps several times to find a minimal H-basis from the Gröbner basis, but if the Gröbner basis is large probably it is better to use hBasisMD with the m given by the largest degree in the Gröbner basis.

The procedure hBasisMDQ works by running the membership test above on each member of the Gröbner basis, or optionally the H-basis **F**. Here is the code

```
Options[hBasisMDQ] = {useF → False};
hBasisMDQ[F_, H_, X_, tol_, OptionsPattern[]] :=
  Module[{G, m, j, degG, degH, selG, SH, SG, r1, r2},
    G = If[OptionValue[useF], G = F,
        G = GroebnerBasis[F, X, MonomialOrder → DegreeLexicographic]];
    m = Max[tDegMD[#, X] &/@ G];
    degG = Sort[DeleteDuplicates[tDegMD[#, X] &/@ G]];
    G = SortBy[G, tDegMD[#, X] &];
    lastHBGroebner = G;
    If[MemberQ[degG, 0], Echo["F not proper ideal"]; Return[False]];
    Echo[{Length[G], degG}, "{size of Groebner Basis, degrees}"];
    degH = Sort[DeleteDuplicates[tDegMD[#, X] &/@ H]];
    If[degG[[1]] < degH[[1]], Echo[degG[[1]], "No poly in H of degree "];
      Return[False]];
    Catch[Do[SH = sylvesterMD[Select[H, tDegMD[#, X] ≤ k &], k, X];
        r1 = Length[SingularValueList[N[SH], Tolerance → tol]];
        r2 = r1;
        selG = Select[G, tDegMD[#, X] == k &];
        j = Length[selG];
        i = 0;
        While[r1 == r2 && i < j,
          i++;
          SG = sylMD[selG[[i]], k, X];
          r2 = Length[SingularValueList[N[Join[SH, SG]], Tolerance → tol]]];
        If[r1 == r2, Continue[], Echo[{k, i}, "Problem at poly i degree k "];
          Throw[Return[False]]],
        {k, degG}]];
    True]
```

Gröbner bases may be large so this routine could take a long time to run, especially if **H** is an h-Basis. But since it stops at the first omission this version does not give running information. Again, this could give a false negative in the numerical case, but a return of **True** should be reliable.

Example 2.4.2.1 , see 2.4.1.1

In[140]:=
```
f1 = x + y - 2 z + y z^2 - z^4;
f2 = -x^2 + y - x y + 2 x z - z^2 - x y z^2 + x z^4;
```

In[142]:=
```
hBasisMDQ[{f1, f2}, {f1, f2}, {x, y, z}, dTol]
```

» {size of Groebner Basis, degrees} {2, {1, 2}}

» No poly in F of degree 1

Out[142]= False

Adding the previously known linear polynomial

In[150]:= **hBasisMDQ [{f1, f2}, {f1, f2, x + y – 2 z}, {x, y, z}, dTol]**

» {size of Groebner Basis, degrees} {2, {1, 2}}

» Problem at {degree, poly} {2, 1}

Out[150]= False

We see what we need from

In[145]:= **lastHBGroebner**

Out[145]= $\{x + y - 2\,z,\ y - z^2\}$

Example 2.4.2.2 Twisted Cubic (Section 2.1)

Consider the twisted Cubic first as a naive curve

In[151]:= **tw2 = {y – x ^ 2, z – x ^ 3};**
hBasisMDQ [tw2, tw2, {x, y, z}, dTol]

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

» Problem at {degree, poly} {2, 2}

Out[152]= False

This is not an H-basis. So even without the geometric input of section 2.1 we need additional/dif - ferent equations. The suggestion is

In[153]:= **lastHBGroebner**

Out[153]= $\{x^2 - y,\ x\,y - z,\ -y^2 + x\,z,\ y^3 - z^2\}$

But even this is bigger than necessary

In[154]:= **hBasisMDQ [{x ^ 2 – y, x y – z, x z – y ^ 2}, {x ^ 2 – y, x y – z, x z – y ^ 2}, {x, y, z}, dTol]**

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

Out[154]= True

So the basis we found in 2.1 of thee quadratics is sufficient as an H - basis.

2.4.3 Application: Making slightly inconsistent numerical systems consistent.

The following example of 4 linear equations in 4 unknowns is motivated by an example at the end of section 3.2.

```
In[330]:= lsys = {0.277262174208273` + 0.5144436627966619` x +
      0.10598605713201434` y + 0.8045124985078014` z,
    0.7202433507070195` - 0.626115747655119` x + 0.27531975154910765` y +
      0.1158776108984591` z, -0.258819002786987` - 0.3361601677996709` x -
      0.0989359825030034` y + 0.9001226231727609` z, 0.4685805085964169` -
      0.849601020102557` x + 0.17911927833946` y - 0.16287018674812506` z}

Out[330]= {0.277262 + 0.514444 x + 0.105986 y + 0.804512 z, 0.720243 - 0.626116 x + 0.27532 y + 0.115878 z,
   -0.258819 - 0.33616 x - 0.098936 y + 0.900123 z, 0.468581 - 0.849601 x + 0.179119 y - 0.16287 z}

In[331]:= NSolve[lsys]

Out[331]= {}
```

So this system in inconsistent. But

```
In[332]:= hsys = hBasisMD[lsys, 1, {x, y, z}, 1.*^-8]
```

» Initial Hilbert Function {1, 0}

» Final Hilbert Function {1, 0}

```
Out[332]= {1. x, 2.61602 + 1. y, 1. z}

In[333]:= {x, y, z} /. NSolve[hsys]

Out[333]= {{0., -2.61602, 0.}}
```

is consistent.

2.5 Duality, Union , Intersection and decomposition of Curves.

Already in 1916 Macaulay talked about the dual to his Macaulay matrix. Duality will play a small but important technical role in our considerations.

2.5.1 Duality

Given a matrix, generally a Macaulay or Sylvester matrix, M the *dual matrix* is a matrix $\bar{D}$ with independent columns with the property that $M.\bar{D} = 0$ where here 0 represents the zero matrix of the appropriate size. Essentially a dual matrix of M is just a matrix whose columns give a basis for the null space of M. We will assume our matrix has numerical entries so instead of using the built in **NullSpace** procedure we choose a tolerance and use the following, see for example Appendix 1 of my curve theory book.

In[31]:=
```
dualMatrix[A_, tol_] := Module[{ns, r, c, U, S, V},
    c = Dimensions[A][[2]];
    {U, S, V} = SingularValueDecomposition[N[A], Tolerance → tol];
    r = Length[Select[Diagonal[S], # > 0 &]];
    Take[V, All, r - c]]
```

Example 2.5.1.1 Consider the matrix

In[142]:=
```
M = RandomReal[{-1, 1}, {3, 5}];
M // MatrixForm
```

Out[143]//MatrixForm=
$$\begin{pmatrix} -0.739775 & -0.276164 & 0.648798 & 0.524576 & -0.542521 \\ 0.578205 & -0.88494 & 0.0673685 & -0.636653 & 0.309926 \\ -0.900584 & 0.355809 & 0.975681 & -0.0357853 & -0.0451929 \end{pmatrix}$$

In[146]:=
```
𝒟1 = NullSpace[M];
𝒟1 // MatrixForm
𝒟2 = dualMatrix[M, dTol];
𝒟2 // MatrixForm
```

Out[147]//MatrixForm=
$$\begin{pmatrix} 0.540947 & 0.106097 & 0.497849 & 0.568587 & 0.353518 \\ -0.468768 & -0.28463 & -0.284309 & 0.294731 & 0.729072 \end{pmatrix}$$

Out[149]//MatrixForm=
$$\begin{pmatrix} 0.540947 & -0.468768 \\ 0.106097 & -0.28463 \\ 0.497849 & -0.284309 \\ 0.568587 & 0.294731 \\ 0.353518 & 0.729072 \end{pmatrix}$$

In this case the difference is that **dualMatrix** gives a column vector rather than giving the nullspace basis as rows. If we had used an integer matrix then we would have

In[159]:=
```
A = RandomInteger[{-9, 9}, {3, 5}];
A // MatrixForm
𝔻1 = NullSpace[A]; 𝔻1 // MatrixForm
𝔻2 = dualMatrix[A, dTol]; 𝔻2 // MatrixForm
𝔻3 = Transpose[NullSpace[N[A]]]; 𝔻3 // MatrixForm
```

Out[160]//MatrixForm=
$$\begin{pmatrix} 8 & -4 & -1 & 2 & 3 \\ -5 & -5 & 2 & 3 & -5 \\ 4 & 9 & 3 & 1 & 8 \end{pmatrix}$$

Out[161]//MatrixForm=
$$\begin{pmatrix} -218 & -159 & -115 & 0 & 331 \\ -70 & 119 & -374 & 331 & 0 \end{pmatrix}$$

Out[162]//MatrixForm=
$$\begin{pmatrix} -0.0879398 & -0.487715 \\ 0.26806 & -0.380637 \\ -0.704857 & -0.20789 \\ 0.646603 & -0.0485011 \\ -0.0741015 & 0.756095 \end{pmatrix}$$

Out[163]//MatrixForm=
$$\begin{pmatrix} -0.0879398 & -0.487715 \\ 0.26806 & -0.380637 \\ -0.704857 & -0.20789 \\ 0.646603 & -0.0485011 \\ -0.0741015 & 0.756095 \end{pmatrix}$$

So we see that for small well conditioned matrices we could use the formula

```
Transpose[Nullspace[N[M]]]
```

instead of **dualMatrix.**

On the other hand, the *left dual* space $\mathcal{L}$ of M is the matrix with independent rows with $\mathcal{L}.M = 0$. I have been calling it the localDualMatrix given by

```
localDualMatrix[A_, tol_] := Transpose[dualMatrix[Transpose[A], tol]];
```

Note that this is properly a row matrix, that is the rows form a basis for the left null space.

Traditionally the dual of the Macaulay matrix was considered to be a space of differentials describing the local structure. The left (or local) dual of this should recover our original. We will typically be interested here in the dual of the Sylvester Matrix with the left dual of that recovering our curve.

Example 2.5.1.2 Consider the twisted cubic.

In[164]:=
```
twCubic = {x^2 - y, x y - z, x z - y^2}
```

Out[164]=
$$\{x^2 - y,\ x\,y - z,\ -y^2 + x\,z\}$$

In[180]:=
```
S2 = sylvesterMD[twCubic, 2, {x, y, z}]
D2 = dualMatrix[S2, dTol]
```

Out[180]= {{0, 0, −1, 0, 1, 0, 0, 0, 0, 0}, {0, 0, 0, −1, 0, 1, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 1, −1, 0, 0}}

Out[181]= {{0., −0.707107, 0., −0.5, 0.5, 0., 0.}, {0.707107, 0., −0.707107, 0., 0., 0., 0.},
{0., 0.5, 0., −0.353553, 0.353553, 0., 0.}, {0.5, 0., 0.5, 0., 0., 0., 0.},
{0., 0.5, 0., −0.353553, 0.353553, 0., 0.}, {0.5, 0., 0.5, 0., 0., 0., 0.}, {0., 0., 0., 0.5, 0.5, 0., 0.},
{0., 0., 0., 0.5, 0.5, 0., 0.}, {0., 0., 0., 0., 0., 1., 0.}, {0., 0., 0., 0., 0., 0., 1.}}

This doesn't mean much to us. Now take the local dual of this

In[182]:=
```
LD2 = localDualMatrix[D2, dTol]
```

Out[182]= {{−1.38778 × 10^−16, 1.51669 × 10^−16, 0.43613, −0.244521, −0.43613, 0.244521, −0.5, 0.5, 0., 0.},
{−1.11022 × 10^−16, −3.07488 × 10^−16, 0.345805, 0.616781,
−0.345805, −0.616781, −1.73672 × 10^−17, −9.28866 × 10^−17, 0., 0.},
{6.93889 × 10^−17, −1.59101 × 10^−16, −0.43613, 0.244521, 0.43613, −0.244521, −0.5, 0.5, 0., 0.}}

But

In[183]:=
```
F = Chop[LD2].mExpsMD[2, {x, y, z}]
```

Out[183]= {−0.43613 x^2 + 0.43613 y + 0.244521 x y + 0.5 y^2 − 0.244521 z − 0.5 x z,
−0.345805 x^2 + 0.345805 y − 0.616781 x y + 0.616781 z,
0.43613 x^2 − 0.43613 y − 0.244521 x y + 0.5 y^2 + 0.244521 z − 0.5 x z}

is actually another system for the twisted cubic. But note

In[184]:=
```
hBasisMD[F, 2, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 3}

» Final Hilbert Function {1, 3, 3}

Out[184]= {1. x^2 − 1. y, 1. x y − 1. z, 1. y^2 − 1. x z}

is our original system! This is why H-bases and our **hBasisMD** are so useful.

2.5.3 Intersection and Union of curves.

The intersection of two curves is typically a point set. But to find the equation set one simply combines the two equations. For the twisted cubic system above we noticed in Section 2.1 that the naive curves $\{x^2 - y,\ x\,y - z\}$ and $\{x\,y - z,\ y^2 - x\,z\}$ each have an extra line but these extra lines are different so the intersection $\{x^2 - y,\ x\,y - z,\ y^2 - x\,z\}$ gives just the twisted cubic without the extra lines.

The union of two space curves is more difficult. For plane curves we simply multiplied the equations. But in space we have several equations for each.

The trick is to go to duals, duality takes unions to intersections and vice versa. So we take the dual matrices of appropriate Sylvester matrices and then join these, note same m. Then we take the local dual of the combined dual matrix. The question is how big do we make the matrices. Here the idea of H-bases helps. We make sure each Sylvester matrix is large enough to contain an H-basis. And at the end we give the result as an H-Basis.

Example 2.5.3.1: We will take the union of a line and the twisted cubic for a relatively easy but non-trivial example starting from H-bases (recommended).

```
In[133]:= twc = {x^2 - y, x y - z, y^2 - x z};
          ln = lineMD[{-1, 1, -1}, {2, 4, 8}, {x, y, z}]
```

```
Out[134]= {0.12738 - 0.764319 x - 0.477694 y + 0.414004 z, 0.818223 + 0.398339 x - 0.414498 y + 0.00538627 z}
```

We don't show the intermediate matrices but we do give their dimensions. First we calculate duals of the Sylvester matrices

```
In[135]:= Dtwc = dualMatrix[sylvesterMD[twc, 3, {x, y, z}], dTol];
          Dimensions[Dtwc]
          Dln = dualMatrix[sylvesterMD[ln, 3, {x, y, z}], dTol];
          Dimensions[Dln]
```

```
Out[136]= {20, 10}
```

```
Out[138]= {20, 4}
```

Join these column wise to get the dual of the union.

```
In[139]:= dualF = Join[Dtwc, Dln, 2];
          Dimensions[dualF]
```

```
Out[140]= {20, 14}
```

Finally take the localDual and reduce by **hBasisMD**.

```
In[144]:= Fraw = localDualMatrix[dualF, dTol].mExpsMD[3, {x, y, z}];
          Length[Fraw]
          F = hBasisMD[Fraw, 3, {x, y, z}, dTol]
```

```
Out[145]= 8
```

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

» Final Hilbert Function {1, 3, 4, 4}

```
Out[146]= {-1. x^2 + 1. y + 1. x y - 1. z, 2. x^2 - 2. y + 1. y^2 - 1. x z}
```

Even though the twisted cubic is not a naive curve, the union is, in fact this is a quadratic surface intersection curve (QSIC), see section 3.2. An unintended feature of my **hBasisMD** function is that even though the line was given by numeric equations the end result is integer! One could have

exploited that immediately at the input level

In[150]:=
```
hBasisMD [ln, 2, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 1, 1}

» Final Hilbert Function {1, 1, 1}

Out[150]=
```
{-2. - 1. x + 1. y, -2. - 3. x + 1. z}
```

We will look at this example again. For now note that we could also calculate the intersection.

In[148]:=
```
NSolve [Join [twc, ln]]
```

Out[148]=
```
{}
```

But this is actually wrong, Mathematica does not like numerical systems of 5 equations in 3 unknowns! Using exact representations

In[192]:=
```
{x, y, z} /. NSolve[Join[twc, {-2 - x + y, -2 - 3 x + z}]]
```

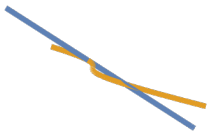
Out[192]=
```
{{2., 4., 8.}, {-1., 1., -1.}}
```

We might expect a third point since we are intersecting a cubic and a line, but it is a well known fact that no 3 points on the twisted cubic are co-linear [see Harris].

Note it is easy to plot this curve since both components are parametic

In[182]:=
```
ParametricPlot3D[{{-1 + 3 t, 1 + 3 t, -1 + 9 t}, {t, t ^ 2, t ^ 3}},
  {t, -2, 3}, ImageSize → Small, Boxed → False, Axes → False]
```

Out[182]=



Example 2.5.3.2: Another simple example: three lines.

One can go on for a long time constructing equations systems for unions of lines in space, see for example my paper on *[Numeric Lines]*.

In[195]:=
```
l1 = {x, y};
l2 = {x, z};
l3 = {z, y - 1};
```

In[198]:=
```
Dl1 = dualMatrix [sylvesterMD [l1, 3, {x, y, z}], dTol];
Dl2 = dualMatrix [sylvesterMD [l2, 3, {x, y, z}], dTol];
Dl3 = dualMatrix [sylvesterMD [l3, 3, {x, y, z}], dTol];
DG = Join[Dl1, Dl2, Dl3, 2];
Dimensions [DG]
```

Out[202]=
```
{20, 12}
```

```
In[203]:= Graw = localDualMatrix [DG, dTol].mExpsMD [3, {x, y, z}];
          Dimensions [Graw]
Out[204]= {10}

In[205]:= hBasisMD [Graw, 4, {x, y, z}, dTol]
```

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

» Final Hilbert Function {1, 3, 3, 3, 3}

```
Out[205]= {-1. x + 1. x y, 1. x z, 1. y z}
```

So this is the intersection of 3 quadric surfaces. In Section 3.2 below we study the classification of curves given as the intersection of 2 quadric surfaces, QSIC, and although there are examples with three lines, this shows that not all unions of 3 lines in $\mathbb{R}^3$ are QSIC.

2.5.3.3 Here is one more example relevant to Section 3.2

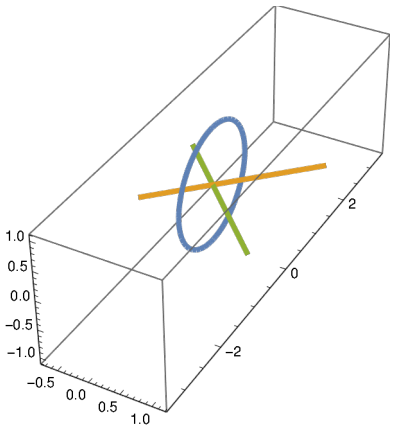
```
In[117]:= q1 = {x, y ^ 2 + z ^ 2 - 1};
          q2 = {z, x - y};
          q3 = {z, x + y};

In[123]:= Dq1 = dualMatrix [sylvesterMD [q1, 4, {x, y, z}], dTol];
          Dq2 = dualMatrix [sylvesterMD [q2, 4, {x, y, z}], dTol];
          Dq3 = dualMatrix [sylvesterMD [q3, 4, {x, y, z}], dTol];
          DQ = Join[Dq1, Dq2, Dq3, 2];
          Dimensions [DQ]
Out[127]= {35, 19}

In[128]:= Qraw = localDualMatrix [DQ, dTol].mExpsMD [4, {x, y, z}];
          Length [Qraw]
Out[129]= 17

In[130]:= Q = hBasisMD [Qraw, 4, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 5, 5, 4}

» Final Hilbert Function {1, 3, 5, 5, 4}

```
Out[130]= {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}
```

Since all the pieces can be parameterized it is easy to plot. Again this looks similar to a QSIC but is not a QSIC. [See C. Tu, W. Wang, B. Mourrain, J. Wang case numbers 23-26]

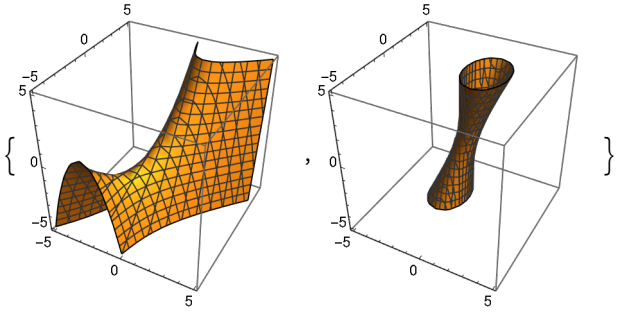


2.5.4 Decomposition of reducible curves.

Unlike the plane case where the single equation of a reducible curve factors, possibly with irrational complex coefficients, the equation system for a reducible space curve, see our examples in the previous section, do not factor. For Example 2.5.3.1 the two equations are give smooth quadric surfaces and thus not factorable.

In[186]:=
```
{ContourPlot3D[−x^2+y+xy−z == 0, {x, −5, 5}, {y, −5, 5}, {z, −5, 5}, ImageSize → Small],
 ContourPlot3D[{2x^2 − 2y + y^2 − xz == 0},
  {x, −5, 5}, {y, −5, 5}, {z, −5, 5}, ImageSize → Small]}
```

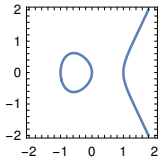
Out[186]=



It is important not to confuse topological components with algebraic compo - nents. For plane curves the simple example

In[117]:=
```
ContourPlot[y^2 == x^3 − x, {x, −2, 2}, {y, −2, 2}, ImageSize → Tiny]
```

Out[117]=



show two topological components but this curve is irreducible. We will

have plenty of examples like this for space curves later.

Another big difference between plane curves and space curves is the the plane Bézout theorem says that a reducible curve with components of degree d_1, d_2 will have $d_1\, d_2$ singular intersection points, possibly one of d_1, d_2 could be 1. We saw in the plane curve books that if this number is large enough we can even use these points to factor. But reducible space curves could have no singular points at all, for example a curve consisting of two skew lines.

Without fully describing a space curve the only sure way to test for irreducibility is to use one of the higher powered solvers such as [PHCpack] or Bertini [Bates, Hauenstein, Sommese]. I give my solution to this problem below but it may require plotting the curve first using methods later in the book.

2.5.4.1 Dual Interpolation

We saw in Section 2.5.3 that duality takes unions to intersections, that is the duals of components can have separate rows in the dual matrix. We exploit this by considering the dual matrix of the curve and attempting to find equations describing a given component. But first we need a technical subroutine.

To try to explain, in principle the Sylvester Matrix of high enough order contains all the information necessary to determine the curve. One property of a curve is the Macaulay information at a point. Of could recover the equation, perhaps using duality and **hBasisMD** and take the Taylor series at that point which can be used to do a hand calculation of the Macaulay matrix. Or figure out how this works within the dual matrix. At one point your author did this in general but don't ever ask him to show his work but the answer is encoded in this Mathematica procedure.

```
c2zMD[q_, n_] := Module[{m, Tn, ss, bi, bj, r1, C, s, pow},
    pow[a_, m_] := If[m ≤ 0, 1, a^m];
    s = Length[q];
    Tn = expsMD[s, n];
    ss = Length[Tn];
    ss = Length[Tn];
    C = {};
    Do[bj = Tn[[j]];
      C = Append[C,
          Table[Product[Binomial[Tn[[i]][[k]], bj[[k]]] * pow[q[[k]], (Tn[[i]][[k]] - bj[[k]])], {k, s}],
            {i, ss}]],
      {j, ss}];
    Transpose[C]]
```

The following example gives some idea how this might work.

Example 2.5.4.1.1 See Example 2.5.3.1 the union of a line and twisted cubic.

In[217]:=
```
F = {-x^2 + y + x y - z, 2 x^2 - 2 y + y^2 - x z};
p = {1, 1, 1};
```

I start with the answer, the Macaulay matrix at this point. Since this is a regular point the interesting part of this is the first two rows. Since the two equations become separated we look at the equivalent **nrref** form.

In[253]:=
```
Take[nrref[macaulayMD[F, 2, p, {x, y, z}], dTol][[2]], 2, 4] // MatrixForm
```

Out[253]//MatrixForm=
$$\begin{pmatrix} 0 & 1. & 0 & -0.333333 \\ 0 & 0 & 1. & -0.666667 \end{pmatrix}$$

Now I show how to recover this from the Sylvester Matrix using my procedure **c2zMD** above.

In[255]:=
```
S2 = sylvesterMD[F, 2, {x, y, z}];
DS2 = dualMatrix[S2, dTol];
ICDS2 = Inverse[c2zMD[p, 2]].DS2;
Take[nrref[localDualMatrix[ICDS2, dTol], dTol][[2]], 2, 4] // MatrixForm
```

Out[258]//MatrixForm=
$$\begin{pmatrix} 0 & 1. & 0 & -0.333333 \\ 0 & 0 & 1. & -0.666667 \end{pmatrix}$$

Incidentally this example somewhat explains why I called the left dual the *local dual,* it gives local information.

For our problem the point is that this is sort of reversible. We go back to the original Macaulay matrix and up the order to 4.

In[298]:=
```
DM = dualMatrix[macaulayMD[F, 4, p, {x, y, z}], dTol];
CDM = c2zMD[p, 4].DM;
LCDM = localDualMatrix[CDM, dTol].mExpsMD[4, {x, y, z}];
hBasisMD[LCDM, 4, {x, y, z}, dTol]
```

» Initial Hilbert Function {1, 3, 1, 0, 0}

» Final Hilbert Function {1, 3, 1, 0, 0}

Out[301]=
$$\{1.x^2 - 1.y,\ 1.xy - 1.z,\ 1.y^2 - 1.xz,\ -1. + 5.x - 10.y + 10.z - 5.xz + 1.yz,\ -5. + 24.x - 45.y + 40.z - 15.xz + 1.z^2\}$$

We don' t quite get the original system but the surprise is the first 3 equations define the twisted cubic, not the union F which was the only input data. This is because we started with a Macaulay matrix which gives only local information at the point $p = \{1, 1, 1\}$and doesn't see the line. We would get better results if we used additional points on the twisted cubic and/or higher order. So this will give us our algorithm for finding equations of irreducible components of reducible curves.

In[87]:=
```
Options[dualInterpolationMD ] := {hBasis → True}
dualInterpolationMD [F_, P_, m_, X_, tol_, OptionsPattern []] :=
  Module[{M, DM, DSi, DS, S, G, i, np},
    np = Length[P];
    DS = {{}};
    For[i = 1, i ≤ np, i++,
      M = macaulayMD [F, m, P⟦i⟧, X];
      DM = dualMatrix [M, tol];
      DSi = c2zMD[P⟦i⟧, m].DM;
      DS = Join[DS, DSi, 2]];
    S = localDualMatrix [DS, tol];
    If[Dimensions [S]⟦1⟧ == 0, Print["no curve, try larger m"]; Abort[]];
    G = S.mExpsMD [m, X];
    If[OptionValue [hBasis], Return[Chop[hBasisMD [G, m, X, tol], tol]], Return[G]];
  ]
```

Example 2.5.4.1.1 Continued

In[117]:=
```
F = {-x^2 + y + x y - z, 2 x^2 - 2 y + y^2 - x z};
P = {{0, 0, 0}, {.5, .25, .125}, {1, 1, 1}}
dualInterpolationMD [F, P, 4, {x, y, z}, 1.*^-10 ]
```

Out[118]= {{0, 0, 0}, {0.5, 0.25, 0.125}, {1, 1, 1}}

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

» Final Hilbert Function {1, 3, 3, 3, 3}

Out[119]= $\{1.\,x^2 - 1.\,y,\ 1.\,x\,y - 1.\,z,\ 1.\,y^2 - 1.\,x\,z\}$

This is our standard H - basis for the twisted cubic.

Here are two points on the line

```
In[131]:= q1 = N[{1/2, 5/2, 7/2}];
          q2 = N[{-1/4, 7/4, 5/4}];
```

```
In[133]:= Q = {q1, q2}
```

Out[133]= {{0.5, 2.5, 3.5}, {−0.25, 1.75, 1.25}}

```
In[134]:= dualInterpolationMD [F, Q, 2, {x, y, z}, 1.*^-10 ]
```

» Initial Hilbert Function {1, 1, 1}

» Final Hilbert Function {1, 1, 1}

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

Which is an H-basis for our line.

Example 2.5.4.1.2 A slightly more difficult example is Example 2.5.3.3. One component is the circle in the plane $x = 0$.

```
In[140]:= G = {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};
          P2 = N[{{0, 1, 0}, {0, 0, 1}, {0, Sqrt[2]/2, Sqrt[2]/2}}]
```

Out[141]= {{0., 1., 0.}, {0., 0., 1.}, {0., 0.707107 , 0.707107 }}

```
In[142]:= dualInterpolationMD [G, P2, 4, {x, y, z}, 1.*^-10 ]
```

» Initial Hilbert Function {1, 2, 2, 2, 2}

» Final Hilbert Function {1, 2, 2, 2, 2}

Out[142]= $\{1.\,x,\ -1. + 1.\,y^2 + 1.\,z^2\}$

2.6 Fractional Linear Transformations

We come to our most important procedure in this book. We have already introduced Mathematica's **TransformationFunction** which is otherwise known as a *projective linear transformation* or *linear fractional transformation.* As the reader is well aware your author prefers the name *fractional linear transformation, FLT.* These transformations can have any dimensional domain and range and are given by *transformation matrices* which are $(n+1)\times(k+1)$ matrices where the transformation goes from $\mathbb{R}^k \longrightarrow \mathbb{R}^n$.

Possibly they could be complex as well. Thus an example $\mathbb{R}^4 \longrightarrow \mathbb{R}^2$ could be

Example 2.6.0.1

```
In[151]:= A = RandomInteger [{-9, 9}, {3, 5}];
In[151]:= A = {{7, 8, -4, 6, -8}, {9, -2, -6, 0, -2}, {9, 6, -3, 7, 2}};
In[155]:= A // MatrixForm
```

$$\text{Out[155]//MatrixForm=} \quad \begin{pmatrix} 7 & 8 & -4 & 6 & -8 \\ 9 & -2 & -6 & 0 & -2 \\ 9 & 6 & -3 & 7 & 2 \end{pmatrix}$$

```
In[153]:= TransformationFunction [A][{w, x, y, z}]
```

$$\text{Out[153]=} \quad \left\{\frac{-8+7w+8x-4y+6z}{2+9w+6x-3y+7z}, \frac{-2+9w-2x-6y}{2+9w+6x-3y+7z}\right\}$$

I also have alternate notation

```
In[154]:= fltMD[{w, x, y, z}, A]
```

$$\text{Out[154]=} \quad \left\{\frac{-8+7w+8x-4y+6z}{2+9w+6x-3y+7z}, \frac{-2+9w-2x-6y}{2+9w+6x-3y+7z}\right\}$$

In my Plane Curve Book and Chapter 1 of this book I restrict to invertible square transformation functions and give also corresponding functions **FLT2D, FLT3D** which take equations to equations. This makes these much more useful. Actually **FLT3D** will work for any dimension n as long as the transfor - mation matrix is invertible. These work equation by equation by simply composing each equation with the inverse transformation.

In the general case, however, we don't have an inverse transformation and the number of equations in the range may be more or fewer than equations in the domain. In the example above a curve in $\mathbb{R}^4$ would have 3 or more equations but a curve in $\mathbb{R}^2$ has only one. Thus many of the techniques we have introduced in this chapter, in particular Sylvester matrices, duality and H-bases, will be used.

The key is, as in **FLT2D, FLT3D**, is that the transformation of equations works naturally in the opposite direction as the transformation of points. But duality turns this around: *the transform of dual spaces works in the same direction as the transformation of points.* The other thing is we will have to deal with is the fact that these transformations are actually transformations of projective space so we will need to work with homogeneous polynomials. Then these FLT will be simply linear transformations rather than rational functions. We will need the following simple subroutines

In[48]:=
```
fVecMD[f_, m_, X_] := Module[{n, FA, d},
    n = Length[X];
    {FA, d} = fAssocMD[f, X];
    Values[shiftFAMD[FA, 0, m]]]
fMatMD[F_, m_, X_] := Table[fVecMD[F[[i]], m, X], {i, Length[F]}];
gMapMD[T_, m_, X_, Y_] :=
  fMatMD[Expand[mExpsMD[m, Y] /. Thread[Y → T]], m, X]
```

So we take the Sylvester matrix of our domain system, dualize, map the duals with a linear version **gmapMD** of our transformation, return with the **localDualMatrix** getting a large system which we reduce using **hBasisMD**. Because this may be time consuming we do add some options to help the user keep track of what is going on. There are also some warning messages included all making the code somewhat longer than usual in this book.

In[54]:=
```
Options[FLTMD] = {timing → False, hilbertReport → False, hBasis → True};
FLTMD[F_, A_, m_, X_, Y_, tol_, OptionsPattern[]] :=
    Module[{H, XH, YH, T, S, DS, G, TDS, ST, B0, B1, B, n, s, time},
      time = TimeUsed[];
      n = Length[X];
      s = Length[Y];
      If[Dimensions[A] ≠ {s + 1, n + 1}, Print[Style["Dimension Error A", Orange]];
        Abort[]];
      XH = Append[X, #x];
      YH = Append[Y, #y];
      H = Table[homogMD[f, X, #x], {f, F}];
      T = A.XH;
      G = gMapMD[T, m, XH, YH];
      S = sylvesterMD[H, m, XH];
      If[OptionValue[timing], Echo[TimeUsed[] - time, "Start Dual"]];
      DS = dualMatrix[N[S], tol];
      TDS = G.DS;
      If[OptionValue[timing],
        Echo[{Dimensions[TDS], MatrixRank[TDS]}, "Dim TDS,rank TDS"]];
      ST = localDualMatrix[TDS, tol];
      If[Length[ST] == 0, Print[Style["Fail, try larger m", Orange]]; Abort[]];
      B0 = ST.mExpsMD[m, YH];
      If[! OptionValue[hBasis], Return[B0 /. {#y → 1}]];
      If[OptionValue[timing], Echo[TimeUsed[] - time, "Start HBasis"]];
      B1 = hBasisMD[B0, m, YH, tol];
      B = B1 /. {#y → 1};
      If[OptionValue[timing], Echo[TimeUsed[] - time, "Total Time"]];
      B];
```

F is the equation system in the domain, A is the transformation matrix, X,Y are the variables for the domain, range respectively. m will be the order of

the Sylvester matrix used so it must be at least the largest total degree of a polynomial in F but it often needs to be larger. Especially when dealing with numerical data the tolerance may need to be loosened. Since most interesting FLT are numerical this is one good reason why I have been working numerically. It does help if F is an H-basis.

Because of the choices this some what of a trial and error type of algorithm, it probably works in good cases but is not guaranteed. It is therefore a good idea to check the results. The important property that the output G must satisfy is

If F /. Thread [X → p] = 0 then G /. Thread[Y → fltMD[p, A]] = 0.

where, of course, "=0" is interpreted in the numerical sense.

Example 2.6.1 continued.

Consider the cyclic 4 curve of Example 2.2.3 and A above in 2.6.1.

In[162]:=
```
C4 = {w + x + y + z, w x + x y + y z + z w, w x y + x y z + y z w + z w x, w x y z - 1};
g = FLTMD[C4, A, 6, {w, x, y, z}, {x, y}, 1.*^-9][[1]]
```

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

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

Out[163]=
$$1. - 2.21919\,x - 4.23331\,x^2 - 2.28808\,x^3 - 0.674452\,x^4 - 0.143014\,x^5 - 0.00887454\,x^6 - 5.05948\,y + 10.7164\,x\,y + 14.1276\,x^2\,y + 5.2559\,x^3\,y + 1.06694\,x^4\,y + 0.102317\,x^5\,y + 10.9212\,y^2 - 18.5895\,x\,y^2 - 16.1773\,x^2\,y^2 - 3.67542\,x^3\,y^2 - 0.372327\,x^4\,y^2 - 13.2113\,y^3 + 14.2582\,x\,y^3 + 7.34919\,x^2\,y^3 + 0.747335\,x^3\,y^3 + 9.46424\,y^4 - 4.70723\,x\,y^4 - 1.09081\,x^2\,y^4 - 3.67116\,y^5 + 0.544018\,x\,y^5 + 0.556459\,y^6$$

Consider point p of the cyclic 4:

In[164]:=
```
p = {2, -1/2, -2, 1/2};
C4 /. Thread[{w, x, y, z} → p]
g /. Thread[{x, y} → fltMD[p, A]]
```

Out[165]= {0, 0, 0, 0}

Out[166]= 5.50501×10^{-9}

Since our tolerance was 10^{-9} this is good enough for zero. One might want to try a few more points.

We could give more examples now, but we will have many examples in the rest of this book so we will stop here.

2.7 Geometry and Projections

In this section we discuss the the geometry of FLT and the main application, projections.

2.7.1 Some Geometry

As mentioned above a transformation matrix for a transformation $\mathbb{R}^n \longrightarrow \mathbb{R}^k$ is a $(k+1)\times(n+1)$ matrix A.

I will mention here that since transformation functions are essentially projective transformations that that the matrix is homogeneous in that if one multiplies all entries by the same non-zero real (or complex) number the transformation remains the same.

If A is square, that is $k = n$, and A^{-1}exists then the transformation is *invertible*. Geometrically this means that if **FLTMD** takes curve F to curve G then these curves are *isomorphic*, that is geometrically the same. G may be rotated, reflected, translated or the infinite hyperplane may have been moved or possibly all of the above. Some positional attributes may have changed such as critical points, infinite points or number of affine topologi - cal components. But geometrical attributes such as number of ovals or pseudo-lines, algebraic irreducibility and number and characteristics of singular points remain unchanged. For invertible transformation functions one may use **FLT3D** instead of **FLTMD** even if n is not 3. This will be much quicker and the number of equations will not change.

If the last r0w is {0, 0, , …, 0, 1}, or by homogeneity the last entry is some other non-zero number, then we call this transformation function and it's matrix *affine*. This means the infinite hyperplane remains in place and we are just messing with the affine geometry. While critical points may change have the same infinite points and same number of topological components. This latter fact is the original meaning of the word *affine*. The formula **flt[X,A]** will be a list of polynomials rather than rational functions. For example

```
In[121]:= A = {{1, 2, 3, 4}, {5, 6, 7, 8}, {0, 0, 0, 1}};
          fltMD[{x, y, z}, A]
Out[122]= {4 + x + 2 y + 3 z, 8 + 5 x + 6 y + 7 z}
```

If, for an affine transformation A, the last column is {{0}, {0}, … {0}, {1}} then the transformation function is a *linear transformation.* In this case we may strip A by removing the last row and column to get a $k \times n$ matrix, that is $\tilde{A}$ = Drop[A,-1,-1]

In[125]:=
```
A = {{1, 2, 3, 0}, {5, 6, 7, 0}, {0, 0, 0, 1}};
Ã = Drop[A, −1, −1]
```

Out[126]= {{1, 2, 3}, {5, 6, 7}}

Then we can actually perform the transformation just by matrix multiplication

In[127]:=
```
fltMD[{x, y, z}, A]
Ã.{x, y, z}
```

Out[127]= {x + 2 y + 3 z, 5 x + 6 y + 7 z}

Out[128]= {x + 2 y + 3 z, 5 x + 6 y + 7 z}

The process of stripping is reversible, that is a linear transformation $\mathbb{R}^n \longrightarrow \mathbb{R}^k$ given by an $n \times k$ matrix $\tilde{A}$ will give a transformation matrix in the sense of this section by, for example

In[134]:=
```
B = {{1, 2, 3}, {5, 6, 7}};
B̂ = Append[Join[B, {{0}, {0}}, 2], {0, 0, 0, 1}]
```

Out[135]= {{1, 2, 3, 0}, {5, 6, 7, 0}, {0, 0, 0, 1}}

In[136]:=
```
B.{x, y, z}
fltMD[{x, y, z}, B̂]
```

Out[136]= {x + 2 y + 3 z, 5 x + 6 y + 7 z}

Out[137]= {x + 2 y + 3 z, 5 x + 6 y + 7 z}

Finally, a linear transformation B is an *orthogonal transformation* if the rows, equivalently columns, form an orthonormal set. In the real case only, for a $k \times n$ matrix, $k \leq n$ this means **B.Transpose[B]** is the $k \times k$ identity matrix or if $k \geq n$ then **Transpose[B].B** is the $n \times n$ identity. For complex matrices one uses the **ConjugateTranspose**. Orthogonal transformations preserve *Euclidean geometry*, that is that lengths and angles are preserved which does not necessarily happen with affine transformations in general. More importantly operations with orthogonal transformations are more numerically stable, so since we often work with numerical transformation matrices this is good. On the other hand orthogonal matrices almost always have irrational entries and so numerical methods are preferred with them.

Two utility functions that may be useful are given below, they allow us to go between linear transformations and FLT transformations.

```
m2TM[M_] := With[{dim = Dimensions[M]},
    Join[Append[M, Table[0, {dim⟦2⟧}]], Append[Table[{0}, {dim⟦1⟧}], {1}], 2]]
tM2M[T_] := With[{dim = Dimensions[T]}, Take[T, dim⟦1⟧ - 1, dim⟦2⟧ - 1]]
```

2.7.2 Projections

In general a projection will be a linear transformation from $\mathbb{R}^n \longrightarrow \mathbb{R}^k$, $k < n$, given by a $k \times n$ matrix P. Such a matrix can be embedded into a $(k+1) \times (n+1)$ matrix A by the utility functions above. This is so we can treat the projection, as above, as an FLT and have it transform curves as well as points. It is nice if projections are orthogonal, but we will not assume this.

Later we may start with a FLT projection, that is an FLT with fewer rows than columns. These are more general in that infinite points may become affine. These are not really more general as it can be shown that projecting a curve with an arbitrary FLT projection is the same as transforming the curve with an invertible FLT and then doing a linear projection on the image. It is a bit hard to show this so rather than give a proof we just give an algorithm to accomplish this although in practice we will rarely use this.

In[296]:=
```
factorFLT[A_] := Module[{n, k, m, tab1, tab2, A1, A2, A3, B, B1, B2, B3, P, M},
    {n, k} = Dimensions[A] - {1, 1};
    m = k + 1;
    tab1 = Table[{i, m} → #1[i], {i, n}];
    B1 = ReplacePart[IdentityMatrix[m], tab1];
    A1 = A.B1;
    B1 = ReplacePart[IdentityMatrix[m], (tab1 /. Solve[Take[A1, n, -1] == 0])⟦1⟧];
    A1 = A.B1;
    B2 = ReplacePart[IdentityMatrix[m], {{m, m} → A1⟦n + 1, m⟧^-1}];
    A2 = A1.B2;
    B3 = ReplacePart[IdentityMatrix[m], Table[{m, i} → -A2⟦n + 1, i⟧, {i, k}]];
    A3 = A2.B3;
    B = N[B1.B2.B3];
    {Chop[A3], Chop[Inverse[B]]}];
```

Later, for example at the end of section 3.2, we will give some examples of how to use this.

One type of projection is projecting onto several coordinates. For convenience we have a FLT projection from $\mathbb{R}^n \longrightarrow \mathbb{R}^{n-1}$ which removes the i^{th} component.

```
fCompProj [i_, n_] := Module [{F},
      If[i > n, Abort []];
      F = IdentityMatrix [n + 1];
      Delete [F, {i}]];
```

We will distinguish ordinary projections like this one from *generic* projections. These are essentially random or pseudo-random projections although for some purposes they are expected to be stable on a given curve under small perturbations of the projection. This is not quite guaranteed by randomness.

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*

```
prd3D = {{-0.30519764945947847` , 0.9522890290055899` , 0.`},
      {-0.14191095867181538` , -0.045480825358668514` , 0.9888340479238873` }};
```

The associated fractional linear transformation is

```
fprd3D =
    {{-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 3. It is convenient to have a function to quickly plot the projection of a general curve F in $\mathbb{R}^3$.

In[203]:=
```
showProjection3D [F_, pr_, m_, X_, {u_, v_}, rng_] := Module [{PRT, AXS, marks},
    PRT = FLTMD [F, pr, m, X, {u, v}, dTol];
    Echo [PRT, "projection Function "];
    AXS := ListLinePlot [{{{0, 0}, fltMD [{1, 0, 0}, pr]}, {{0, 0}, fltMD [{0, 1, 0}, pr]},
          {{0, 0}, fltMD [{0, 0, 1}, pr]}}, PlotStyle → Orange, PlotRange → All];
    marks := ListPlot [{{fltMD [{1.2, 0, 0}, pr]}, {fltMD [{0, 1.2, 0}, pr]}, {fltMD [{0, 0, 1.2}, pr]}},
        PlotMarkers → {"x= 1", "y=1", "z=1"}, PlotStyle → Black];
    Show [ContourPlot [PRT == 0, {u, -rng, rng}, {v, -rng, rng}], AXS, marks, Frame → False]]
```

Here F is a general curve, **pr** is an FLT projection, m the order of Sylvester matrices to use, generally larger then the degrees of equations in F, X are the variables in $\mathbb{R}^3$, {u,v} the variables in $\mathbb{R}^2$ and **rng** the size of the image, eg. if **rng** is 2 then the projection is given in the square $\{\{x, y\}, -2 \le x, y \le 2\}$.

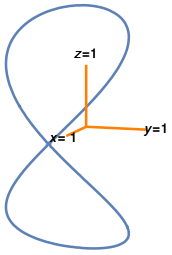
2.7.2.1 Example: The Viviani curve

In[204]:=
```
V = {-4 + x^2 + y^2 + z^2, -1 + (-1 + x)^2 + y^2}
showProjection3D [V, fprd3D , 4, {x, y, z}, {x, y}, 3]
```

Out[204]= $\left\{-4 + x^2 + y^2 + z^2,\ -1 + (-1 + x)^2 + y^2\right\}$

» projection Function $\left\{1. + 4.30229\, x + 3.68817\, x^2 + 0.024428\, x^3 + 0.000444366\, x^4 - 2.00048\, y + 0.312204\, x\, y + 1.0116\, x^2\, y - 3.77986\, y^2 - 1.05115\, x\, y^2 + 0.0400199\, x^2\, y^2 + 0.511479\, y^3 + 0.901056\, y^4\right\}$

Out[205]=



A problem with ordinary projections is that the projection may change the geometry of curves. This may be an accident or, as we will see in Section 3.3, this may happen because of the geometry of the curve.

2.7.2.2 Example: If we take a curve such as $\{x^2 + z^2 - 1,\ y\}$ under the projection **fCompProj[3,3]** we get the curve projection as a line $y = 0$.

In[149]:=
```
FLTMD [{x ^ 2 + z ^ 2 - 1, y}, fCompProj [3, 3], 3, {x, y, z}, {x, y}, dTol]
```

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

» Final Hilbert Function {1, 2, 3, 4}

Out[149]= {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

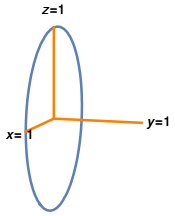
In[152]:=
```
showProjection3D[{x ^ 2 + z ^ 2 - 1, y}, fprd3D, 3, {x, y, z}, {x, y}, 2]
```

» Initial Hilbert Function {1, 3, 5, 7}

» Final Hilbert Function {1, 3, 5, 7}

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

Out[152]=



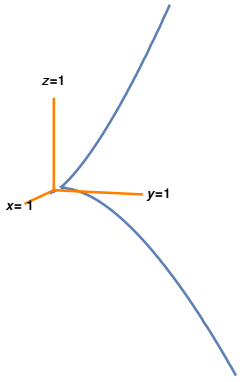
is correctly given as a circle.

2.7.2.3 Example: Even our pseudorandom projection prd3D may not be generic for some curves. For example we consider our twisted cubic:

In[250]:=
```
twCubic = {-y^2 + x z, -x^2 + y, -x y + z};
showProjection3D [twCubic , fprd3D , 3, {x, y, z}, {x, y}, 2]
```

» projection Function $\{-0.464981\, x + 16.8091\, x^2 - 64.3264\, x^3 + 1.\, y - 51.2934\, x\, y + 56.8131\, y^2\}$

Out[251]=



This appears to give a cusp.

In[182]:=
```
P = prd3D + RandomReal [{-.2, .2}, {2, 3}];
FP = m2TM [P]
```

Out[183]= {{-0.134773 , 0.808097 , 0.128253 , 0}, {-0.223291 , 0.0745526 , 0.884676 , 0}, {0, 0, 0, 1}}

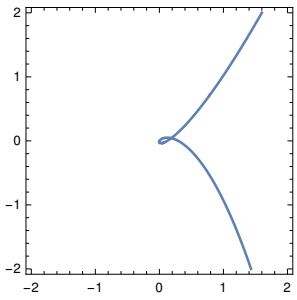
In[252]:=
```
tw2 = FLTMD [twCubic , FP, 3, {x, y, z}, {x, y}, 1.*^-9 ][[1]]
ContourPlot [tw2 == 0, {x, -2, 2}, {y, -2, 2}, MaxRecursion → 4, ImageSize → Small]
```

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

» Final Hilbert Function {1, 3, 6, 9}

Out[252]= $-1.65679\,x + 19.1206\,x^2 - 45.8792\,x^3 + 1.\,y -$
$23.8022\,x\,y + 19.9535\,x^2\,y + 32.7582\,y^2 - 2.89269\,x\,y^2 + 0.139785\,y^3$

Out[253]=



is clearly a node. In his quoted article Barry Mazur [B.Mazur] says that cusps do not occur under generic projections of non-singular curves.

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. In this book we take this not as a theorem but a requirement for a random or pseudo random projection to be *generic* for the curve. Note that for us this is a property of the curve, not the projection, for a randomly generated numerical curve the projections **fCompProj** may be generic but possibly not for an integer coefficient curve.

As mentioned in Chapter 1 a singularity in a projection of a non-singular curve will be called *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, in the case of a generic projection I call these *ghost* components although algebraists may call them *embedded components*. The important result is

Under any projection of a space curve to the plane a non-singular point may go to a singular point. For generic projections the only artifactual singularities will be normal crossings (nodes), cusps or isolated points.

2.8 Fibers and Plotting Space Curves

Our general strategy for plotting space curves is to project onto $\mathbb{R}^2$, path trace and lift the trace to $\mathbb{R}^3$ with the function **fFiberMD** in the next subsection and plot there.

2.8.1 Fiber lifting

A projection is not 1-1, in fact, in this section where we will restrict to linear projections $\mathbb{R}^n \longrightarrow \mathbb{R}^{n-1}$, 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.

Suppose P is the original projection i.e. a $n \times (n-1)$ matrix of rank $n-1$ and 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.

```
pFiberMD [P_, p_, t_] := Module [{n, k, P1, ns, q},
    {n, k} = Dimensions [P];
    If[n ≠ k – 1 || MatrixRank [P] ≠ k – 1 , Echo ["not valid Projection "];
      Abort []];
    P1 = Append [P, RandomReal [{–3, 3}, k]];
    ns = NullSpace [P][[1]];
    q = Inverse [P1].Append [p, RandomReal [{–3, 3}]];
    q + t * ns]
```

For example

In[159]:=
```
p = RandomReal[{–4, 4}, 2]
pFiberMD[prd3D, p, t]
```

Out[159]= {–0.257823, –0.846821}

Out[160]= {7.60814 + 0.941656 t, 2.16758 + 0.30179 t, 0.335184 + 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 $n \times (n-1)$ matrix of rank $n-1$, 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[73]:=
```
Options[fFiberMD] = {complex → False}
fFiberMD[F_, P_, p_, X_, tol_, OptionsPattern[]] :=
  Module[{Pf, FF, FFs, sol, sol0, sol1, k, n, l, q, u, j, t0},
    n = Dimensions[P][[2]];
    k = Length[F];
    Pf = pFiberMD[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]];
    FF = Chop[FF, tol];
    If[OptionValue[complex], sol = NSolve[FF[[1]], sol = NSolve[FF[[1]], t734, Reals]];
    If[Length[sol] == 0, Echo[p, "(1) no point in fiber at"]; Return[{}]];
    sol0 = t734 /. sol;
    j = 2;
    While[j ≤ k && Length[sol0] > 0 && (FF[[j]] /. {t734 → t0}) ≠ 0,
      If[OptionValue[complex], sol = NSolve[FF[[j]], sol = NSolve[FF[[j]], t734, Reals]];
      If[Length[sol] == 0, Echo[p, "(2) 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, "(3) 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 no point in fiber warning occurs then you can try loosening the tolerance. If this happens in list form you may need to delete empty sets {} In the output. The numbers in parenthesis in this warning may help in trouble shooting.

2.8.1.1 Example:

In[142]:=
```
F = {-9 x - 45 y - 9 x z + 9 y z, 18 x - 0.25 x^2 + 36 y + 0.5 x y - 0.25 y^2 - 9 x z + 9 x z^2,
     -54 + 1.5 x - 1.5 y + 99 z - 54 z^2 + 9 z^3};
```

We first try the projection onto the xy plane.

In[124]:= **Pxy = {{1, 0, 0}, {0, 1, 0}};**

We look at some examples of **fFiberMD.**

In[144]:= **fFiberMD [F, Pxy, RandomReal [{−5, 5}, 2], {x, y, z}, 1.*^−9]**

» (3) no point in fiber at {−3.62597 , 4.11431 }

Out[144]= {}

In[145]:= **fFiberMD [F, Pxy, {−6, 30}, {x, y, z}, 1.*^−9]**

Out[145]= {{−6., 30., 4.}}

In[147]:= **fFiberMD [F, Pxy, {0, 0}, {x, y, z}, 1.*^−9]**

» multiple fiber points {0, 0}

Out[147]= {{0., 0., 1.}, {0., 0., 2.}, {0., 0., 3.}}

In the first case the fiber is empty which happens for most points. In the second case the fiber consists of one point which is typical of points in the projection of the curve. In the last case there are 3 points in the fiber.

The projection of the curve on the xy plane is

In[150]:= **f = FLTMD [F, fCompProj [3, 3], 5, {x, y, z}, {x, y}, 1.*^−9][[1]]**

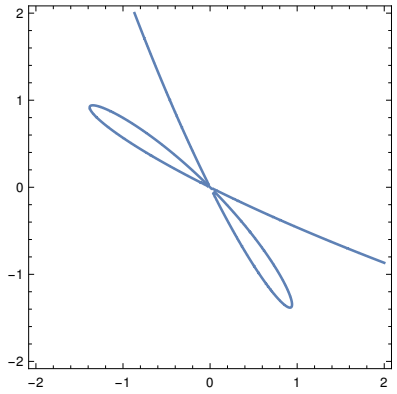
» Initial Hilbert Function {1, 3, 6, 10, 15, 20}

» Final Hilbert Function {1, 3, 6, 10, 15, 20}

Out[150]= $1.\,x^3 - 0.00694444\,x^4 + 3.5\,x^2\,y + 0.0277778\,x^3\,y + 3.5\,x\,y^2 - 0.0416667\,x^2\,y^2 + 1.\,y^3 + 0.0277778\,x\,y^3 - 0.00694444\,y^4$

In[153]:= **ContourPlot [f == 0, {x, −2, 2}, {y, −2, 2}, MaxRecursion → 6]**

Out[153]=



This shows the point {0, 0,} with 3 points in its fiber is a singular point of multiplicity 3 as verified by

In[154]:= **tangentVectorMD [{f}, {0, 0}, {x, y}]**

» Hilbert Function {1, 2, 3, 3, 3}

» No unique tangent vector at {0, 0}

Our general plotting strategy calls for us to trace the plane curve f. Unfortunately it has a singularity which will require us to break this into at least 6 paths always tracing into the singularity at {0,0}. We will show one.

In[158]:=
```
ps = {x, y} /. NSolve[{f, x^2 + y^2 - 3}, {x, y}, Reals]
```

Out[158]=
```
{{1.58268, -0.70365}, {-0.70365, 1.58268}}
```

In[159]:=
```
p1 = ps[[1]];
pth1 = pathFinder2D[f, p1, {0, 0}, .1, x, y]
```

Out[160]=
```
{{1.58268, -0.70365}, {1.48995, -0.66622}, {1.39735, -0.628458}, {1.3049, -0.590356},
 {1.21259, -0.551904}, {1.12043, -0.513092}, {1.02842, -0.47391}, {0.936584, -0.434346},
 {0.844914, -0.394389}, {0.753423, -0.354025}, {0.662119, -0.313241},
 {0.571011, -0.272021}, {0.480108, -0.230349}, {0.389423, -0.188206},
 {0.298967, -0.145575}, {0.208753, -0.102433}, {0.118797, -0.0587558}, {0, 0}}
```

We can now lift this to $\mathbb{R}^3$ with the following

In[162]:=
```
Pth = Flatten[fFiberMD[F, Pxy, #, {x, y, z}, 1.*^-9] & /@ pth1, 1]
```

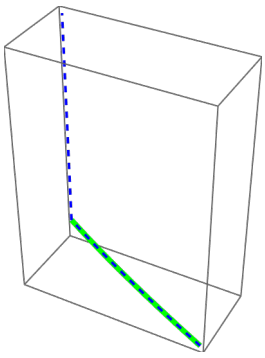
» multiple fiber points {0, 0}

Out[162]=
```
{{1.58268, -0.70365, 0.846583}, {1.48995, -0.66622, 0.853897}, {1.39735, -0.628458, 0.861351},
 {1.3049, -0.590356, 0.868949}, {1.21259, -0.551904, 0.8767}, {1.12043, -0.513092, 0.884613},
 {1.02842, -0.47391, 0.892695}, {0.936584, -0.434346, 0.900956}, {0.844914, -0.394389, 0.909407},
 {0.753423, -0.354025, 0.918059}, {0.662119, -0.313241, 0.926925},
 {0.571011, -0.272021, 0.936019}, {0.480108, -0.230349, 0.945356},
 {0.389423, -0.188206, 0.954954}, {0.298967, -0.145575, 0.964832},
 {0.208753, -0.102433, 0.975012}, {0.118797, -0.0587558, 0.98552}, {0., 0., 1.}, {0., 0., 2.}, {0., 0., 3.}}
```

This is good except for the last 3 points which are all liftings of {0, 0}. We have to pick just one of these, the one that most closely matches the previous point. We see that is {0,0,1}. The plot is the not exciting green curve which is what we want. Had we not dropped the other points we would have the blue dashed curve that goes though all three fiber lifts.

In[174]:=
```
Pth1 = Drop[Pth, -2];
Graphics3D [{{Green, Thick, Line[Pth1]}, {Blue, Dashed, Line[Pth]}}, ImageSize → Small]
```

Out[175]=



2.8.1.2 Example 2.8.1.1 Continued

Rather than continue on the other 5 tracings we project again using our pseudo-random projection **prd3D** which has no singular points.

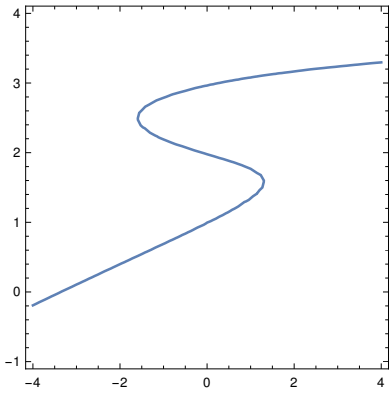
In[178]:=
```
g = FLTMD [F, fprd3D, 5, {x, y, z}, {x, y}, 1.*^-9, quiet → True][[1]]
```

Out[178]=

$$1. + 0.271206\ x - 0.00614083\ x^2 + 0.000532178\ x^3 - 4.49813 \times 10^{-6}\ x^4 - 1.85033\ y - 0.216147\ x\,y + 0.000588023\ x^2\,y - 0.0000621309\ x^3\,y + 1.01583\ y^2 + 0.0494672\ x\,y^2 - 0.000321821\ x^2\,y^2 - 0.168582\ y^3 - 0.000740863\ x\,y^3 - 0.000639577\ y^4$$

In[180]:=
```
ContourPlot [g == 0, {x, -4, 4}, {y, -1, 4}]
```

Out[180]=



A single trace and lift suffices

In[183]:=
```
sol = {x, y} /. NSolve [{g, x ^ 2 + y ^ 2 - 13}, {x, y}, Reals]
```

Out[183]=
```
{{1.75625, 3.1489}, {-3.60479, -0.0740102}}
```

In[185]:=
```
pth2 = pathFinder2D [g, sol⟦2⟧, sol⟦1⟧, .3, x, y, maxit → 60]
```

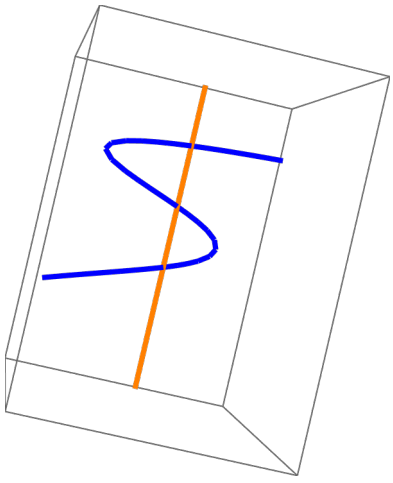
Out[185]=
```
{{−3.60479, −0.0740102}, {−3.31723, 0.0114823}, {−3.02954, 0.096539}, {−2.74174, 0.181205},
 {−2.45383, 0.265537}, {−2.16585, 0.349601}, {−1.87782, 0.433483}, {−1.58976, 0.517291},
 {−1.30173, 0.601166}, {−1.01376, 0.68529}, {−0.725946, 0.769914}, {−0.438382, 0.855388},
 {−0.151228, 0.942226}, {0.135258, 1.03122}, {0.420622, 1.1237}, {0.703909, 1.22217}, {0.982472, 1.33246},
 {1.24036, 1.47417}, {1.30691, 1.60214}, {1.14235, 1.72082}, {0.857493, 1.80651}, {0.564651, 1.87072},
 {0.270109, 1.92755}, {−0.0248564, 1.98228}, {−0.31963, 2.038}, {−0.613649, 2.09744}, {−0.906041, 2.16407},
 {−1.19469, 2.24418}, {−1.46944, 2.35513}, {−1.59794, 2.49846}, {−1.49041, 2.62449}, {−1.2187, 2.73378},
 {−0.928884, 2.80859}, {−0.634948, 2.86767}, {−0.339176, 2.91739}, {−0.0423628, 2.96074}, {0.25512, 2.99935},
 {0.553067, 3.03426}, {0.851357, 3.06616}, {1.14991, 3.09555}, {1.44866, 3.1228}, {1.75625, 3.1489}}
```

In[187]:=
```
Pth2 = Flatten [fFiberMD [F, prd3D , #, {x, y, z}, 1.*^−9 ] &/@ pth2 , 1]
```

Out[187]=
```
{{5.48543, −2.02738, 0.619139}, {4.99583, −1.88232, 0.642004}, {4.51393, −1.73466, 0.665653},
 {4.03997, −1.58434, 0.69017}, {3.57421, −1.43128, 0.715652}, {3.11697, −1.27541, 0.742214},
 {2.6686, −1.11665, 0.769998}, {2.22951, −0.95488, 0.799178}, {1.8002, −0.79, 0.829972},
 {1.38129, −0.621866, 0.862659}, {0.973522, −0.450314, 0.897609}, {0.577872, −0.275144, 0.935325},
 {0.195637, −0.0961059, 0.976521}, {−0.171355, 0.0871173, 1.02228}, {−0.520314, 0.274941, 1.07436},
 {−0.846425, 0.467907, 1.13602}, {−1.1393, 0.666562, 1.21466}, {−1.35985, 0.866691, 1.33552},
 {−1.34487, 0.941369, 1.47053}, {−1.09363, 0.849083, 1.62236}, {−0.773361, 0.652601, 1.74594},
 {−0.485792, 0.43725, 1.84223}, {−0.222806, 0.212235, 1.92711}, {0.0197011, −0.0197877, 2.00658},
 {0.243541, −0.257592, 2.08412}, {0.449036, −0.500483, 2.16255}, {0.634915, −0.747952, 2.24522},
 {0.796855, −0.999161, 2.33792}, {0.918955, −1.24855, 2.45619}, {0.92399, −1.38187, 2.59572},
 {0.806603, −1.30657, 2.70979}, {0.623279, −1.08, 2.80443}, {0.456993, −0.828961, 2.86776},
 {0.302882, −0.569689, 2.91731}, {0.157592, −0.305663, 2.9589}, {0.0192306, −0.0383221, 2.99517},
 {−0.113399, 0.231558, 3.02759}, {−0.241125, 0.503499, 3.05707}, {−0.364552, 0.777176, 3.08421},
 {−0.484145, 1.05236, 3.10943}, {−0.60027, 1.32886, 3.13304}, {−0.716456, 1.61462, 3.1559}}
```

In[190]:=
```
Graphics3D[{{Blue, Thick, Line[Pth2]}, {Orange, Thick, Line[{{0, 0, −1}, {0, 0, 4}}]}}]
```

Out[190]=



The orange line is the z-axis which intersects the curve in 3 places. Again, don't expect to find a generic projection with no singularities, that will usually not happen as remarked above. But at least generic projections do eliminate singularities of multiplicity greater than 2.

2.8.2 Example: Application to Cyclic 4

Recall the cyclic-4 curve, Example 2.2.3, is given by

```
In[122]:= C4 = {w + x + y + z, w x + x y + y z + z w, w x y + x y z + y z w + z w x, w x y z - 1};
```

Here we sketch an analysis of the cyclic-4 curve using our method. For curves in $\mathbb{R}^n$ for $n > 3$ we project first to $\mathbb{R}^3$, hopefully this will not introduce new singularities, then to $\mathbb{R}^2$ preferably with a random or pseudo-random projection. We then lift back to $\mathbb{R}^3$ for plotting.

For definiteness here is our random affine projection $\mathbb{R}^4 \longrightarrow \mathbb{R}^3$

```
In[129]:= 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` }};
          FP43 =
            m2TM[
              P43];
```

We could project directly to $\mathbb{R}^2$ but will need to know the image of **C4** in $\mathbb{R}^3$, it takes some time but the answer is

```
In[171]:= C43 = FLTMD[C4, FP43, 6, {w, x, y, z}, {x, y, z}, 1.*^-9]
```

» Initial Hilbert Function {1, 4, 9, 15, 21, 26, 30}

» Final Hilbert Function {1, 4, 9, 15, 21, 26, 30}

$$\begin{aligned}\text{Out[171]}= \{&0.515334\, x^2 + 0.0261946\, x y + 0.000332871\, y^2 + 1.43574\, x z + 0.0364895\, y z + 1.\, z^2,\\ &-0.63185\, x^3 + 0.561652\, x^2 y + 0.73255\, x y^2 + 0.0182448\, y^3 - 0.880176\, x^2 z + 0.80476\, x y z + 1.\, y^2 z,\\ &1. + 0.58731\, x^4 - 1.11729\, x^3 y - 0.522163\, x^2 y^2 +\\ &\quad 0.249241\, x y^3 - 0.0282582\, y^4 + 0.849617\, x^3 z - 1.26749\, x^2 y z\}\end{aligned}$$

Now we project to $\mathbb{R}^{2.}$

```
In[208]:= C42 = FLTMD[C43, fprd3D, 6, {x, y, z}, {x, y}, 1.*^-9][[1]]
```

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

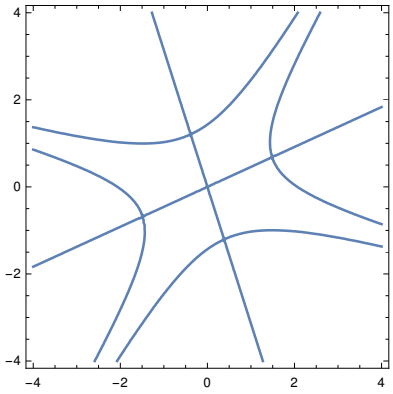
» Final Hilbert Function {1, 3, 6, 10, 15, 21, 27}

$$\begin{aligned}\text{Out[208]}= &-1.43989\, x^2 + 0.0789016\, x^6 + 2.68184\, x y + 0.323495\, x^5 y + 1.\, y^2 -\\ &0.558284\, x^4 y^2 - 2.00039\, x^3 y^3 + 1.90728\, x^2 y^4 + 0.0432741\, x y^5 - 0.237496\, y^6\end{aligned}$$

We plot **C42**

In[179]:=
```
ContourPlot [C42 == 0, {x, −4, 4}, {y, −4, 4}]
```

Out[179]=



In[180]:=
```
cp2 = criticalPoints2D [C42, x, y]
```

Out[180]=
```
{{−1.48804 , −0.682283 }, {−1.48804 , −0.682283 }, {1.48804 , 0.682283 },
 {1.48804 , 0.682283 }, {1.51333 , 0.612185 }, {−1.51333 , −0.612185 }, {0.384516 , −1.20751 },
 {0.384516 , −1.20751 }, {−0.384516 , 1.20751 }, {−0.384516 , 1.20751 }, {−0.473031 , 1.16934 },
 {0.473031 , −1.16934 }, {−1.41781 × 10^−38, −2.33748 × 10^−38}, {0., 0.}, {0., 0.}, {0., 0.}}
```

It appears that we have two lines through {0,0} which will be components of the point curve V(h). From the critical points the other singularities are clear so the two lines are

In[203]:=
```
l1 = line2D [{0, 0}, cp2[[2]], x, y]
l2 = line2D [{0, 0}, cp2[[8]], x, y]
```

Out[203]=
```
0. − 1.5907 x + 3.46926 y
```

Out[204]=
```
0. − 1.58524 x − 0.504796 y
```

Note by symmetry we expect these to be perpendicular, but they are not. By nDivideMD we get

In[207]:=
```
c42 = nDivideMD [C42, l1 * l2, {x, y}, 1.*^−9 ]
```

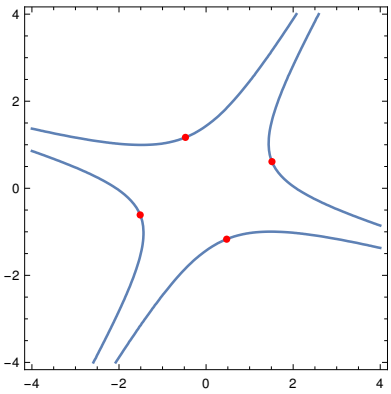
Out[207]=
```
−0.571014 + 0.0312899 x^4 + 0.186567 x^3 y + 0.14782 x^2 y^2 − 0.388404 x y^3 + 0.135614 y^4
```

In[125]:=
```
cp42 = criticalPoints2D [c42, x, y]
```

Out[125]=
```
{{−7162.89 , 2009.12 }, {7162.89 , −2009.12 }, {1.51333 , 0.612185 },
 {−1.51333 , −0.612185 }, {−0.473031 , 1.16934 }, {0.473031 , −1.16934 }}
```

In[126]:=
```
ContourPlot [c42 == 0, {x, −4, 4}, {y, −4, 4}, Epilog → {Red, PointSize [Medium ], Point [cp42]}, ]
```

Out[126]=



In[127]:=
```
infc42 = infiniteRealPoints2D [c42, x, y]
```

Out[127]=
```
{{−54.2416 , −92.8895 , 0}, {−1.59294 , 0.446802 , 0}}
```

From the fact that there are two infinite points and apparently 4 arcs connecting with each that these both have multiplicity 2. One could check these by inspecting the infinite points as in my plane curve book. But this total multiplicity is too large for a reducible curve so it must be irreducible. One could use **singularFactor** from the plane curve book but we can use instead **dualInterpolationMD** discussed earlier in this book.

In[140]:=
```
c42a = dualInterpolationMD [{c42}, cp42[[{3, 4}]], 2, {x, y}, 1.*^−9 ][[1]]
c42b = dualInterpolationMD [{c42}, cp42[[{5, 6}]], 2, {x, y}, 1.*^−9 ][[1]]
```

» Initial Hilbert Function {1, 2, 2}

» Final Hilbert Function {1, 2, 2}

Out[140]=
$$2.05197 - 0.480342\ x^2 - 1.43202\ x\,y + 1.\ y^2$$

» Initial Hilbert Function {1, 2, 2}

» Final Hilbert Function {1, 2, 2}

Out[141]=
$$-2.05197 - 0.480342\ x^2 - 1.43202\ x\,y + 1.\ y^2$$

Note that these are both factors of c42 but don't multiply to c42 because the curves are defined only up to a constant, but checking

In[161]:=
```
Expand [c42a * c42b / c42a[[1]] / c42b[[1]] − c42 / c42[[1]]]
```

Out[161]=
$$0. + 5.07927 \times 10^{-15}\ x^2 - 6.8695 \times 10^{-16}\ x^4 + 3.88578 \times 10^{-15}\ x\,y - 6.32827 \times 10^{-15}\ x^3\,y + 1.75415 \times 10^{-14}\ y^2 - 6.27276 \times 10^{-15}\ x^2\,y^2 + 5.9952 \times 10^{-15}\ x\,y^3 - 9.40914 \times 10^{-15}\ y^4$$

In particular, c42 is reducible as the union of two quadratics and two linear curves.

Normally we would now lift to $\mathbb{R}^3$ to get a plot of the curve, or at least its projection in $\mathbb{R}^3$. We can work with each component separately. For the lines it is possible that they lift to higher degree curves, but not likely given our pseudo-random projection. So we could start with two points on, say **l1** but will stay away from the origin and infinite point. By very elementary algebra setting $x = 3$ we get

In[167]:= **b = –Expand [(l1 /. {x → 3}) / Coefficient [l1, y]]**

Out[167]= 1.37553 – 1. y

In[169]:= **p1 = {3, b /. {y → 0}}**

Out[169]= {3, 1.37553 }

p1 is on our line. Lifting to **C43** with a very loose tolerance

In[176]:= **fFiberMD [C43, prd3D, p1, {x, y, z}, 1.*^–4]**

» (3) no point in fiber at {3, 1.37553 }

Out[176]= {}

we find there is no point! Perhaps checking this very carefully and trying other points on the lines **l1,l2** we suspect that *these are ghost lines.* Finding that they still occur in the plane but not in space also for other projections confirms this.

We can still try to lift the quadratic curves to $\mathbb{R}^3$. We pick 8 points on the union c42 of the quadratics

In[190]:= **sol = {x, y} /. NSolve [{c42, x ^ 2 + y ^ 2 – 9}]**

Out[190]= {{–1.81015 , –2.39235 }, {1.81015 , 2.39235 }, {–1.21355 , –2.74359 }, {1.21355 , 2.74359 },
{2.96438 , –0.460916 }, {–2.96438 , 0.460916 }, {2.77979 , –1.12817 }, {–2.77979 , 1.12817 }}

In[195]:= **sol3 = Flatten [fFiberMD [C43, prd3D, ⌗, {x, y, z}, 1.*^–7] & /@ sol, 1]**

Out[195]= {{2.88159 , –0.977326 , –2.05077 }, {–2.88159 , 0.977326 , 2.05077 },
{3.2387 , –0.236387 , –2.32065 }, {–3.2387 , 0.236387 , 2.32065 }, {0.301768 , 3.20961 , –0.275188 },
{–0.301768 , –3.20961 , 0.275188 }, {1.08098 , 3.2655 , –0.835578 }, {–1.08098 , –3.2655 , 0.835578 }}

and, surprise

In[197]:= **planar3D [cp43]**

» Residue = 2.48742×10^{-8}

Out[197]= $-1.05818 \times 10^{-7} - 3.56651\ x - 0.0906435\ y - 4.9682\ z$

these points all lie on a plane!

We can further lift these on the affine projection P43 from $\mathbb{R}^4$ to $\mathbb{R}^3$.

In[201]:= **sol4 = Flatten [fFiberMD [C4, P43, ⌗, {w, x, y, z}, 1.*^–7] & /@ sol3, 1]**

Out[201]= {{2.63725 , –0.379183 , –2.63725 , 0.379183 }, {–2.63725 , 0.379183 , 2.63725 , –0.379183 },
{2.80448 , 0.356572 , –2.80448 , –0.356572 }, {–2.80448 , –0.356572 , 2.80448 , 0.356572 },
{–0.336978 , 2.96755 , 0.336978 , –2.96755 }, {0.336978 , –2.96755 , –0.336978 , 2.96755 },
{0.316884 , 3.15573 , –0.316884 , –3.15573 }, {–0.316884 , –3.15573 , 0.316884 , 3.15573 }}

In[204]:= **linearSetMD [Take [sol4, 5], {w, x, y, z}]**

Out[204]= $\{-8.88178 \times 10^{-16} - 0.5\ w - 0.5\ x - 0.5\ y - 0.5\ z\}$

These all lie in the vector subspace $w + x + y + z = 0$ which is not a surprise since that is one of the equations in the system **C4**.

But since these are numerical we can calculate the rank

In[205]:=
```
SingularValueList [sol4]
```

Out[205]= $\{8.72602, 7.75478, 1.79033 \times 10^{-8}\}$

which is numerically 2. So they actually lie in a plane in $\mathbb{R}^4$. We numerically calculate the null space of this set as

In[209]:=
```
{U, S, V} = SingularValueDecomposition [sol4];
Take[V, All, -2] // MatrixForm
```

Out[210]//MatrixForm=
$$\begin{pmatrix} -0.5 & 0.5 \\ 0.5 & 0.5 \\ -0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

which says that set lies also in the hyperplane $-w + x - y + z = 0$

In[211]:=
```
(-w + x - y + z) /. Thread [{w, x, y, z} → #] & /@ sol4
```

Out[211]= $\{0., -1.68754 \times 10^{-14}, 4.17632 \times 10^{-8}, 1.77636 \times 10^{-15}, 1.06581 \times 10^{-14}, -2.66454 \times 10^{-15}, 3.9968 \times 10^{-15}, -2.66454 \times 10^{-15}\}$

*So the curve **C4** lies in a 2-plane of* $\mathbb{R}^4$ *and the plot is the same as **c42.*** Actually this is quite well known, see for example the reference [Androvic, Verschelde].

The two linear equations, $w + x + y + z = 0$, $-w + x - y + z = 0$ are equivalent to the two equations $w = -y$, $x = -z$. If we just look at the output of **sol4** above we see that this is the case at least for the display digits. It is easy to see from the membership problem that the second equation is not a member of the **C4** system. This is also quite obvious when we add the second equation to **C4** and find the H-basis

In[135]:=
```
C4e = Append [C4, -w + x - y + z];
HC4e = hBasisMD [C4e, 4, {w, x, y, z}, dTol]
```

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

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

Out[136]= $\{1. w + 1. y, 1. x + 1. z, -1. + 1. w^2 x^2\}$

The first two equations here are the ones we deduced above while the last says $w = \pm\frac{1}{x}$. Again this is easy to check in **sol4.**

Further this is the equation of the union of two disjoint hyperbolas in the $\{w - y, x - z\}$ plane of $\mathbb{R}^4$, the fact we worked hard to get. This is also a well known fact but other derivations are at least as hard as ours above.

We recall that in Section 2.2 that we noticed the strange fact

In[137]:=
```
tangentVectorMD[C4, {1, -1, -1, 1}, {w, x, y, z}]
```

» Hilbert Function {1, 2, 1, 1, 1}

» No unique tangent vector at {1, −1, −1, 1}

that this point was singular of multiplicity 1. But with our extended **C4e** we have

In[138]:= **tangentVectorMD [C4e, {1, −1, −1, 1}, {w, x, y, z}]**

» Hilbert Function {1, 1, 1, 1, 1}

Out[138]= {0.5, 0.5, −0.5, −0.5}

so this point is regular.

The takeaway from this discussion is that what makes the Cyclic 4 system strange is that it is missing, by membership, an equation satisfied by the 1-dimensional solution. This also explains the ghost lines in the projection of the **C4** on the plane, these are gone when projecting **C4e**.

2.8.3 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.

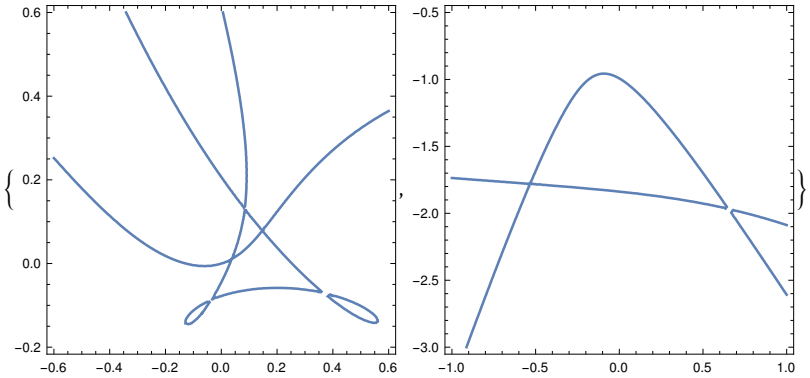
This curve was originally randomly generated as the intersection of 3 quadratic hypersurfaces of 4 space.

In[238]:=
```
f1 = 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;
f2 = −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;
f3 = 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;
F4 = {f31, f32, f33};
```

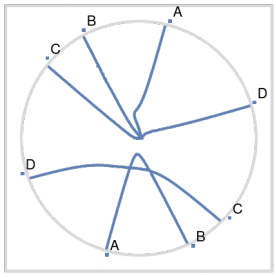
We first project with Pxyz, the simple projection setting $w = 0$. This gives a system of 7 equations of degree 5 in the three variables x, y, z. We will not reproduce this. We next project by our default pseudo-random projection PRD. We get a numerical plane curve of degree 8 with coefficients ranging 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}]}
```

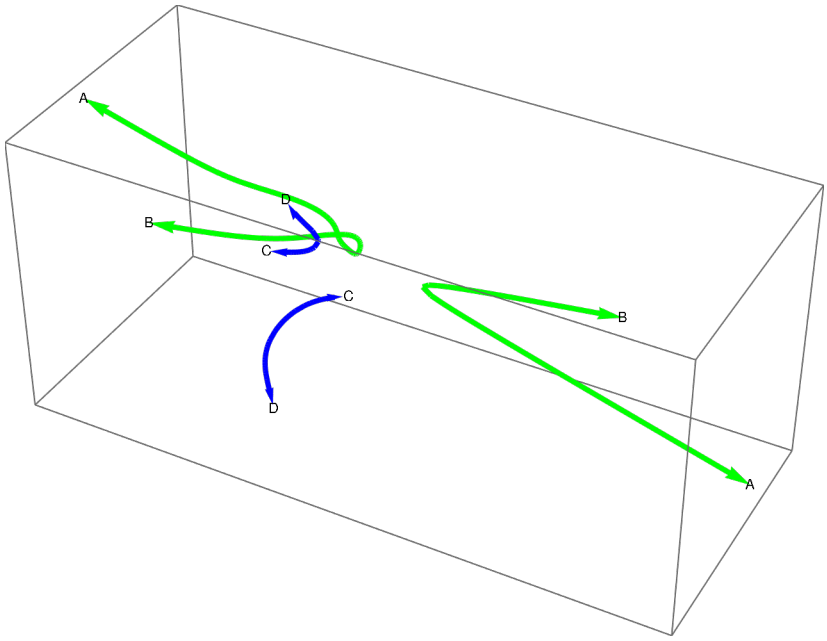
Out[154]=



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 p**athFinder2D** using the **closestPoint2D** algorithm. But with a curve of degree 8 it is way too slow. Our 2D differential equation path finder interpolates the curve quickly with a piecewise linear curve but the points given are too approximate for fF**iber**. So we use **pathFinderT2D** 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.

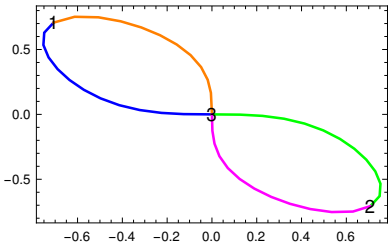
2.9 Fundamental Theorem

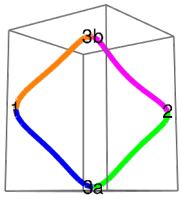
In my plane curve book I introduce the Fundamental Theorem. This does carry over to space curves in the general case. Again projection to the plane and fiber lifting can be used to find a graph of the space curve. Singular points in the plane projection may lift to several points so the corresponding 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.

2.9.1.1 Example

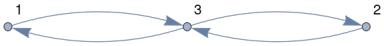
In[232]:=
```
F = {x + y - x z + y z, -x - 2 x^2 y - 2 x y^2 - 2 y^3 + x z + 2 x^3 z,
     -1 + 6 x^2 + 8 x y + 4 y^2 - 4 x^2 z + z^2 + 2 x^2 z^2, x^4 + x y + y^4};
```

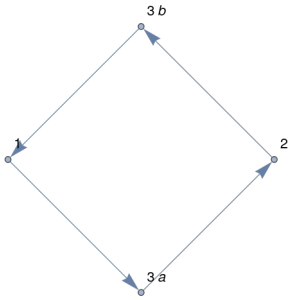
Projecting with the non - generic projection $z \to 0$ gives the last equation $x^4 + x\,y + y^4 = 0$. Plotting this with path tracing and lifting gives





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 typically 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 Funda - mental 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.

2.9.1.2 Example 2.8.2 continued.

The infinite points of F4 are given by

```
In[243]:= infF4 = infiniteRealPointsMD[F4, {x, y, z, w}, 1.*^-10]
Out[243]= {{-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}}
```

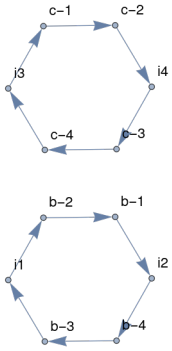
labeled by i1,…i4 which project to infinite plane points

{{−1.25675 , 1.55582 }, {1.07143 , 1.6888 },
{−1.63582 , 1.1507 }, {−1.68853 , −1.07186 }}

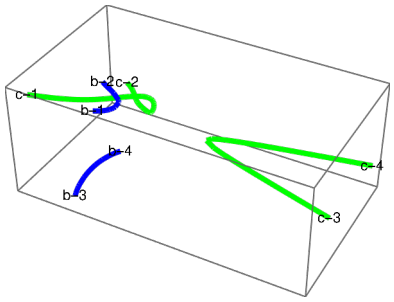
Using the Fundamental Theorem in the plane we can infer the following graph

In[244]:=
```
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[244]=



which compares to the plot above with endpoints labeled.



2.9.2 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 deter-

mine 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 appear in Section 3.2 below

2.9.2.1 Example

In[245]:=
```
case8 = {x y + z, 1 + 2 x y + y^2 - z^2};
```

Checking infinite points

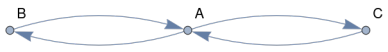
In[246]:=
```
IP = infiniteRealPoints3D [case8, {x, y, z}]
```

Out[246]=
```
{{0., -0.707107, 0.707107, 0}, {1., 0., 0., 0}, {1., 0., 0., 0}, {0., 0.707107, 0.707107, 0}}
```

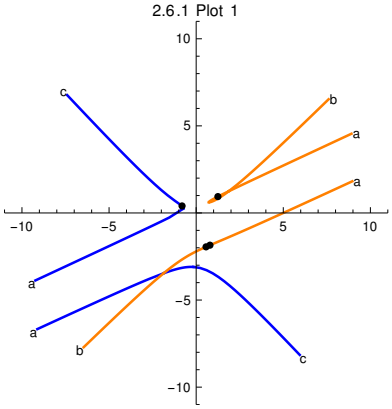
The second infinite point is singular which is why it repeats. We will label these distinct points C, A, B respectively. It can be shown that a graph for this 3 dimensional curve is

In[247]:=
```
Graph[{"A" → "C", "C" → "A", "A" → "B", "B" → "A"}, VertexLabels → "Name"]
```

Out[247]=



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

This plane intersects the space curve in

```
In[249]:= sol1 = {x, y, z} /. NSolve[Append[case8, plane]]
Out[249]= {{−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[250]:= fltMD[#, fprd3D] &/@ sol1
Out[250]= {{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 projection 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.

2.10 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.

Example 2.10.1

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 particular 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-Yuan 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 → -1.01508, y → 0.994216, z → -2.64656},
 {x → -2.78473 + 0.767326 i, y → -2.16878 - 0.716577 i, z → -1.0773 + 1.35012 i},
 {x → -2.78473 - 0.767326 i, y → -2.16878 + 0.716577 i, z → -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.

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. A short version of this section is given in Volume 22 of *The Mathematica Journal*.

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

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

The key observation is

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

$$\text{Out[119]=}\ \left\{2+3\,t+4\,t^2,\ 5+6\,t+7\,t^2\right\}$$

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

$$\text{Out[121]=}\ \left\{\frac{2+3\,t+4\,t^2}{1+8\,t+9\,t^2},\ \frac{5+6\,t+7\,t^2}{1+8\,t+9\,t^2}\right\}$$

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

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

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

Note that this theorem implies that $T_d = \left\{t,\ t^2,\ \dots,\ t^d\right\}$ 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-Yuan Example

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

We use the example of L.Shen and C. Yuan in $\mathbb{R}^3$. [L.Shen, C.Yuan, 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-Yuan. 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

$\tilde{sy}$ = {-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}
```

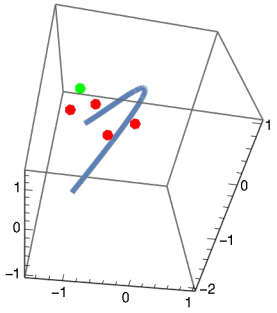
So

```
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 com - ponent 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-Yuan 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}

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

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-Yuan 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}]
```

Out[101]= $\{4.44089\times 10^{-16}, -1.77636\times 10^{-15}, 1.\}$

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` $[\{t, t\wedge 2, \ldots, t^d\}, 2, t, \{x1, \ldots, xd\}]$

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, 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-Yuan 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]= {(1 - t^4)/(1 + 2 t^2 + t^4), (4 t)/(1 + 2 t^2 + t^4)}
```

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

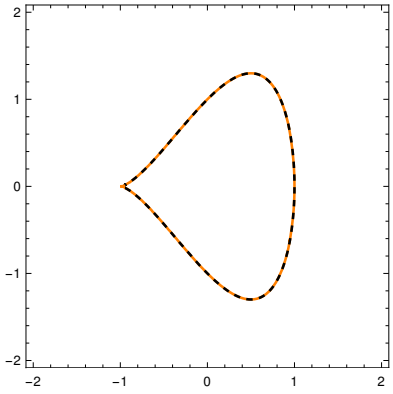
In[214]:=
```
piriformEq = FLTMD[{x3^2 − x2 x4, x2 x3 − x1 x4, x2^2 − x4, x1 x3 − x4, x1 x2 − x3, x1^2 − x2},
    piriformA, 4, {x1, x2, x3, x4}, {x, y}, dTol][[1]]
```

» 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]= $\{1.\,w - 1.\,x - 3.\,y - 1.\,z,\ 1. - 0.5\,x - 0.5\,y - 0.5\,z,$
$1.\,x^2 + 2.\,x\,y + 2.33333\,y^2 - 3.33333\,x\,z - 3.33333\,y\,z + 1.\,z^2\}$

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]]**

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

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

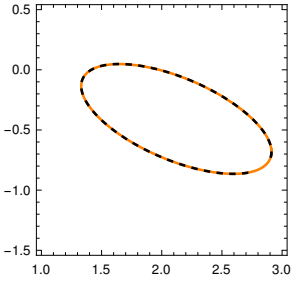
In[267]:= **fprojEq = FLTMD[fparEq, projA, 2, {x, y, z, w}, {x, y}, dTol][[1]]**

» Hilbert Function {1, 2, 2}

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]

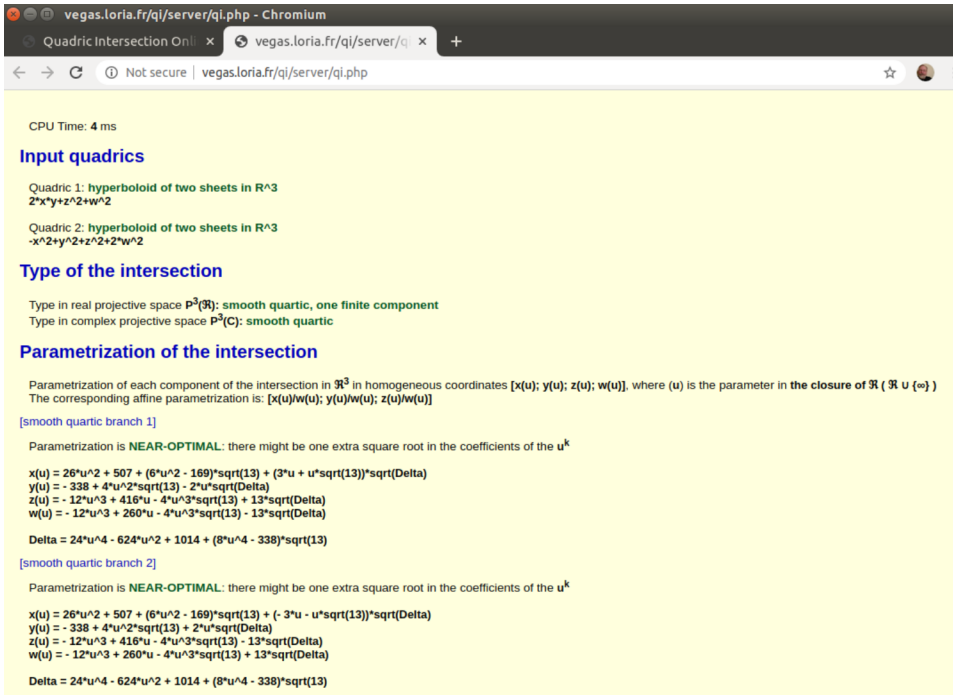
Out[268]=



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
```

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

Out[125]= $4+2x-3y-6z$

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

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

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 Wol-fram 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, ℧},
    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]]) &;
    ℧ = 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,
      ℧}]
```

In[117]:= `{hr, Ωr, ℧r} = 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]:=
```
υrx = Simplify[υr[{x, y}][[1]]]  (* non evaluative cell *)
υry = Simplify[υr[{x, y}][[2]]]
υrz = Simplify[υr[{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 Ʊ is a fraction of two quadratics in x,y 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

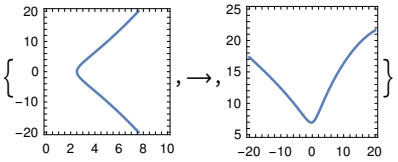
In[129]:=
```
{x, y} = TransformationFunction[Inverse[Aw]][{t, s}]
```

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 Ʊrx gives

In[124]:=
```
ux = Simplify[Ʊrx[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[Ory[TransformationFunction[Inverse[Aw]][{t, s}]]];
uz = Simplify[Orz[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

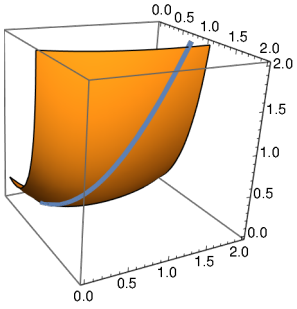
Out[238]= 2.51122

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]=
$$-1.33227\times 10^{-15}-1.80591\,t-1.44353\,t^2-1.45535\,t^3+(2.13788\times 10^{-14}+1.23957\times 10^{-14}\,t)\,\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[174]=
$$5.55112\times 10^{-17}-2.16386\,t+2.33569\,t^2+0.343561\,t^3+(-9.34093\times 10^{-15}+1.8324\times 10^{-14}\,t)\,\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[175]=
$$-6.53777\times 10^{-15}-1.59014\times 10^{-14}\,t-6.0631\times 10^{-16}\,t^2+5.91741\times 10^{-15}\,t^3+(2.31988\times 10^{-16}+3.60332\,t)\,\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

Out[176]=
$$4.44089\times 10^{-16}-2.52873\,t-2.59678\,t^2+1.\,t^3+(-2.09392\times 10^{-14}+3.03155\times 10^{-16}\,t)\,\sqrt{-1.09443-0.84291\,t+1.\,t^3}$$

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

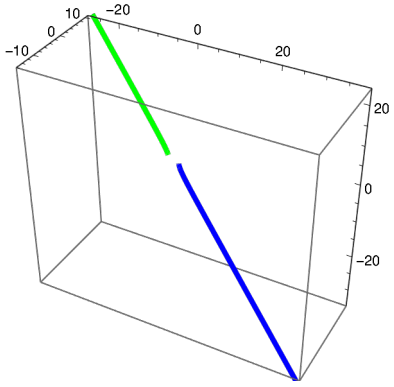
Out[170]= 1.29839

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]]
```

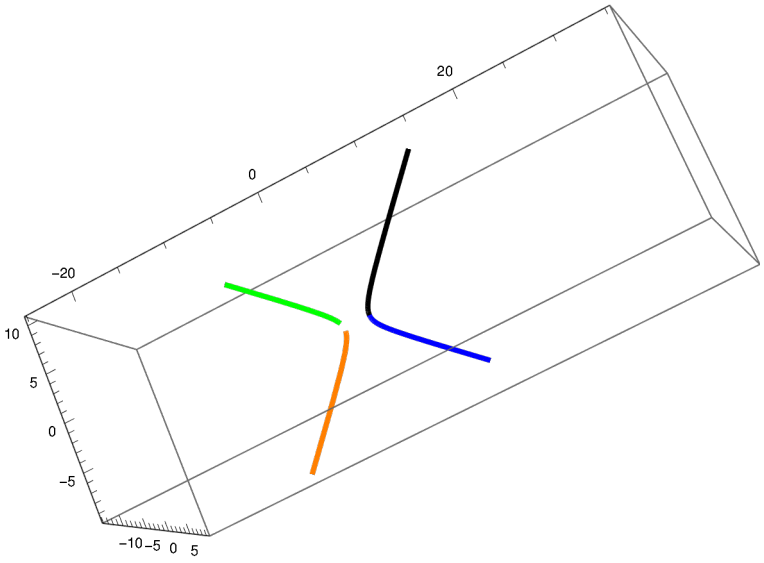
Out[196]=
$$\{\{t \to 3.35133\}, \{t \to 3.35133\}, \{t \to -0.754546\}, \{t \to -0.754546\},\ \{t \to 1.75617 \times 10^{-16} - 8.66265 \times 10^{-15}\, i\}, \{t \to 1.75617 \times 10^{-16} + 8.66265 \times 10^{-15}\, i\}\}$$

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, ℧3} = nsQSIC3D[Q3, {2, 4, 8}, {x, y, z}];
```

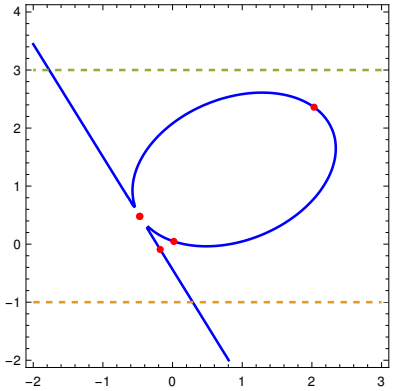
In[266]:=
```
h3
```

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 × 10^16, 3.66954 × 10^16}

Out[275]= {2., -2.59898 × 10^16, -4.53894 × 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 ℧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
```

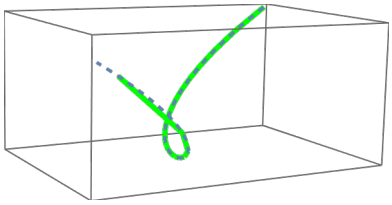
Out[280]=
```
{{-0.632381, 0.399905, -0.252893}, {-0.706213, 0.498737, -0.352214},
 {-0.780865, 0.609751, -0.476133}, {-0.857674, 0.735604, -0.630908},
 {-0.938997, 0.881715, -0.827928}, {-1.02819, 1.05717, -1.08697},
 {-1.12999, 1.27687, -1.44284}, {-1.2516, 1.56651, -1.96065}, {-1.40504, 1.97415, -2.77377},
 {-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*)
```

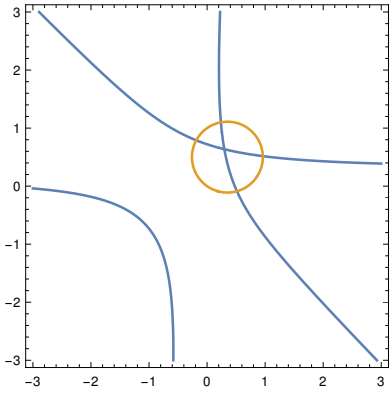
In[266]:=
```
h0 = 2.945656140191897` - 5.217838216917842` x -
     1.9646988724373289` x^2 - 0.07585852051106767` x^3 - 4.806985694928316` y -
     0.20976724723500872` x y + 8.782381871328202` x^2 y +
     1.478912121204738` y^2 + 7.944951077631805` x y^2 - 0.6576260858236058` y^3
```

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

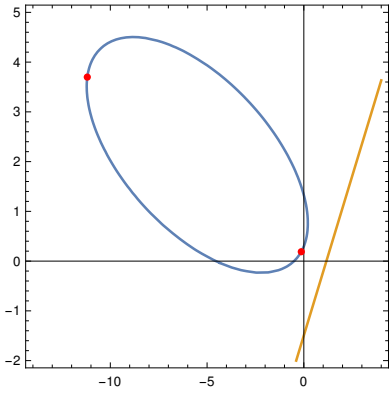
In[270]:= **cpqf = criticalPoints2D[qf, x, y]**

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

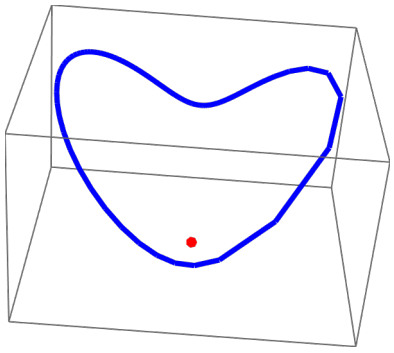
In[278]:= **pth1 = pathFinder2D[qf, cpqf⟦1⟧, cpqf⟦2⟧, .25, x, y, maxit → 70];**
pth2 = pathFinder2D[−qf, cpqf⟦1⟧, cpqf⟦2⟧, .25, x, y, maxit → 50];
pth4 = Join[pth1, Reverse[pth2]];

Now we lift to Q4

In[281]:= **PTH4 = ʊ4/@ 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[263]:=
```
Q5 = {x^2+y^2+z^2-16, 57-12x+4x^2+y^2-64z+16z^2};
```

We first project.

In[264]:=
```
h5 = FLTMD[Q5, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

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

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

Out[264]=
$$1. - 0.0344652\,x - 0.033403\,x^2 + 0.00677934\,x^3 + 0.00134825\,x^4 - 1.62713\,y - 0.0245903\,x\,y + 0.0241702\,x^2\,y - 0.00181159\,x^3\,y + 0.893879\,y^2 + 0.0164066\,x\,y^2 - 0.00620129\,x^2\,y^2 - 0.208146\,y^3 - 0.000832304\,x\,y^3 + 0.0196645\,y^4$$

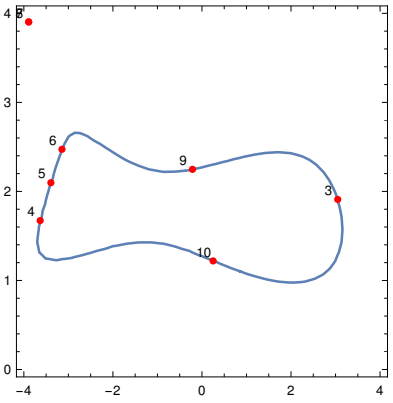
We first find and label critical points.

In[265]:=
```
cp5 = criticalPoints2D[h5, x, y];
ap5 = Association[Table[i → cp5[[i]], {i, 10}]]
```

Out[266]=
```
<| 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[267]:=
```
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[267]=



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

In[268]:=
```
fFiberMD[Q5, prd3D, cp5[[1]], {x, y, z}, 1.*^−6]
fFiberMD[Q5, prd3D, cp5[[7]], {x, y, z}, 1.*^−6]
```

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

Out[268]= {}

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

Out[269]= {}

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[270]:=
```
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[275]:=
```
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;
```

» multiple fiber points {−3.14407, 2.47282}

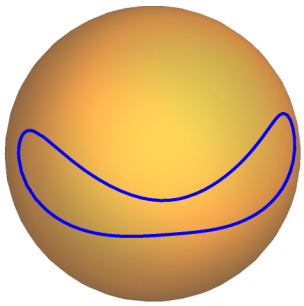
» (3) no point in fiber at {−3.63348, 1.67267}

» (3) no point in fiber at {−3.63348, 1.67267}

» (3) no point in fiber at {−3.14407, 2.47282}

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

In[280]:=
```
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]
```



3.2.4 Some more examples from [TWMW].

We give some more examples from the classification of QSIC in [TWMW]. In Example 2.5.3.1 we already saw that the union of the twisted cubic and a line through two points was a QSIC.

In[281]:=

Out[281]=

In other cases it will be enough just to project to the plane.

Example 6

In[282]:=
```
Q6 = {x^2 + y^2 + z^2 - 1, x^2 + 2 y^2};
```

We project with our pseudo-random projection.

In[283]:=
```
h6 = FLTMD[Q6, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

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

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

Out[283]=
$$1. + 1.27881\,x^2 + 0.903815\,x^4 + 0.162081\,x\,y - 0.451232\,x^3\,y - 2.04542\,y^2 - 1.14578\,x^2\,y^2 - 0.165762\,x\,y^3 + 1.04594\,y^4$$

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

In[284]:=
```
cp6 = criticalPoints2D[h6, x, y]
```

Out[284]=
```
{{-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[285]:=
```
fFiberMD[Q6, prd3D, cp6[[1]], {x, y, z}, 1.*^-6]
fFiberMD[Q6, prd3D, cp6[[3]], {x, y, z}, 1.*^-6]
```

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

Out[285]= {}

» (1) no point in fiber at {0.636105, 1.13489}

Out[286]= {}

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

In[287]:=
```
fFiberMD[Q6, prd3D, cp6[[5]], {x, y, z}, 1.*^-12]
fFiberMD[Q6, prd3D, cp6[[7]], {x, y, z}, 1.*^-12]
```

Out[287]= $\{\{5.80425\times 10^{-13}, 1.86406\times 10^{-13}, 1.\}\}$

Out[288]= $\{\{-4.94937\times 10^{-13}, -1.58706\times 10^{-13}, -1.\}\}$

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[289]:=
```
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[290]:=
```
criticalPoints3DC[Q6, {x, y, z}]
```

Out[290]= {{-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.

In[291]:=
```
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[292]:=
```
h7 = FLTMD[Q7, fprd3D, 4, {x, y, z}, {x, y}, dTol][[1]]
```

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

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

Out[292]=
$$1. - 1.80651\,x^2 + 0.350558\,x^4 + 0.653265\,x\,y - 1.44199\,x^3\,y - 2.04542\,y^2 + 1.56429\,x^2\,y^2 - 0.668101\,x\,y^3 + 1.04594\,y^4$$

The result is a circle and two lines in the plane.

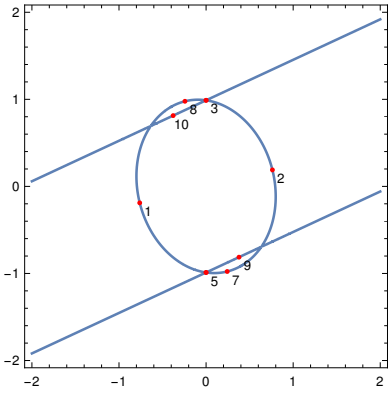
In[293]:=
```
cp7 = criticalPoints2D[h7, x, y];
acp7 = <| Table[i → Chop[cp7[[i]]], {i, 10}] |>
```

Out[294]=
```
<| 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[295]:=
```
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[295]=



In[296]:= We lift points on the lines to $\mathbb{R}^3$.

In[296]:=
```
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[296]= $\{6.66134\times 10^{-16}, 2.22045\times 10^{-16}, 1.\}$

Out[297]= $\{1.23871, -5.55112\times 10^{-16}, 1.\}$

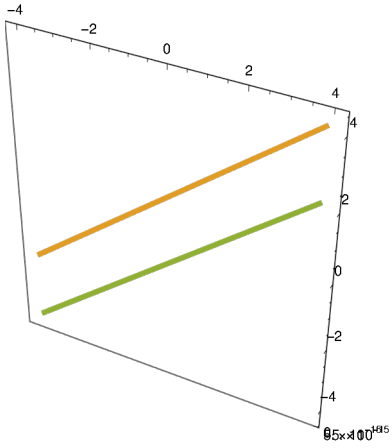
Out[298]= $\{-6.66134\times 10^{-16}, -3.33067\times 10^{-16}, -1.\}$

Out[299]= $\{-1.23871, -6.66134\times 10^{-16}, -1.\}$

We can now plot in $\mathbb{R}^3$

In[300]:=
```
ParametricPlot3D[{{0, Cos[t], Sin[t]}, p1+t*p2, p3+t*p4}, {t, -Pi, Pi}]
```

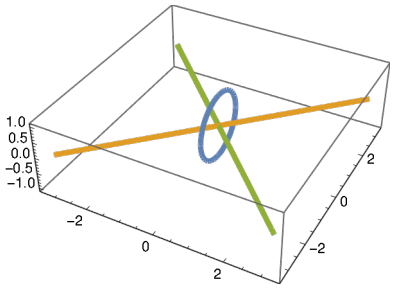
Out[300]=



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[301]:=
```
ParametricPlot3D [{{0, Cos[t], Sin[t]}, {t, t, 0}, {t, -t, 0}}, {t, -Pi, Pi}]
```

Out[301]=



In[302]:=

This is no longer a QSIC. By Example 2.5.3.3 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.

3.2.5 A numerical Example of a degenerate QSIC.

In this example we show that our direct method works well for numerical QSIC even in the singular case.

In[303]:=
```
K =
{-3.0343373677870256` + 4.760714817579225` x - 0.8673102054064943` x^2 -
    2.3198076300045427` y + 1.8198277283436077` x y - 0.4433840726939407` y^2 +
    3.4471924447384925` z - 2.7042312969112072` x z + 1.317721526336166` y z +
    0.02094474750770381` z^2, 0.00005226006460796005` - 0.014540466884057102` x +
    1.0114088969866952` x^2 + 0.00003995379614314041 6` y -
    0.005558229348449886` x y + 7.636355973833214` *^-6 y^2 -
    0.00005937062298695252` z + 0.00825942891482889` x z -
    0.00002269497546077125 3` y z - 0.9999831378371788` z^2}
```

Out[303]=
$$\{-3.03434 + 4.76071\,x - 0.86731\,x^2 - 2.31981\,y + 1.81983\,x\,y - 0.443384\,y^2 + 3.44719\,z - 2.70423\,x\,z + 1.31772\,y\,z + 0.0209447\,z^2,\ 0.0000522601 - 0.0145405\,x + 1.01141\,x^2 + 0.0000399538\,y - 0.00555823\,x\,y + 7.63636\times 10^{-6}\,y^2 - 0.0000593706\,z + 0.00825943\,x\,z - 0.000022695\,y\,z - 0.999983\,z^2\}$$

We check for infinite points

In[304]:=
```
ipK = infiniteRealPoints3D[K, {x, y, z}]
```

Out[304]=
$$\{\{-0.329006, -0.885872, 0.327087, 0\}, \{0.171804, 0.970211, 0.170802, 0\}, \{0.498804, 0.706713, 0.501749, 0\}, \{-0.472181, 0.742597, 0.474969, 0\}\}$$

Our standard method from Chapter 2 is to project on the plane and lift back up to plot.

We choose a random projective FLT for projection, but for replication we give it here

In[305]:=
```
A =
{{-0.6934276433346329`, 0.07381779176491898`, -0.7573184238468178`,
    -0.12486357381645385`}, {-0.41481421719883427`, -0.24723634736560696`,
    0.5906825357052634`, 0.21818634914942425`}, {-0.6431194318709657`,
    -0.3715495908236628`, 0.3379270707587114`, -0.9578085718413809`}}
```

Out[305]=
$$\{\{-0.693428, 0.0738178, -0.757318, -0.124864\}, \{-0.414814, -0.247236, 0.590683, 0.218186\}, \{-0.643119, -0.37155, 0.337927, -0.957809\}\}$$

In[306]:=
```
K2 = FLTMD[K, A, 4, {x, y, z}, {x, y}, 1.*^-9][[1]]
```

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

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

Out[306]=
$$1. - 9.1079\,x + 23.8809\,x^2 - 23.5535\,x^3 + 7.31227\,x^4 - 3.41374\,y + 18.0057\,x\,y - 26.7585\,x^2\,y + 11.1384\,x^3\,y + 3.39349\,y^2 - 10.1319\,x\,y^2 + 6.36118\,x^2\,y^2 - 1.27861\,y^3 + 1.61432\,x\,y^3 + 0.1536\,y^4$$

We map our infinite points of K to K2. We also intersect K2 with the line $y = -2$

In[307]:=
```
ipK2 = fltiMD[#, A] &/@ ipK
sol2 = {x, y} /. NSolve[{K2, y + 2}][[{1, 2, 4, 3}]]
```

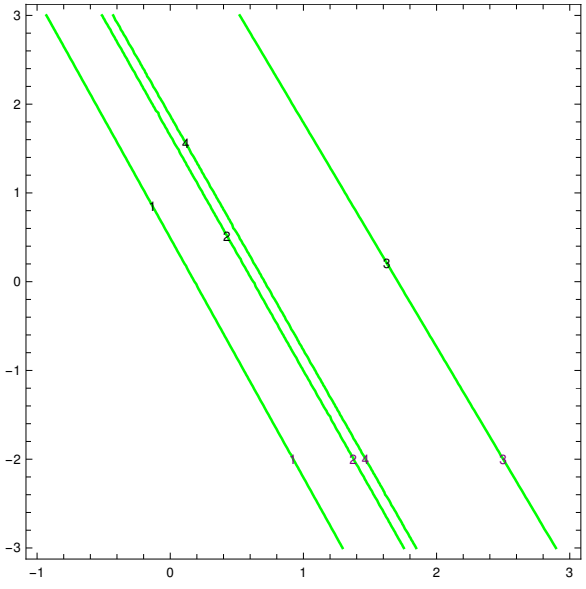
Out[307]= {{-0.130453, 0.842512}, {0.427985, 0.508764}, {1.62802, 0.206039}, {0.119715, 1.55542}}

Out[308]= {{0.922963, -2.}, {1.37574, -2.}, {2.5008, -2.}, {1.46808, -2.}}

We now plot

In[309]:=
```
ContourPlot[{K2 == 0}, {x, -1, 3}, {y, -3, 3}, ContourStyle → Green,
  Epilog → {{Black, PointSize[Medium], Table[Text[i, ipK2[[i]]], {i, 4}]},
    {Purple, PointSize[Medium], Table[Text[i, sol2[[i]]], {i, 4}]}}, ImageSize → Medium]
```

Out[309]=



We see we have four apparently parallel lines, since FLT preserve lines we can expect 4 lines in **K**. Importantly, note that we permuted our set **sol2** so that the indices of the two point sets match up on each line, this gives us two points on each line so we can lift back to **K**. Our first set of points come from the infinite points of K which can be viewed as slopes [Section 1.1 of my plane curve book]. There are two problems, first fiber my lifting function **fFiberMD** only works for linear projections. Secondly when we plot we need

nice endpoints which will must be chosen when we plot. The first problem is handled nicely with my **factorFLT** function [see 2.7.2] and for the second we will find equations for each line.

```
In[310]:= {P, B} = factorFLT[A];
          pl = tM2M[P];
          pl // MatrixForm
          B // MatrixForm
```

Out[312]//MatrixForm=

$$\begin{pmatrix} -0.693428 & 0.0738178 & -0.757318 \\ -0.414814 & -0.247236 & 0.590683 \end{pmatrix}$$

Out[313]//MatrixForm=

$$\begin{pmatrix} 1. & 0 & 0 & 0.0730708 \\ 0 & 1. & 0 & -1.0051 \\ 0 & 0 & 1. & 0 \\ -0.643119 & -0.37155 & 0.337927 & -0.957809 \end{pmatrix}$$

We then have the intermediate curve **K3** below which we do not need to fully describe.

```
In[314]:= K3 = FLT3D[K, B, {x, y, z}]
```

Out[314]=

$$\{-3.81978 - 10.1431\,x - 5.81064\,x^2 - 0.234581\,y - 0.27421\,x\,y - 0.00322577\,y^2 - 1.28772\,z - 1.76506\,x\,z - 0.0406579\,y\,z + 0.8923\,z^2,\ 0.00422664 + 0.136201\,x + 1.09725\,x^2 + 0.0027815\,y + 0.0448162\,x\,y + 0.000457618\,y^2 - 0.00232266\,z - 0.0374233\,x\,z - 0.000764258\,y\,z - 0.999681\,z^2\}$$

We can now lift the points of **sol2** above to **K**.

```
In[315]:= kpts = fltMD[fFiberMD[K3, pl, #1, {x, y, z}, 1.*^-8][[1]], Inverse[B]] &/@ sol2
```

Out[315]=

$$\{\{0.158762, -2.18855, -0.157835\}, \{0.148034, -1.78005, 0.14717\}, \{0.488106, -1.92447, 0.490987\}, \{0.630171, -3.60709, -0.633891\}\}$$

Next we can describe the lines on K by one Mathematica function

```
In[316]:= l := lineMD[kpts[[#]], ipK[[#]], {x, y, z}] &
```

We don't actually need to see the equations but as an example

```
In[317]:= l[1]
```

Out[317]=

$$\{0.277262 + 0.514444\,x + 0.105986\,y + 0.804512\,z,\ 0.720243 - 0.626116\,x + 0.27532\,y + 0.115878\,z\}$$

The following utility functions find points on these lines by specifying only the x-coordinate.

```
In[318]:= u := {x, y, z} /. NSolve[Append[l[#1], x + #2]][[1]] &
          v := {x, y, z} /. NSolve[Append[l[#1], x + #2]][[1]] &
```

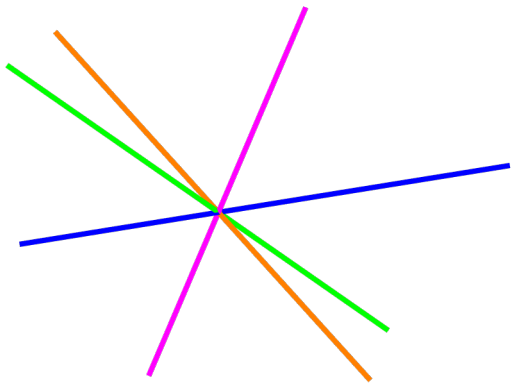
By trial we can find nice endpoints for plotting

In[320]:=
$$\begin{pmatrix} u1 & u2 & u3 & u4 \\ v1 & v2 & v3 & v4 \end{pmatrix} = \begin{pmatrix} u[1, 0.7`] & u[2, 0.3`] & u[3, 0.6`] & u[4, 0.8`] \\ v[1, -0.4`] & v[2, -0.3`] & v[3, -0.6`] & v[4, -0.8`] \end{pmatrix}$$

Out[320]=
{{{−0.7, −4.50082, 0.695916}, {−0.3, −4.31018, −0.29825},
{−0.6, −3.46611, −0.603542}, {−0.8, −1.35787, 0.804723}}, {{0.4, −1.539, −0.397666},
{0.3, −0.921867, 0.29825}, {0.6, −1.76594, 0.603542}, {0.8, −3.87418, −0.804723}}}

In[197]:=
```
Graphics3D[{{Blue, Thick, Line[{u1, v1}]}, {Green, Thick, Line[{u2, v2}]},
  {Orange, Thick, Line[{u3, v3}]}, {Magenta, Thick, Line[{u4, v4}]}}, Boxed → False]
```

In[321]:=



So we see **K** consists of 4 lines through a point. What is most interesting is that we never actually used that point in our construction. Note in particular that these lines were given numerically so, for example,

In[322]:=
```
NSolve[Join[l[1], l[2]]]
```

Out[322]=
{}

finding the intersection of any two of these lines is an inconsistent problem to **NSolve**. But it is not to our methods. One possibility is to consider the linear equation set

In[323]:=
```
F = {l[1], l[2], l[3], l[4]}
```

Out[323]=
```
{{0.277262 + 0.514444 x + 0.105986 y + 0.804512 z, 0.720243 - 0.626116 x + 0.27532 y + 0.115878 z},
  {-0.258819 - 0.33616 x - 0.098936 y + 0.900123 z,
    0.468581 - 0.849601 x + 0.179119 y - 0.16287 z},
  {-0.214984 - 0.629812 x - 0.0821798 y + 0.741866 z,
    0.853126 - 0.402957 x + 0.326115 y - 0.0587414 z},
  {-0.0354267 + 0.698008 x - 0.0135422 y + 0.715085 z,
    0.867771 + 0.288235 x + 0.331713 y - 0.232079 z}}
```

and find an H-Basis.

In[324]:=
```
sys = hBasisMD[F, 1, {x, y, z}, 1.*^-6]
```

» Initial Hilbert Function {1, 0}

» Final Hilbert Function {1, 0}

Out[324]=
```
{1. x, 2.61602 + 1. y, 1. z}
```

Solving this last system for the singular point

In[325]:=
```
spt = {x, y, z} /. Solve[sys == 0][[1]]
```

Out[325]=
```
{0., -2.61602, 0.}
```

Note

In[326]:=
```
tangentVectorMD[K, spt , {x, y, z}]
```

» Hilbert Function {1, 3, 4, 4, 4}

» No unique tangent vector at {0., -2.61602 , 0.}

Note the multiplicity of the singular point is correctly given as 4. Thus all this numerical work does give a consistent story.

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+3z-2x+2);
q2 = z-x^2;
Q = {q1, q2}
```

Out[127]=
$$\{-2+2\,x+y^2-3\,z-z^2,\; -x^2+z\}$$

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, ℧} = 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.

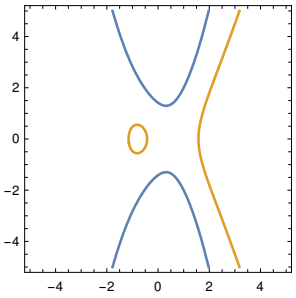
Thus we have a *birational equivalence* between the quartic curve f and the cubic curve wh.

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.

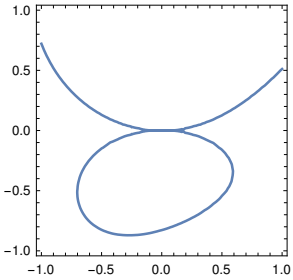
In[168]:= **ContourPlot[{f == 0, wh == 0}, {x, −5, 5}, {y, −5, 5}, ImageSize → Small]**

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 $\mathbb{R}^3$

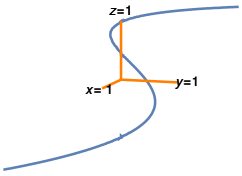
In[116]:=
```
F1 = {f1, y - x z};
```

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.

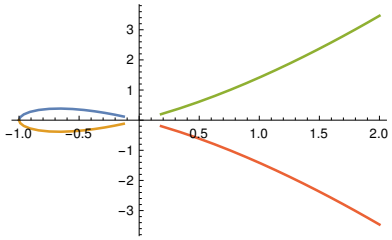
In[128]:=
```
B /. Thread[{x, y, z} → {0, 0, 3.13}]
```

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

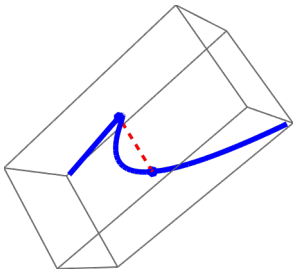
In[143]:=
```
excpt1 = fFiberMD[B1, {{1, 0, 0}, {0, 1, 0}}, {0, 0}, {x, y, z}, dTol]
```

» 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.

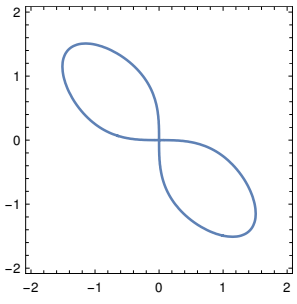
3.3.2.2 A lemniscate

Consider the lemniscate

In[143]:=
```
f2 = x^4 + 4 x y + y^4;
```

In[144]:=
```
ContourPlot[f2 == 0, {x, -2, 2}, {y, -2, 2}, ImageSize → Small]
```

Out[144]=



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]=
$$\{\{9.01348\times 10^{-14}, 0.\}, \{-9.0847\times 10^{-14}, 0.\}, \{3.3185\times 10^{-15}, 0.\},$$
$$\{-1.249\times 10^{-14}, 0.\}, \{-1.02562\times 10^{-12}, 1.11022\times 10^{-16}\}\}$$

In[ ]:=
```
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.}

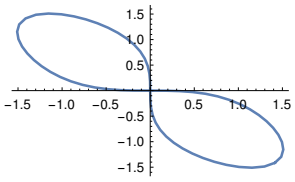
Thus the blowup contains the image of Φ but not other points on the exceptional line. We can plot our blow-up B.

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

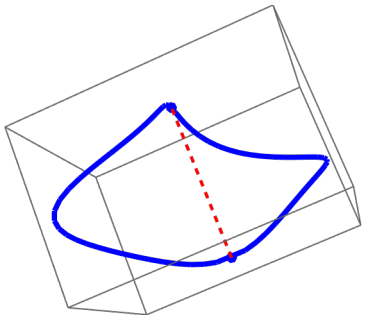
In[174]:=
```
excpt2 = fFiberMD[B2, {{1, 0, 0}, {0, 1, 0}}, {0, 0}, {x, y, z}, dTol]
```

» 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;
```

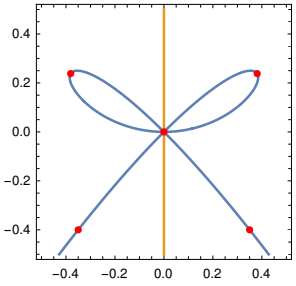
In[187]:=
```
cpf3 = DeleteDuplicates[Chop[criticalPoints2D[f3, x, y]]]
pts3 = {x, y} /. NSolve[{f3, y + .4}, {x, y}, Reals]
```

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]
```

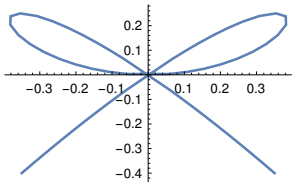
» multiple fiber points {0, 0}

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]
```

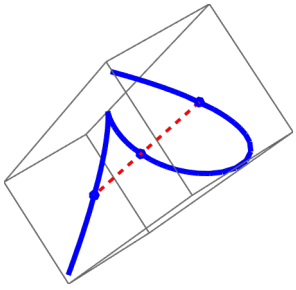
Out[247]=



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]=



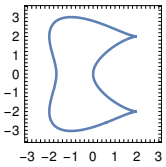
3.3.2.4 The Bicuspid

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
```

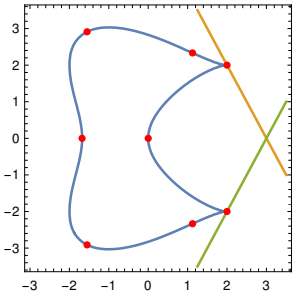
The critical points of the bicuspid are

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

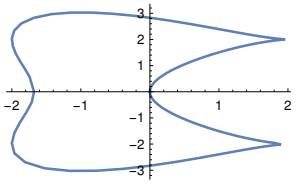
```
In[141]:= Chop[F4 /. Thread[{x, y, z, w} → {2, 2, z, w}]]
          F4 /. Thread[{x, y, z, w} → {2, -2, z, w}]
Out[141]= {0, 0, -4 + 4. w}

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]
```

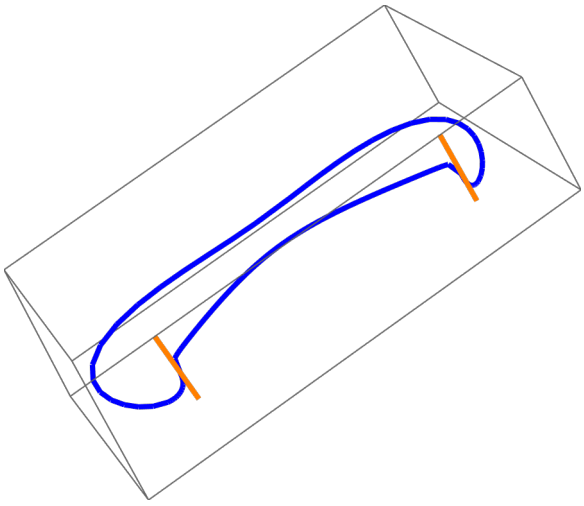
Out[154]= 138

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 = B[[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 → −0.527983 − 0.0480729 i}, {w → −0.527983 + 0.0480729 i}, {w → 0.76187 − 0.813969 i}, {w → 0.76187 + 0.813969 i}, {w → 3.19801 − 7.05408 i}, {w → 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 → −3.64706}, {w → −0.546267}, {w → −0.287786 − 2.11328 i}, {w → −0.287786 + 2.11328 i}, {w → 0.049907}, {w → 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

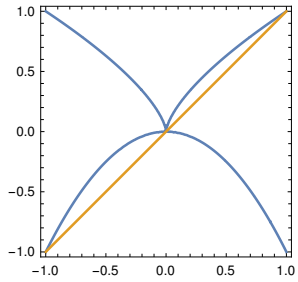
We consider the singularity at {0, 0} of

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]=

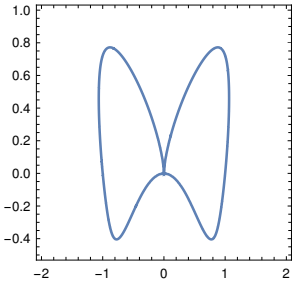


This is technically a reducible curve and we only discuss genus for irreducible curves, however we can still blow up. This will be essentially the singularity of a higher degree irreducible curve such as

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]])] &**

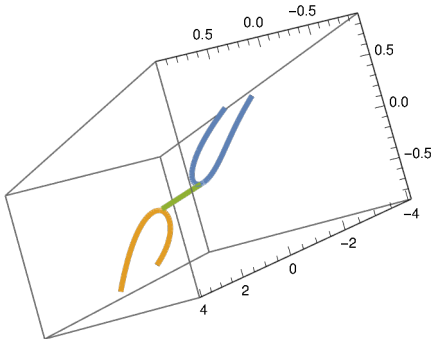
Note that

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}]**



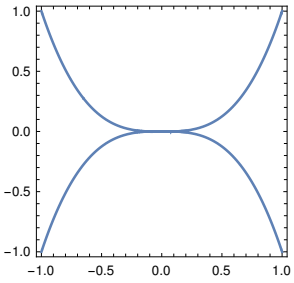
3.3.2.5 A harder compound example

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

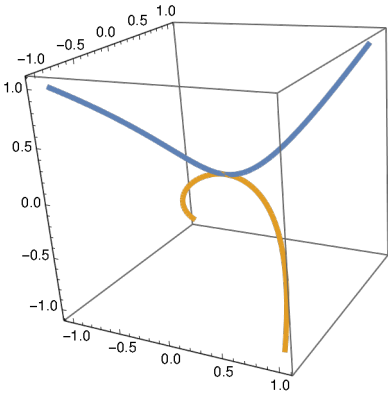
In[208]:=
```
F6 /. Thread[{x, y, z} → Φ[{t, t^3}]]
F6 /. Thread[{x, y, z} → Φ[{t, -t^3}]]
```

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}

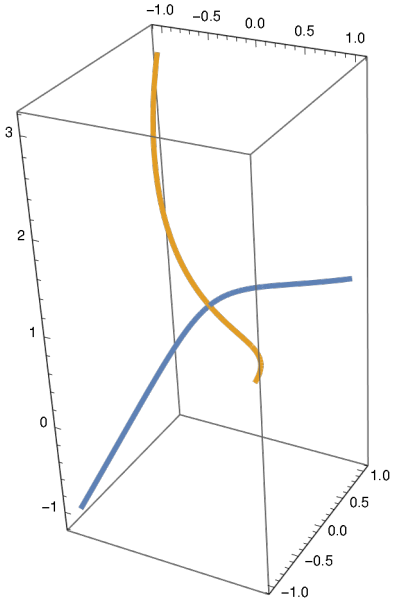
Out[131]= {0, 0, 0}

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]= $-0.628167 \times (-1 + w) - 0.441975\,x + 0.453086\,y - 0.545897\,z$

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}

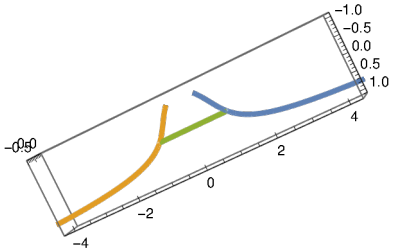
Out[135]= {0, 0, 0, 0}

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.

3.3.3 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.

3.3.4 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 when working numerically. I give the numerical version in section 1.2.1 and 2.7.2:

For random numerical 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.

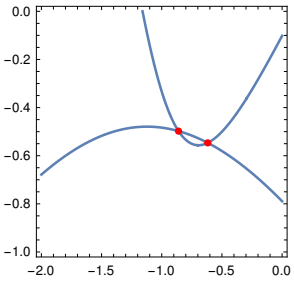
Example 3.3.4.1: A nice example is the bow curve 3.3.2.3. We found the non-singular blow-up to be curve

```
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]=

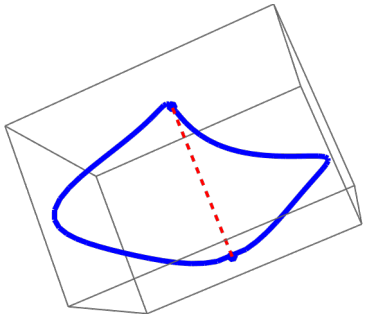


The third singular point {-1.98573, -11.9106} is an isolated singularity. Since $\nu = 3$ and $d = 4$ then $g = 0$ which is what we expect given this is a parameterized curve.

Example 3.3.4.2: 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

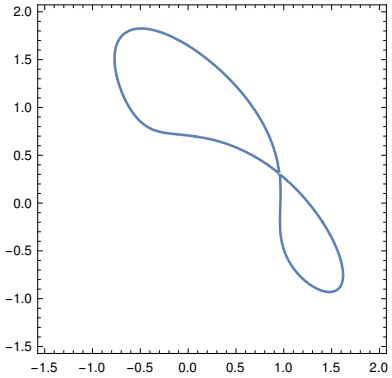
In[312]:=
```
A2 = Append [Orthogonalize [RandomReal [{−1, 1}, {2, 4}]], {0, 0, 0, 1}]
h2 = FLTMD [B2, A2, 6, {x, y, z}, {x, y}, 1.*^−7 ][[1]]
```

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.

3.3.4.3 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.

3.3.5 Examples of non-singular Curves of genus 0 - 6

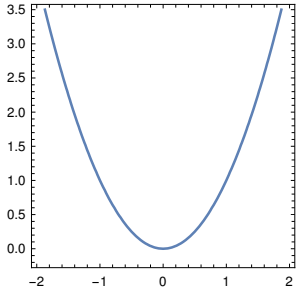
Now that we have developed our software and theory I end by plotting an example of a curve of each genus from 0 to 6. We don't show work but we

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]
```

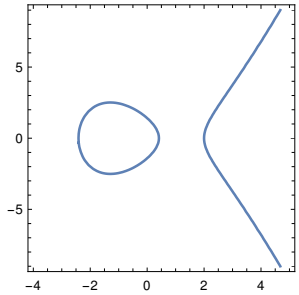
Out[128]=



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]=

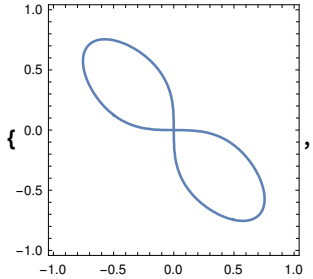


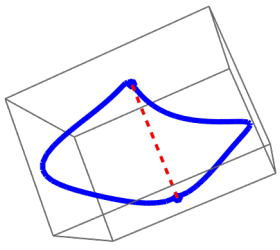
Genus 2, Lemniscate $x^4 + xy + y^4$

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]=

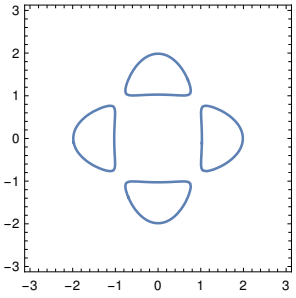
{ 

,  }

Genus 3, Klein Curve $\left(x^2+\frac{y^2}{4}-1\right)\left(\frac{x^2}{4}+y^2\right)=-.04$

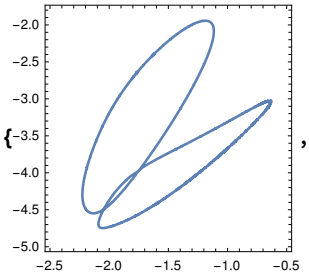
In[122]:=
```
ContourPlot[(x^2+y^2/4-1) (x^2/4+y^2-1) == -.04,
  {x, -3, 3}, {y, -3, 3}, ImageSize→Small]
```

Out[122]=

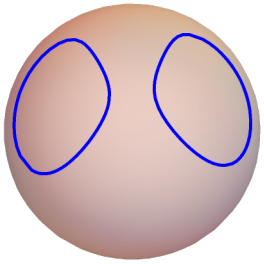


Genus 4 (See Example 3.3.4.3) g4 on the left, {f1, f2} plotted on f1 on the right. In addition to the singular points shown in the plot of g4 there are two isolated real singular points and 2 complex singular points.

{

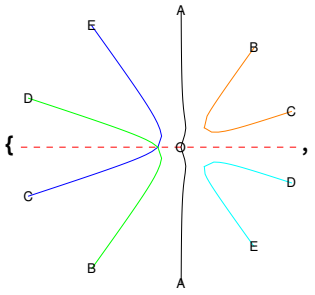


,

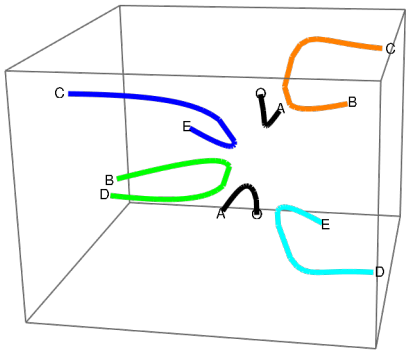


}

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.)



,



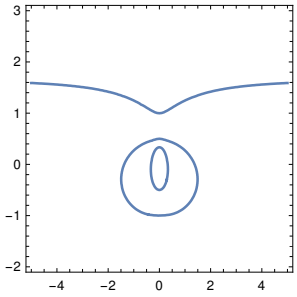
}

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]=



4 References

1. B.H. Dayton, *A Numerical Approach to Real Algebraic Curves, with the Wolfram Language*, Wolfram-Media, 2018.
2. B.H. Dayton, *A Wolfram Language Approach to Real Numerical Plane curves* https://www.mathematica-journal.com/2018/08/29/a-wolfram-language-approach-to-real-numerical-algebraic-plane-curves/, 2018.
3. B.H. Dayton, T.Y. Li, Z. Zeng, *Multiple Zeros of Non-linear Systems*, Mathematics of Computation, 80, no. 276, pp. 2143-2168, 2011. Free access at https://www.ams.org/mcom/2011-80-276/S0025-5718-2011-02462-2/.
4. Wolfram-alpha: https://www.wolframalpha.com/input/?i=Viviani+Curve.
5. J. Harris, *Algebraic Geometry, A first Course*, Graduate Texts in Mathematics, Springer, 2010.
6. D. Adrovic and J. Verschelde, *Tropical Approach to the Cyclic n-Roots Problem*, Presentation 2013 http://homepages.math.uic.edu/~adrovic/jmm13a.pdf (accessed May 9,2020).
7. Y. Yang and X. Bican, *A Hybrid Procedure for Finding Real Points on a Real Algebraic Set,* J Syst Sci Complex (2019) 32: 185–204.
8. F.S. Macaulay, *The Algebraic Theory of Modular Systems,* Cambridge University Press, 1916.
9. Z.Zeng, B.Dayton, *The approximate GCD of inexact polynomials*, Proceedings of ISSAC 2004, ACM.
10. D.Cox, J.Little, D. O'Shea, *Using Algebraic Geometry,* Graduate Texts in Mathematics 185, Springer, 1998.
11. B.H. Dayton, *Ideals of numeric representations of Unions of Lines*, in *Interactions of Classical and Numerical Algebraic Geometry*, D.Bates, G-M . Besana,S. Di Rocco and C.W.Wampler Eds, Contemporary Mathematics 496, AMS, 2009. (see https://barryhdayton.space/NumericLines.pdf and the appendix).
12. Xiangcheng Yu., PHC pack, https://kepler.math.uic.edu
13. S. Telen, B. Mourrain, B. van Barel, *Solving polynomial systems via truncated normal forms*, Siam J. Matrix Anal. Appl. Vol39 no3 (2018) pp. 1421-1447.
14. L. Shen, C. Yuan, *Implicitization using Univariate Resultants*, J Sys Sci Complex (2010) 23, pp. 804-814.
15. D.J. Bates, J. D. Hauenstein, A.J. Sommese, C.W.Wampler, *Numerically solving Polynomial Systems with Bertini*, SIAM, 2013.
16. C. Tu, W. Wang, B. Mourrain, J. Wang, *Using signature sequences to classify intersection curves of two quadrics*, Computer Aided Geometric Design 26 (2009), pp. 317-335.
17. L . Dupont, D. Lazard, S. Lazard and S. Petitjean,*Near-optimal parameterization of the intersection of quadrics, Parts I,II,III,* J. Symbolic Comput. 3(43), 2008, pp. 168–232. See also http://vegas.loria.fr/qi/server.
18. B.H. Dayton, *Algorithms for real numerical varieties with application to parameterizing quadratic surface intersection curves, Albanian J. Math, Vol. 7 no. 2, 2013.* (see http://barryhdayton.space/RQSIC.pdf).
19. B. H. Dayton*, Theory of Equations, https://barryhdayton.space/theoryEquations See specifically Chapter 6.*
20. S. Abhyankar*, Algebraic Geometry for Scientists and Engineers, AMS, 1990.*

21. A. Bonifant, J.Milnor, *Group Actions, Divisors , and Plane Curves, Bulletin of AMS, Volume 57, Number 2, April 2020, Pages 171–267*
https://www.ams.org/journals/bull/2020-57-02/S0273-0979-2020-01681-2/S0273-0979-2020-01681-2.pdf

22. B. Mazur, *Arithmetic on Curves, Bulletin of AMS 14, 1986.*
https://www.ams.org/journals/bull/1986-14-02/S0273-0979-1986-15430-3/S0273-0979-1986-15430-3.pdf

23. R. Hartshorne, *Algebraic Geometry, Springer-Verlag, 1977.*

24. *B.H. Dayton, Degree versus Dimension for Rational Parametric Curves,* Mathematica Journal 22, Free PDF at https://content.wolfram.com/uploads/sites/19/2020/09/Dayton.pdf , Mathematica Notebook version also available.