5/23/18

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

Summary of *Algebraic Space Curves*.

www.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 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 and 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.

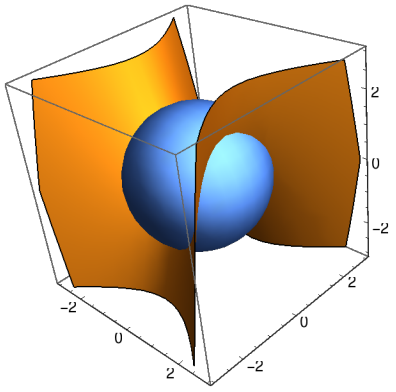
1 | Naive Case: curves in $\mathbb{R}^3$

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

Example 1.

In[51]:=
```
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[51]=



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

In[55]:=
```
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 $-\frac{113492\,x}{178835} - \frac{121484\,y}{178835} + \frac{171802\,z}{178835} == 1.$

Out[55]= {−1.43465, 1.09213, 0.865473}

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[61]:=
```
nv1 = Grad[f11, {x, y, z}] /. Thread[{x, y, z} → p1]
nv2 = Grad[f12, {x, y, z}] /. Thread[{x, y, z} → p1]
```

Out[61]= {−2.8693, −2.18425, −1}

Out[62]= {−2.8693, 2.18425, 1.73095}

The tangent vector is simply the cross product

In[63]:=
```
tv1 = Cross[nv1, nv2]
```

Out[63]= {−1.59657, 7.83589, −12.5345}

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

In[52]:=
```
TangentVector3D[{f_, g_}, p_, x_, y_, z_] := Module[{n1, n2, bi, tol},
  tol = 1.*^-8;
  n1 = Grad[f, {x, y, z}] /. Thread[{x, y, z} → p];
  n2 = Grad[g, {x, y, z}] /. Thread[{x, y, z} → p];
  bi = Cross[n1, n2];
  If[Norm[bi] < tol, Echo[p, "No tangent vector at "]; bi, Normalize[bi]]]
```

In[69]:=
```
TangentVector3D[{f11, f12}, p1, x, y, z]
```

Out[69]= {−0.107381, 0.527021, −0.843041}

We took a somewhat arbitrary tolerance, that can be changed. Also note the direction of the tangent vector can be changed by switching *f*, *g* or multiplying either by -1.

A point with a tangent vector is called *regular* while one without a tangent vector is called *singular*. As noticed above our break between regular and singular is somewhat arbitrary and might be adjusted for a specific situation.

In this naive case we can also get critical points

In[70]:=
```
NaiveCritPts3D[{f_, g_}, x_, y_, z_] := Module[{J},
  J = D[{f, g, x^2 + y^2 + z^2}, {{x, y, z}}];
  {x, y, z} /. NSolve[{f, g, Det[J]}, {x, y, z}, Reals]]
```

For our Example 1 above this does not work since the equation f12=0 is a sphere about the origin so every point is crittical

In[71]:=
```
NaiveCritPts3D[{f11, f12}, x, y, z]
```

NSolve: Infinite solution set has dimension at least 1. Returning intersection of solutions with $-\frac{113492\,x}{178835}-\frac{121484\,y}{178835}+\frac{171802\,z}{178835}==1.$

Out[71]= {{−1.43465, 1.09213, 0.865473}, {−1.15996, −1.4471, −0.748597}}

We can modify this by

In[9]:=
```
{r1, r2, r3} = RandomReal[{.7, 1.3}, 3];
J = D[{f11, f12, r1 x^2 + r2 y^2 + r3 z^2}, {{x, y, z}}];
critpts = {x, y, z} /. NSolve[{f11, f12, Det[J]}, {x, y, z}, Reals]
```

Out[11]= {{0., 1.24962, −1.56155}, {−1.24962, 0., 1.56155}, {−1.44667, −1.35882, 0.246463},
{1.44667, 1.35882, 0.246463}, {1.24962, 0., 1.56155}, {0., −1.24962, −1.56155},
{1.44667, −1.35882, 0.246463}, {−1.44667, 1.35882, 0.246463}}

With our original NaiveCritPts3D we can also find points on the curve by picking an arbitrary point and finding the point on the curve closest to it.

In[53]:=
```
closestpoint3D[{f_, g_}, p_, x_, y_, z_] := Module[{J, sol},
    J = D[{f, g, (x - p[[1]])^2 + (y - p[[2]])^2 + (z - p[[3]])^2}, {{x, y, z}}];
    sol = {x, y, z} /. NSolve[{f, g, Det[J]}, {x, y, z}, Reals];
    MinimalBy[sol, Norm[# - p] &][[1]]
  ]
```

As long a the entered point is not zero our particular equation f_{12} will not interfere, for example the closest point on the curve to {1,1,1} is

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

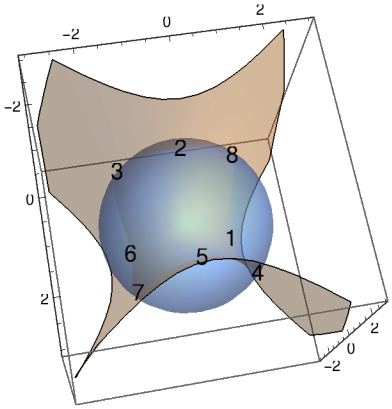
Out[80]=
```
{1.40516, 0.962189, 1.04867}
```

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

In Example 1 we plot

In[28]:=
```
Show[ContourPlot3D[{f11 == 0, f12 == 0}, {x, -3, 3},
    {y, -3, 3}, {z, -3, 3}, Mesh → None, ContourStyle → Opacity[.4]],
  Graphics3D[Table[{Text[Style[i, FontSize → 14], critpts[[i]]]}, {i, 8}]]]
```

Out[28]=



Perhaps by manually rotating this plot it appears we travel through these points in the order 1, 4, 5, 7, 6, 3 ,2, 8, 1.

We have two functions we can use to trace from one critical point to the next. Note that we can never trace out of a singular point, but these will attempt to trace into a singular point. Switching the order of the equations, or replacing one by its negative will change direction of the tracing.

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

In[54]:=
```
PathFinderC3D[{f_, g_}, p_, q_, s_, x_, y_, z_] := Module[{maxit, k, p0, p1, tv1, tv},
    maxit = 30;
    p0 = p;
    L = Reap[Sow[p];
        k = 0;
        While[Norm[q – p0] > 2 s && k < 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 ≥ maxit, Print["Warning, iteration limit reached"]];
        Sow[q]];
    L[[2, 1]]];
```

For example, tracing from critical point 1 to critical point 4 of Example 1

In[43]:=
```
L1 = PathFinderC3D[{f11, f12}, critpts[[1]], critpts[[4]], .3, x, y, z]
```

Out[43]=
```
{{0., 1.24962, –1.56155}, {0.286675, 1.26629, –1.5213}, {0.55283, 1.30888, –1.40755},
  {0.786664, 1.36178, –1.2356}, {0.984504, 1.41054, –1.02036}, {1.14627, 1.44484, –0.773625},
  {1.27283, 1.45773, –0.504894}, {1.36529, 1.44447, –0.222486}, {1.44667, 1.35882, 0.246463}}
```

A main advantage of this is the accuracy of the points and the ability to use a large step size as long as separate parts of the curve do not come too close together. Of course, if you plan to plot these points then using a large step size will make the plot less smooth. The big disadvantage is that, particularly if the degrees of the defining equations are large, this can be very slow.

A faster alternative is

Parameters are the same for this version.

In[55]:=
```
PathFinderT3D[{f_, g_}, p_, q_, s_, x_, y_, z_] :=
  Module[{maxit, k, p0, p1, tv1, tv, pln},
    maxit = 30;
    p0 = p;
    L = Reap[Sow[p];
      k = 0;
      While[Norm[q - p0] > 2 s && k < maxit,
        tv1 = TangentVector3D[{f, g}, p0, x, y, z];
        If[tv1.(q - p0) > 0, tv = tv1, tv = -tv1];
        p1 = p0 + s * tv1;
        pln = tv1.{x, y, z} - tv1.p1;
        p0 = {x, y, z} /. FindRoot[{f, g, pln}, {x, p1[[1]]}, {y, p1[[2]]}, {z, p1[[3]]}];
        If[Length[p0] == 0, Echo["stuck"]; Break[]];
        Sow[p0];
        k++];
      If[k ≥ maxit, Print["Warning, iteration limit reached"]];
      Sow[q]];
    L[[2, 1]]];
```

We trace the second segment in Example 1

In[65]:=
```
L2 = PathFinderT3D[{f11, f12}, critpts[[4]], critpts[[5]], .2, x, y, z]
```

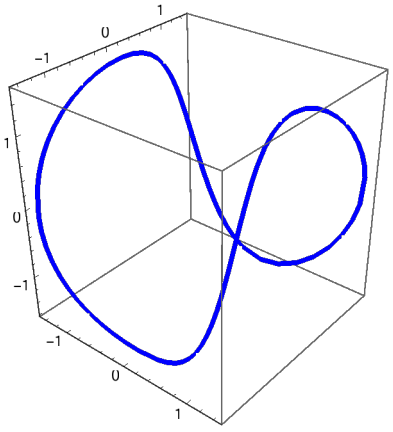
Out[65]=
```
{{1.44667, 1.35882, 0.246463}, {1.45706, 1.29843, 0.437111},
  {1.45516, 1.2227, 0.622493}, {1.4422, 1.13126, 0.800197}, {1.41973, 1.02371, 0.967664},
  {1.38978, 0.899673, 1.12209}, {1.35499, 0.758781, 1.26026}, {1.31882, 0.600834, 1.3783},
  {1.28575, 0.426256, 1.47145}, {1.26107, 0.23705, 1.53411}, {1.24962, 0., 1.56155}}
```

PathFinderT3D is less likely to get all the way from one point to the next and more likely to jump to a different branch or segment of the curve, especially if tracing into a singular point. Working with equations of high degree one can add the option, say, MaxIterations →5 to the FindRoot function to make this even faster, but less accurate. With either of these trying to go too far or in the wrong direction will give an error or possibly incorrect answer.

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

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

Out[93]=



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

Projection Pxy

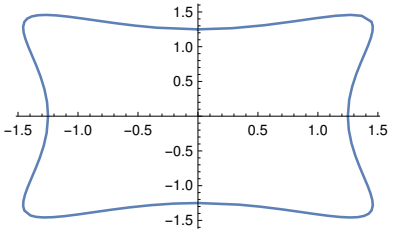
In[9]:=
```
Pxy = {{1, 0, 0}, {0, 1, 0}};
TPxy = {#[[1]], #[[2]]} &;
TPxy[{7, 9, 13}]
```

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

Projection of Example 1

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

Out[68]=



Given a naive space curve we can apply a projection to the equations by finding a sufficient number of random points and interpolating. Although degree is not as an important invariant as with plane curves in the naive

case of a curve $\{f_1, f_2\}$ we can say the *degree* is $\text{tdeg}[f_1] * \text{tdeg}[f_2]$ for the total degree tdeg of the defining equations.

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

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

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

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

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

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

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

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

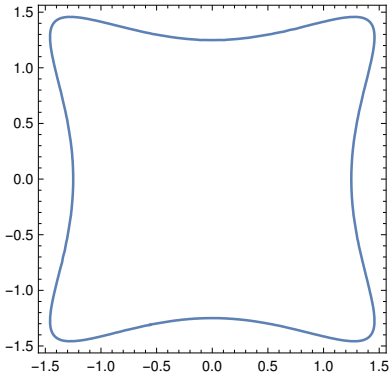
In fact this looks like an exact polynomial

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

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

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

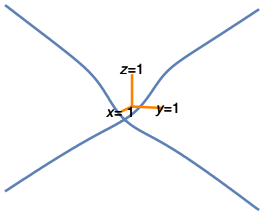
Out[70]=



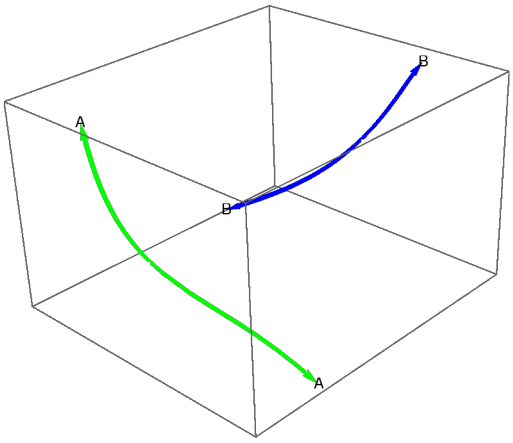
Briefly, one important final example

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

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



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



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

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 components 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 continuous 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.*

The important thing is that, generically, we will get an algebraic curve of the appropriate degree. Thus, topologically many plane curve theorems carry through to space curves such as the Fundamental Theorem in my plane curves book. Some things will not be true, such as Bezout's theorem as distinct space curves need not intersect in any points at all, still we can

conclude that if there are intersections of irreducible space curves the maximum number of such intersections is the product of degrees. We will refine and generalize these ideas in the next section

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.

2.1 Tangent Vectors and Definition of *curve.*

Our first task is to decide if a point on this curve has a tangent vector. To this end we use the function TangentVectorMD[F, p, X] in the software notebook, F being the curve as above, p a point on the curve in $\mathbb{R}^n$ and X being the variable list. If there is a unique tangent vector (up to non-zero scalar multiplication) the function will return a unit vector, otherwise a message. Note that the negative of this unit tangent is also a unit tangent, unfortunately we can't control which we get, and due to some randomization in the function the choice can change in different executions of the function. Mostly, however, we will be mostly interested in whether or not the unique tangent vector exists. The function works by a numerical RREF reduction of the Macaulay matrix at the point. Full details will be in book.

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

```
In[64]:= F2 = {f21, f22, f23} = {x z - y, y z - x^2 - x, z^2 - x - 1}
Out[64]= {-y + x z, -x - x^2 + y z, -1 - x + z^2}
```

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

```
In[55]:= TangentVectorMD[F2, {0, 0, 1}, {x, y, z}]
Out[55]= {0.666667, 0.666667, 0.333333}
```

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

```
In[61]:= TangentVectorMD[F2, {0, 0, RandomReal[{-4, 4}]}, {x, y, z}]
» Not a solution, p = {0, 0, -3.39787}
```

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

```
In[65]:= TangentVectorMD[{f21, f22}, {0, 0, 1}, {x, y, z}]
» No unique tangent vector at {0, 0, 1}
```

```
In[62]:= TangentVectorMD[{f21, f22}, {0, 0, RandomReal[{-4, 4}]}, {x, y, z}]
Out[62]= {-4.93038 × 10^-32, 4.93038 × 10^-32, 1.}
```

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

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

In[70]:=
```
TangentVectorMD[{z (x - y), z (x + y)}, {2, 3, 0}, {x, y, z}]
```

» No unique tangent vector at {2, 3, 0}

We come to our key definition

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

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

An example is the famous Cyclic-4 system consisting of 4 non-linear equations in 4 variables.

In[58]:=
```
C4 = {x + y + z + w, x y + y z + z w + w x, x y z + y z w + z w x + w x y, x y z w - 1};
```

In[51]:=
```
TangentVectorMD[C4, {2, 1/2, -2, -1/2}, {w, x, y, z}]
```

Out[52]= {0.685994, −0.171499, −0.685994, 0.171499}

This has a point with a tangent vector but also a number of points without. We will call this a curve. In fact a segment of the curve can be given parametrically by $\{t, 1/t, -t, -1/t\}$. A similar construction gives a square Cyclic-n for all $n \geq 2$, but only when n is divisible by 4 but not by any other square integer greater than 1 do you get a curve. (see http://homepages.math.uic.edu/~adrovic/jmm13a.pdf).

Note also for the naive "curve" $\{x\,z, y\,z\}$

In[120]:=
```
{a, b, c} = RandomReal[{-3, 3}, 3];
TangentVectorMD[{x z, y z}, {0, 0, c}, {x, y, z}]
```

Out[121]= {0., 0., 1.}

In[122]:=
```
TangentVectorMD[{x z, y z}, {a, b, 0}, {x, y, z}]
```

» No tangent vector at {1.91161, −1.45427, 0}

there are some points with tangent vectors but infinitely many without so this does not meet our definition of a curve.

A point on a curve with a tangent vector is called ***regular****, while a point on a curve with not tangent vector is called* ***singular****.*

The point {1, 1, −1, −1} on Cylic-4 is singular.

In[53]:= **TangentVectorMD [C4, {1, 1, −1, −1}, {w, x, y, z}]**

» No tangent vector at {1, 1, −1, −1}

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

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

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

Degree is 4

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

Degree is 3

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

Degree is 6

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

Degree is ambiguous, try again with higher m

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

Degree is 4

2.2 Fractional Linear Transformations

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

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

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

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

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

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

In[59]:=
```
fltgd[p_, A_] := Module[{U, d},
    U = A.Append[p, 1];
    d = U[[-1]];
    Table[U[[i]] / d, {i, Dimensions[A][[1]] - 1}]]
```

So if

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

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

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

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

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

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

Let

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

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

which is the projection used in Section 1.

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

» Hilbert Function {1, 3, 6, 10, 14}

Out[102]= $\left\{1. - 0.25\,x^2 - 0.25\,x^4 - 0.25\,y^2 + 0.5\,x^2\,y^2 - 0.25\,y^4\right\}$

In[104]:=
```
Expand[-4 g[[1]] ]
```

Out[104]= $-4. + 1.\,x^2 + 1.\,x^4 + 1.\,y^2 - 2.\,x^2\,y^2 + 1.\,y^4$

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

2.3 Projections

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

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

A random projection is one made with random numbers such as

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

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

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

random. An example is our *default pseudorandom projection*

In[60]:=
```
PRD = {{–0.30519764945947847`, 0.9522890290055899`, 0.`},
       {–0.14191095867181538`, –0.045480825358668514`, 0.9888340479238873`}};
```

The associated fractional linear transformation is

In[61]:=
```
FPRD = {{–0.30519764945947847`, 0.9522890290055899`, 0.`, 0.`},
        {–0.14191095867181538`, –0.045480825358668514`,
          0.9888340479238873`, 0.`}, {0.`, 0.`, 0.`, 1.`}};
```

Both of these are assigned global variables.

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

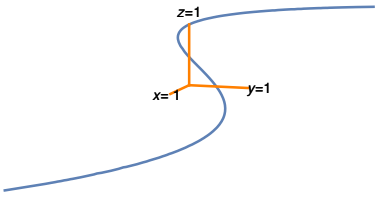
In our notebook of global functions we have a black-box function which plots, using FLTgd, a curve in $\mathbb{R}^3$ using a given projection. In showProjection[F, A , m , X , Y, rng},dtol] *F* is the list of equations of the curve, *A* is the matrix of the projection in FLT form, *m* is the order of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

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

» Hilbert Function {1, 3, 6, 9, 12}

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

Out[60]=

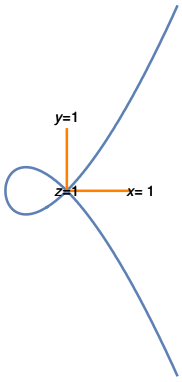


In[61]:=
```
A = {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}};
showProjection[F2, A, 4, {x, y, z}, {x, y}, 3]
```

» Hilbert Function {1, 3, 6, 9, 12}

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

Out[62]=



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

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

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

» Hilbert Function {1, 2, 3, 4}

Out[64]= {1. y}

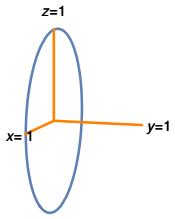
But the point projection is just the interval $-1 \le x \le 1$ of that line. Using our default pseudo-random projection the result

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

» Hilbert Function {1, 3, 5, 7}

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

Out[66]=



is correctly given as a circle.

This example gives one reason why generic projections are preferred over ordinary projections, the probability that the point projection of a curve is

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

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

For A as above

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

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

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

2.4 Fibers and Plotting Space Curves

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

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

In[62]:=
```
Pfiber[P_, p_, t_] := Module[{PC, q, m, n},
  {m, n} = Dimensions[P];
  If[! OrthogonalMatrixQ[P], Echo[P, " not orthogonal"]; Abort[]];
  If[m ≠ n - 1, Echo[P, " not projection matrix"]; Abort[]];
  PC = Orthogonalize[N[Append[P, RandomReal[{-1, 1}, n]]]];
  q = Transpose[PC].Append[p, 0];
  q + t * PC[[n]]]
```

For example

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

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

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

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

In[63]:=
```
Ffiber[F_, P_, p_, X_, tol_] := Module[{Pf, FF, FFs, sol, sol0, sol1, k, n, l, q, u, j, t0},
  n = Dimensions[P][[2]];
  k = Length[F];
  Pf = Pfiber[P, p, t734];
  FF = Chop[Expand[F /. Thread[X → Pf]], tol];
  t0 = RandomReal[{-1, 1}];
  FF = SortBy[FF, (# /. {t734 → t0}) == 0 &];
  If[AllTrue[FF, # == 0 &], Print["inf many sols at", p]; Return[Fail]];
  sol = NSolve[FF[[1]], t734, Reals];
  If[Length[sol] == 0, Echo[p, "no point in fiber at"]; Return[{}]];
  sol0 = t734 /. sol;
  j = 2;
  While[j ≤ k && Length[sol0] > 0 && (FF[[j]] /. {t734 → t0}) ≠ 0,
    sol = NSolve[FF[[j]], t734, Reals];
    If[Length[sol] == 0, Echo[p, "no point in fiber at"];
      sol0 = {}; Break[]];
    sol1 = t734 /. sol;
    sol0 =
      Flatten[Reap[Do[If[Norm[q - u] < tol, Sow[q]], {q, sol0}, {u, sol1}]][[2]]];
    j++];
  sol0 = DeleteDuplicates[sol0, Norm[#1 - #2] < tol &];
  If[Length[sol0] == 0, Echo[p, "no point in fiber at "]];
  If[Length[sol0] > 1, Echo[p, "multiple fiber points"]];
  Pf /. {t734 → #} & /@ sol0
 ]
```

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

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

Out[75]=
```
{{1., 1.41421, 1.41421}}
```

In[76]:=
```
Ffiber[F2, Pxy, {0, 1}, {x, y, z}, dtol]
```

» no point in fiber at {0, 1}

Out[76]=
```
{}
```

In[77]:= **Ffiber[F2, Pxy, {0, 0}, {x, y, z}, dtol]**

» multiple fiber points {0, 0}

Out[77]= {{0., 0., −1.}, {0., 0., 1.}}

In[78]:= **Flatten[Ffiber[F2, Pxy, #, {x, y, z}, dtol] &/@{{1, Sqrt[2]}, {1, −Sqrt[2]}}, 1]**

Out[78]= {{1., 1.41421, 1.41421}, {1., −1.41421, −1.41421}}

2.4.1 Application to Example 2

Continuing with example 2 now using the default generic projecction

In[84]:= **f = FLTgd [F2, FPRD, 4, {x, y, z}, {x, y}, dtol][[1]]**

» Hilbert Function {1, 3, 6, 9, 12}

Out[84]= $1. - 2.37355\,x + 0.0574214\,x^2 + 0.000243617\,x^3 - 2.18663\,y + 0.955595\,x\,y + 0.0153028\,x^2\,y - 1.02271\,y^2 + 0.320413\,x\,y^2 + 2.2363\,y^3$

In[86]:= **cp = DefCritPts2D[f, x, y]**
bp = {x, y} /. NSolve[{f, x^2+y^2−16}, {x, y}, Reals]

Out[86]= $\{\{-2.2102\times10^{13}, 1.05558\times10^{12}\}, \{-2.21017\times10^{13}, 1.05557\times10^{12}\}, \{-210.502, 15.985\}, \{-210.499, 15.9848\}, \{0.228844, 0.222392\}\}$

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

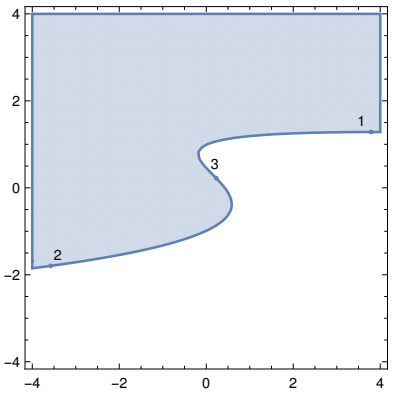
This gives a list of 3 non-singular points for path tracing

In[88]:= **ap = Append[bp, cp[[5]]]**

Out[88]= {{3.78876, 1.2827}, {−3.57476, −1.79474}, {0.228844, 0.222392}}

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

Out[91]=



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

Warning: iteration limit reached

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

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

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

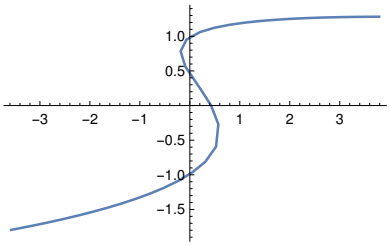
Out[139]= {{0.792628, 1.16608}, {0.495874, 1.12389}, {0.20353, 1.06116}, {−0.0668034, 0.952203},
{−0.184198, 0.78409}, {−0.091331, 0.565073}, {0.228844, 0.222392}}

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

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

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

Out[143]=



Now we can lift the plane curve to space

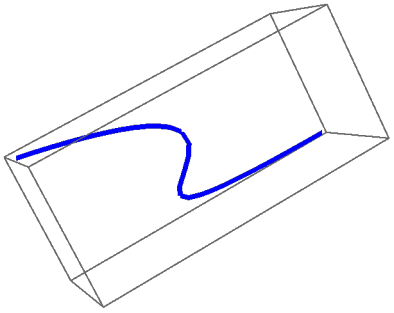
In[145]:= **{tm, SP} = Timing[Flatten[Ffiber[F2, PRD, #, {x, y, z}, dtol] &/@L, 1]]**

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

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

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

Out[146]=



2.4.2 Application to Cyclic 4

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

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

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

Trial and error require that $m \geq 6$

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

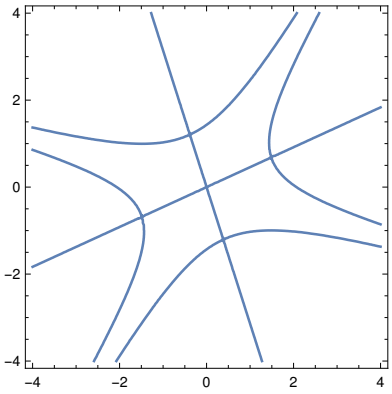
» Hilbert Function {1, 3, 6, 10, 15, 21, 27}

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

We plot h

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

Out[78]=



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

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

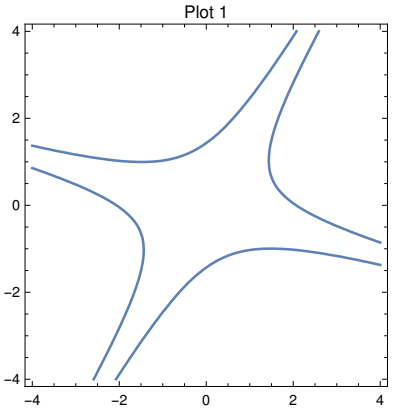
Using numerical division Ndivide from the plane curve appendix we get a degree 4 component which we then plot.

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

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

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

Out[126]=



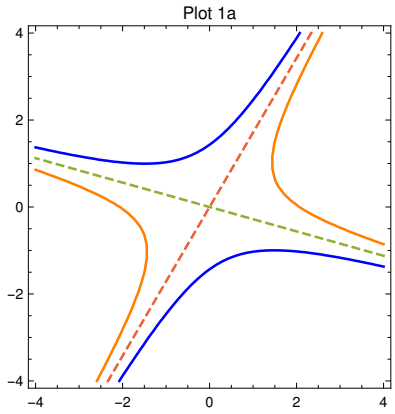
Again using our Plane Curve theory we analyze the two infinite points and find they have intersection multiplicity 2, from the Bezout bound on singularities we conclude that h2 is not irreducible. By tracing and interpolation we find two components

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

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

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

Out[149]=



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

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

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

For example, perhaps run several times,

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

Out[113]= {0.384516, −1.20751}

Out[115]= {1., 1., −1., −1.}

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

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

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

```
In[123]:= L4 = Flatten[Ffiber[C4, P43, #, {w, x, y, z}, 1.*^-5] &/@ L3, 1]
Out[123]= {{1.30801, 0.764519, -1.30801, -0.764519}, {1., 1., -1., -1.},
           {0.942706, 1.06078, -0.942706, -1.06078}, {0.942706, 1.06078, -0.942706, -1.06078}}
```

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

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

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

```
In[128]:= C4b = HBasis[Join[C4, plane2], 6, {w, x, y, z}, 1.*^-10]
» Hilbert Function {1, 2, 3, 4, 4, 4, 4}
Out[128]= {1. w + 1. y, 1. x + 1. z, -1. + 1. w^2 x^2}
```

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

```
In[129]:= h3 = FLTgd [C4b, FA1, 4, {w, x, y, z}, {x, y}, 1.*^-9]
» Hilbert Function {1, 3, 6, 10, 14}
Out[129]= {1. - 0.0547971 x^4 - 0.326729 x^3 y - 0.258872 x^2 y^2 + 0.6802 x y^3 - 0.237496 y^4}
```

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

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

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

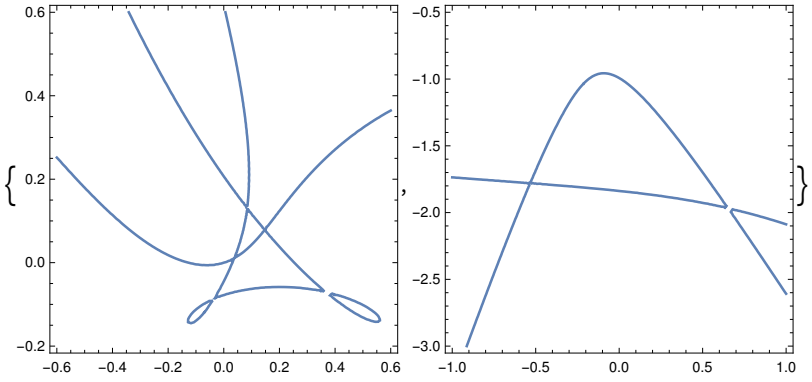
This curve was originally randomly generated as the intersection of 3 quadratic hypersurfaces of 4 space.

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

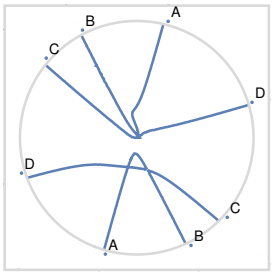
We first project with Pxyz, the simple projection setting $w = 0$. This gives a system of 7 equations of degree 5 in the three variables x, y, z. We will not reproduce this. We next project by our default pseudo-random projection PRD. We get a numerical plane curve of degree 8 with coefficients ranging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

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

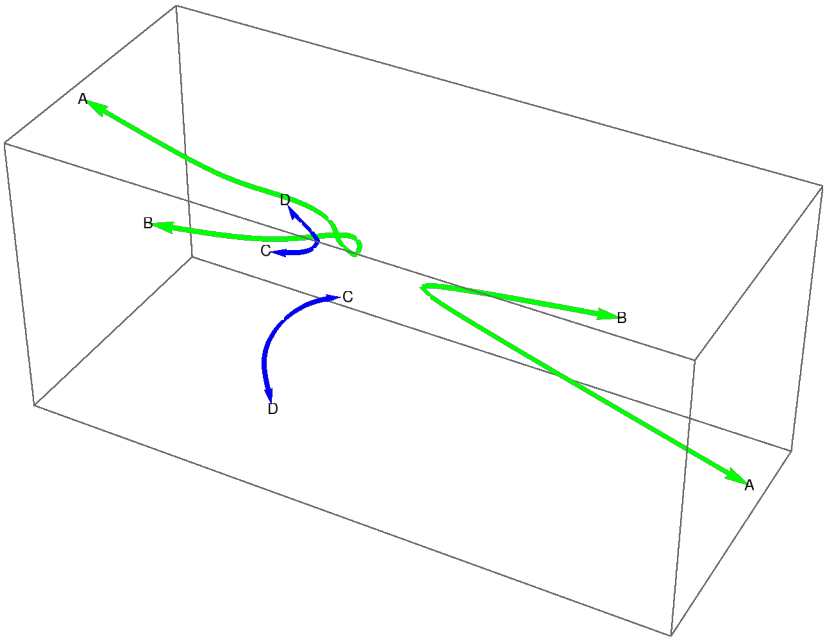
Out[154]=



Note the approximate position of the infinite points are



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



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

2.5 Numerical Irreducible Decomposition.

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

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

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

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

As an example consider the curve given by $\{y^2 - x^3 - x^2,\ x\,z - y\}$. One can easily see that any point {0, 0, z} satisfies this equation, in fact the entire line with these points consists of double points. On can look for the complementary component. Projecting this curve to the plane with the standard $z \to 0$ projection gives the nodal curve $y^2 = x^3 + x^2$. By interpolating the lifts of the non-singular points of this curve to $\mathbb{R}^3$ we find the system defining the non-singular component

```
In[3]:= F2
Out[3]= {-y + x z, -x - x^2 + y z, -1 - x + z^2}
```

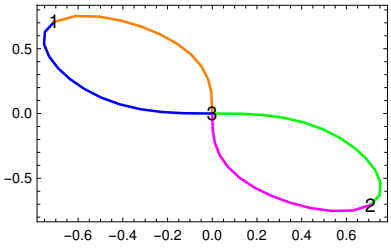
which we discussed earlier.

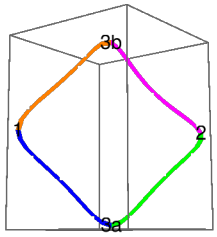
2.6 Fundamental Theorem

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

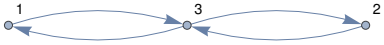
```
E6a = E6a = {x + y - x z + y z, -x - 2 x^2 y - 2 x y^2 - 2 y^3 + x z + 2 x^3 z,
      -1 + 6 x^2 + 8 x y + 4 y^2 - 4 x^2 z + z^2 + 2 x^2 z^2, x^4 + x y + y^4};
```

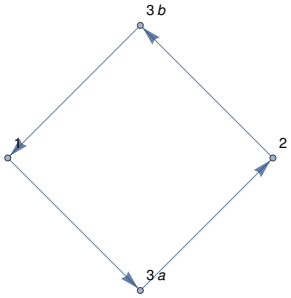
Projecting with the non - generic projection $z \to 0$ gives the last equation $x^4 + x\,y + y^4 = 0$. Plotting this with path tracing and lifting gives





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





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

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

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

3. Later Chapters

The last few chapters 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 chapter will cover Quadratic Surface Intersection Curves. This is a classical area that only recently has seen a full solution. L. Dupont, D. Lazard, S. Lazard and S. Petitjean presented a 65 page discussion and working black box algorithm in 2008. (See http://vegas.loria.fr/qi/index.html) . In 2013 I gave a simple non-black box method in the case of a non-planar component by reducing to a plane curbic. Here I give a complete discus-

sion using both my previous method an the method used in explaining example F1 above.

Another chapter will cover curves defined by a polynomial or rational parameterization in one variable. Standard methods of finding algebraic equations often give additional components, by my dual interpolation method I can describe a system, usually with more equations, describing the algebraic component containing the parameterized curve.

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

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