A Wolfram Language Approach to Real Numerical Algebraic Plane Curves

Barry H. Dayton

This article is a summary of my book *A Numerical Approach to Real Algebraic Curves with the Wolfram Language* [1].

1. Introduction

The nineteenth century saw great progress in geometric (real) and analytic (complex) algebraic plane curves. In the absence of an ability to do the large number of computations for a concrete theory, the twentieth century saw the abstraction to algebraic geometry of this material. Ideas of ideals, rings, fields, varieties, divisors, characters, sheaves, schemes and many types of homology and cohomology arose. The added benefit of this approach is that it became possible to apply geometric techniques to other fields. Probably the most striking accomplishment of this abstract approach was the solution of Fermat's problem by Wiles and Taylor at the end of the century.

The plane geometric curve theory of the nineteenth century was collateral damage. All modern books on the subject want to follow the abstract approach, which raises the bar for those who want to know this theory. In addition, little attention was given to the concrete geometric theory. One goal of my book is to rectify this problem; substituting software for the abstract theory, we can give the theory in terms the non-mathematician can follow.

Since most algebraic curves have only finitely many rational points, I work numerically. The methods are constructive, heuristic and visual rather than the traditional theorem-proof of contemporary mathematics. In fact there is a fundamental oxymoron at the heart of my approach: a numerical algebraic curve is the solution set of an equation $f(x, y) = 0$, where $f(x, y)$ is a polynomial with integer or machine-number coefficients. Evaluating this polynomial at a point (x, y) with machine-number coordinates gives a machine number on the left-hand side, while the right-hand side is a symbolic number, so actual equality is impossible. So my book is not an algebraic geometry book. Having worked during my career as a mathematician in both the abstract and numerical realms, I believe that while these approaches are incompatible, they can and should coexist within mathematics.

2. Lines

We will generally describe an algebraic plane curve by giving a polynomial in two variables with integer or real machine-number coefficients.

From an operational point of view, with an exception noted later, for a given curve $f(x, y)$ we accept the output of `NSolve[{f, g}, {x, y}, ℝ]` and `FindRoot[{f, g}, {x, x0}, {y, y0}]`, where g is another such bivariate polynomial and x_0, y_0 are real machine numbers, as points on the curve $f(x, y) = 0$. In numerical work, we do not accept points $\{x_0, y_0\}$ as solutions to $f(x, y) = 0$ based on the value of $f(x_0, y_0)$, known as the *residue*. A detailed explanation is in [1].

For example, suppose some calculation claims p is a point on f. (If you have set values in your session for `x`, `y` and so on, now is the time to store them if needed and apply `Clear` to them.)

```
f = x^5 - 2 x y^4 + 3 x^2 y - 5 y^2;
p = {-3.0602031048641716`, -2.8206303312619827`};
```

We find a random line g containing p and use `FindRoot` to check the point of intersection of g with f.

```
g = Expand[RandomReal[{-4, 4}, 2].({x, y} - p)]
```

```
15.515 + 1.51103 x + 3.86117 y
```

We see the residue is not zero.

```
f /. Thread[{x, y} → p]
```

```
-5.68434 × 10^-14
```

But p can be reconstructed from `FindRoot`.

```
q = {x, y} /. FindRoot[{f, g}, {x, p[[1]]}, {y, p[[2]]}]
```

```
{-3.0602, -2.82063}
```

It checks.

```
p == q
```

```
True
```

The simplest example of an algebraic plane curve is a line. The first problem for lines is to find the equation of a line through two given points. We give our solution, found at the beginning of Chapter 1 of [1], as it will give the flavor of our approach to this subject.

Let $\{x_1, y_1\}, \{x_2, y_2\}$ be the given points. The desired equation is of the form

$$a\,x + b\,y + c = 0.$$

We thus consider the coefficients a, b, c as unknowns, but x, y as coordinates of the given points. So we have two equations in the three variables a, b, c.

$$a\,x_1 + b\,y_1 + c = 0,$$
$$a\,x_2 + b\,y_2 + c = 0.$$

But this system is underdetermined. It is also not symmetric in the variables, so we use a dummy variable z and add a third equation to get the system

$$\begin{aligned} &a\,x_1 + b\,y_1 + c\,z = 0, \\ &a\,x_2 + b\,y_2 + c\,z = 0, \\ &r_1\,a + r_2\,b + r_3\,c = 1, \end{aligned} \tag{1}$$

where r_1, r_2, r_3 are random real numbers.

Suppose the points are $\{3, 2\}$ and $\{5, 6\}$. Here are the random reals.

```
{r1, r2, r3} = RandomReal[{-1, 1}, 3]
```

```
{0.0780892, 0.690991, -0.880503}
```

Then the line is constructed as follows.

```
ℓ = (a x + b y + c) /.
  Solve[3 a + 2 b + 1 c == 0 && 5 a + 6 b + 1 c == 0 && r1 a + r2 b + r3 c == 1,
    {a, b, c}][[1]]
```

```
-1.33905 + 0.669523 x - 0.334762 y
```

Perhaps this is not what you expected. But we are working with machine numbers so, particularly if this is not our final answer, we should not mind. If this does still bother us, we can always look for integers.

```
Expand[ℓ / Coefficient[ℓ, y]]
```

```
4. - 2. x + 1. y
```

But system (1) gives more options. Suppose instead we were given the point $\{3, 2\}$ and slope 2. We can change the second equation by setting $x_2 = 1, y_2 = 2$ and $z = 0$.

```
(a x + b y + c) /.
 Solve[3 a + 2 b + 1 c == 0 && 1 a + 2 b + 0 c == 0 && r1 a + r2 b + r3 c == 1,
   {a, b, c}][[1]]
```

```
-1.33905 + 0.669523 x - 0.334762 y
```

```
Expand[% / Coefficient[%, y]]
```

```
4. - 2. x + 1. y
```

Since our original l already had slope 2, we are now not surprised to get the same result. Now consider the possibility that our line was given parametrically.

```
lp = {3, 2} + # {2, 4} &;
lp[t]
```

```
{3 + 2 t, 2 + 4 t}
```

This time we replace the second equation in (1) with $x_2 = 2$, $y_2 = 4$, and again $z = 0$. We solve the new system.

```
(a x + b y + c) /.
 Solve[3 a + 2 b + 1 c == 0 && 2 a + 4 b + 0 c == 0 && r1 a + r2 b + r3 c == 1,
   {a, b, c}][[1]]
```

```
-1.33905 + 0.669523 x - 0.334762 y
```

```
Expand[% / Coefficient[%, y]]
```

```
4. - 2. x + 1. y
```

This is the same answer, because again we have the line through {2, 3} and {5, 6}.

```
lp[1]
```

```
{5, 6}
```

We can put all of this into one program if we simply make the convention that a slope or direction vector is denoted by a triple with third coordinate 0. So here is our universal code for creating a line.

```
line[p_, q_, x_, y_] := Module[
  {a, b, c, M},
  M = {
    Switch[Length[p], 2, Append[p, 1], 3, p],
    Switch[Length[q], 2, Append[q, 1], 3, q],
    RandomReal[{-1, 1}, 3]
   };
  Normalize[{a, b, c} /. First@Solve[M.{a, b, c} == {0, 0, 1}]].
   {x, y, 1}
]
```

Our results will differ from the previous ones because we are now choosing the random numbers each run but normalizing the output. The advantage is that each run gives the same answer up to a factor of ± 1.

```
l2 = line[{3, 2}, {5, 6}, x, y]
```

```
0.872872 - 0.436436 x + 0.218218 y
```

```
line[{3, 2}, {1, 2, 0}, x, y]
```

```
- 0.872872 + 0.436436 x - 0.218218 y
```

```
line[{3, 2}, {2, 4, 0}, x, y]
```

```
- 0.872872 + 0.436436 x - 0.218218 y
```

```
Expand[l2 / Coefficient[l2, y]]
```

```
4. - 2. x + 1. y
```

Computing a point far away from {3, 2} by taking $t = 3047$ in our parametric equation, we get approximately $\{2\,t, 4\,t\}$.

```
p2 = lp[3047]
```

```
{6097, 12 190}
```

```
{2, 4} 3047
```

```
{6094, 12 188}
```

So we can consider {2, 4, 0} to be the *infinite point* on the line. But putting {1, 2, 0} in our `line` function gave us the same thing, so these infinite points are *homogeneous*; that is, they can be multiplied by a scalar getting the same infinite point. Note also that adding a coordinate 1 to a coordinate pair *homogenizes* a Cartesian point of the plane.

```
line[{3, 2, 1}, {2, 4, 0}, x, y]
```

```
- 0.872872 + 0.436436 x - 0.218218 y
```

```
line[{-6, -4, -2}, {2, 4, 0}, x, y]
```

```
- 0.872872 + 0.436436 x - 0.218218 y
```

In Chapter 5 of [1], we find that we have invented the *projective plane*. So that we do not get confused, we will henceforth call points (pairs) of our standard Cartesian plane *affine points* and the triples *projective points*.

The method for finding equations of lines can be generalized to find curves of degree d through $\frac{(d+2)(d+1)}{2} - 1$ sufficiently general points. See [2] for the code of `aCurve` (here `a` stands for affine).

We do define two families of curves that are used extensively as examples in [1]. The first are *Gaussian curves*. We start with a single variable polynomial $p(z)$, typically with integer coefficients but possibly complex integers such as $3 + 4\,i$. Replace z by $x + i\,y$; after expanding, the formal real part forms a curve. Gauss used this construction in his first and fourth proofs of the fundamental theorem of algebra, published 50 years apart.

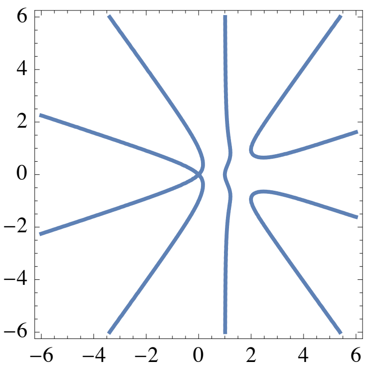
```
gaussCurve[p_, x_, y_, z_] :=
 ComplexExpand[Re[p /. {z → x + I y}]]
```

For example, the following is said to be Gauss's original example for the fourth proof. Note that it has a singular point!

```
f1 = gaussCurve[z^5 - 5 z^4 + 9 z^3 - 5 z^2, x, y, z]
```

$$-5\,x^2 + 9\,x^3 - 5\,x^4 + x^5 + 5\,y^2 - 27\,x\,y^2 + 30\,x^2\,y^2 - 10\,x^3\,y^2 - 5\,y^4 + 5\,x\,y^4$$

```
ContourPlot[f1 == 0, {x, -6, 6}, {y, -6, 6}, ImageSize → Small]
```



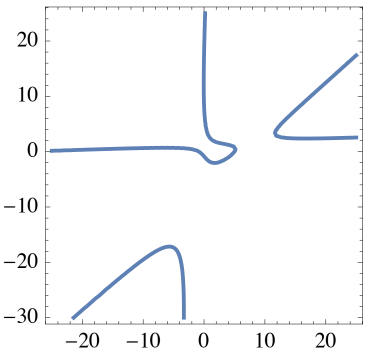
A second family of curves I call Newton's hyperbolas. Here n can range from 1 to 2^{28}.

```
newtonHyperbola[n_, x_, y_] := Module[{v, c, k, n2dig, s},
  If[(n > 2^28) || (n < 0), Print["n out of range"];
   Return[Fail]];
  v = {{0, 0}, {1, 0}, {0, 1}, {2, 0}, {1, 1}, {0, 2},
    {3, 0}, {2, 1}, {1, 2}, {0, 3}, {4, 0}, {3, 1},
    {2, 2}, {1, 3}, {0, 4}, {5, 0}, {4, 1}, {3, 2},
    {2, 3}, {1, 4}, {0, 5}, {6, 0}, {5, 1}, {4, 2},
    {3, 3}, {2, 4}, {1, 5}, {0, 6}};
c = {1, 1, 1, .2, 1, .2, .008, .2, .2, .008, .000064,
      .008, .2, .008, .000064, .2^10, .2^6, .008, .008,
      .2^6, .2^10, .2^15, .2^10, .2^6, .2^3, .2^6,
      .2^10, .2^15};
    k = Which[n < 8, 3, n < 64, 6, n < 1024 , 10, n < 32768,
      15, n < 2^21, 21, True, 28];
    n2dig = IntegerDigits[n, 2, k];
    s = Reverse[n2dig];
    FromCoefficientRules[
     Table[v[[i]] → (-1)^s[[i]] c[[i]], {i, k}], {x, y}]]
```

```
nh376 = newtonHyperbola[376, x, y]
```

$1 + x - 0.2\,x^2 - 0.008\,x^3 + y - x\,y + 0.2\,x^2\,y - 0.2\,y^2 - 0.2\,x\,y^2 + 0.008\,y^3$

```
ContourPlot[nh376 == 0, {x, -25, 25}, {y, -30, 25},
 ImageSize → Small]
```



3. Important Definitions

The *total degree* of a plane curve is an important invariant, but not quite as simple in the numerical case as it may seem. Small coefficients of the highest degrees matter little near the origin but strongly affect the asymptotic and infinite behavior of the curve. Therefore, we approach this symbolically using `CoefficientRules`.

```
tDeg[f_, x_, y_] :=
 Max[Table[Total[t], {t, Keys[CoefficientRules[f, {x, y}]]}]]
```

Sometimes a little care is necessary to make sure that coefficients that are the result of round-off error only are not allowed to increase the degree; a judicious use of `Chop` may be required.

Because we are often working numerically, we use a slightly stronger criterion for a plane curve $f(x, y)$ to be called regular at a point p on f. The quantity $b\left(\frac{\partial}{\partial x} f\right) - a\left(\frac{\partial}{\partial y} f\right)$ is known as the *Jacobian determinant* of the intersection of the curve f and line $a\,x + b\,y + c$ at p.

We say *p is a regular point of* $f = 0$ if the Jacobian is not numerically zero for almost all pairs a, b of machine numbers. In practice, this can be checked by letting a, b be random real numbers. For a regular point p, a *tangent line* is defined as follows.

```
tLine[f_, p_, x, y] :=
 line[p, {D[f, y], -D[f, x], 0} /. Thread[{x, y} → p], x, y]
```

On the other hand, a point p of $f(x, y)$ is called *singular* if the Jacobian is zero at p for all numbers a, b. Again, in practice it is enough to check for a random pair a, b.

An alert reader may notice that since we are working constructively, *regular* and *singular* are not logical negations of each other, but a practical test does distinguish regular from singular points.

An important kind of point for [1] is a *critical point*. A point p on curve f is critical if it is also on the curve defined by $y\,D[f, x] - x\,D[f, y]$. All real critical points of a curve can be found easily in practice by the following.

```
criticalPoints[f_, x_, y_] :=
 DeleteDuplicates[
  {x, y} /. NSolve[{f, y D[f, x] - x D[f, y]}, {x, y}, Reals],
  Norm[#1 - #2] < 1.*^-6 &]
```

Unlike the conditions regular and singular, which are invariant under transformations such as translation, being a critical point is a positional property. Among the critical points are local extrema of the distance from the origin to a point on the curve and, by our definition, singular points. The most important thing about critical points is that *every affine topological component of a plane curve contains at least one critical point*. This means that from our simple function for finding critical points, we will be able to locate all components on the curve, no matter how small—even one-point components.

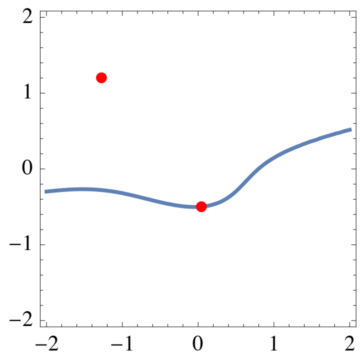
Consider the following contrived example of a numerical cubic curve, which has an isolated point.

```
h1 = -0.3625600420883557` + 0.0805181117798253` x +
   0.31666929459884297` x^2 + 0.19484528794280442` x^3 -
   0.27630436149898124` y - 0.5134344028165023` x y -
   0.6511049000079859` x^2 y + 0.45295488379091586` y^2 -
   1.109368568425435` x y^2 - 0.8788321984542753` y^3;

cp1 = criticalPoints[h1, x, y]

{{-1.27562, 1.20231}, {0.042992, -0.498191}}
```

```
ContourPlot[h1 == 0, {x, -2, 2}, {y, -2, 2},
 Epilog → {Red, PointSize[Medium], Point[cp1]}]
```



The point with coordinates $\{-1.27562,\ 1.20231\}$ is a one-point component of the curve `h1`.

The same idea allows us to find the closest point on a curve to a given point in the plane.

```
closestPoint[f_, q_, x_, y_] := Module[{g, p, cp},
  g = Expand[f /. Thread[{x, y} → Take[q, 2] + {x, y}]];
  cp = criticalPoints[g, x, y];
  q + First@MinimalBy[cp, Norm]
 ]
```

In this case, the closest point may be one invisible on a plot.

```
closestPoint[h1, {-1, 1}, x, y]
```

```
{-1.27562, 1.20231}
```

We may also find the infinite points of a curve. Here is code that is slightly different from [1] but avoids subroutines. This uses a random variable so that different runs give the infinite points in possibly a different order.

```
infiniteRealPoints[f_, x_, y_] :=
 Module[{d, ca, K, fm, k, rnl, sol, sol1},
  d = tDeg[f, x, y];
  ca = <|CoefficientRules[f, {x, y}]|>;
  K = Select[Keys[ca], Total[#] == d &];
  fm = FromCoefficientRules[Table[k → ca[k], {k, K}],
    {x, y}];
  rnl = RandomReal[{-1, 1}, 2].{x, y} - 1;
  sol = NSolve[{fm, rnl}, {x, y}, Reals];
  If[Length[sol] == 0, Return[{}], sol = {x, y} /. sol];
  sol1 = DeleteDuplicates[sol, Norm[#1 - #2] < .001 &];
  Normalize /@ Table[Append[sol1[[i]], 0], {i, Length[sol1]}]
 ]
```

Here is an example using Newton hyperbola 376.

```
infiniteRealPoints[nh376, x, y]
```

```
{{-0.99913, -0.0417029, 0.},
 {-0.707107, -0.707107, 0.}, {-0.0417029, -0.99913, 0.}}
```

4. Topology and Tracing

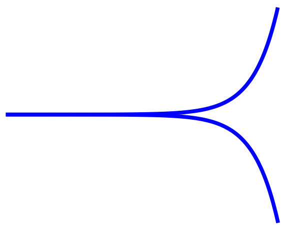
We start with an idea Gauss used in his 1849 proof of the fundamental theorem of algebra characterizing real plane curves. Given a bivariate polynomial $f(x, y)$, Gauss considered the semialgebraic set $f^+ = \{\{x, y\} \mid f(x, y) > 0\}$. *The algebraic curve* $f(x, y) = 0$ *is the complete topological boundary in* $\mathbb{R}^2$ *of* f^+.

Among other things, this nicely solves our conundrum as to the precise meaning of the curve $f(x, y) = 0$ when $f(x, y)$ is a polynomial with machine-number coefficients, as the inequality $f(x, y) > 0$ does make sense numerically.

Another consequence of this definition is that for each regular point p of the curve, a line different from the tangent line intersecting the curve at this point travels from f^+ to the negative set f^- at p. We will see later in this section that a curve defined by a square-free f has only finitely many singular points, so a contour plot gives a reasonable picture of the curve in a bounded region with appropriate scaling. Contour plots may miss large parts or all of the curve if the polynomial $f(x, y)$ has a factor repeated an even number of times. Fortunately, if f is a polynomial with integer coefficients, then the built-in function `Factor` finds the repeated factors, and one can produce a square-free polynomial with the same curve. For machine-coefficient polynomials, there is a function given in Appendix 1 of [1] and in [2] that can check to see if f is square free and if not, produce a square-free polynomial giving the same curve.

This last paragraph also tells us that the complement of a square-free curve is two colored, with the curve separating the colors. In particular, an algebraic real plane curve cannot have bifurcations [3]. That is, the following cannot be a plot of an algebraic curve.

```
Plot[10^-11 {Exp[5 x], -Exp[5 x]}, {x, 2, 5}, Axes → False,
 PlotStyle → Blue]
```



There are always an even number of *branches* going in and out of singular points, an essential idea we will use in the next section.

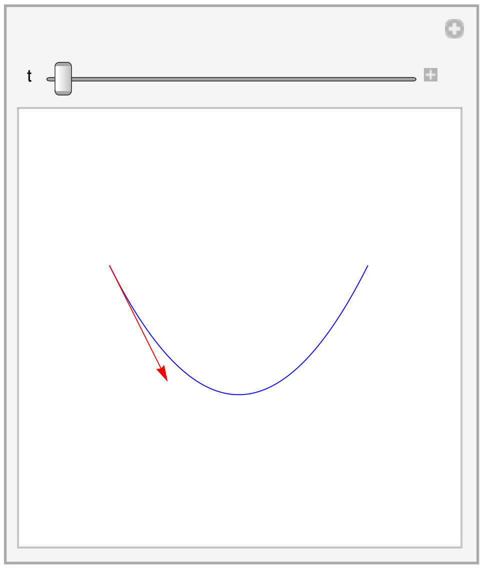
For now, the main use of the Gauss point of view is that a square-free curve is oriented; that is, we can specify a direction of travel along the curve. In his proof, Gauss proposed “walking along the curve” with f^+ on our right. Essentially, we are traveling around topological components clockwise. As an aside, the curve Gauss was using is our Gaussian curve of the particular complex univariate polynomial $p(z)$ that he was proving has a zero. Thinking of points of the plane as complex numbers, Gauss showed the walker would always stumble over a zero of $p(z)$.

We implement this by noting that for regular points, this right-hand direction is given by the vector $(D_x f, D_y f)$, so we can use the following code (`g` stands for Gauss, `T` for tangent, and `vec` for vector).

```
gTvec[f_, p_, x_, y_] :=
 Cross[
  Normalize[Chop[{D[f, x], D[f, y]} /. Thread[{x, y} → p],
    1.*^-6]]]
```

Example: Consider the curve $x^2 - y$. (In the PDF and HTML versions, the graphic is not interactive.)

```
Manipulate[
 Graphics[{
   {Blue, Line[Table[{t, t^2}, {t, -1, 1, .01}]]},
   {Red,
    Arrow[
     {{t, t^2}, {t, t^2} + gTvec[x^2 - y, {t, t^2}, x, y]}]}
  }, PlotRange → {{-1.5, 1.5}, {-1, 2}}, ImageSize → Small],
 {t, -1, 1},
 SaveDefinitions → True
]
```



This leads to path tracing. In [1], we consider various methods, including using a method based on the built-in `NDSolveValue`. Here, we use a very common method given by the following.

```
Options[pathFinderT] = {maxiterations → 40};

pathFinderT[f_, p_, q_, s_, x_, y_, OptionsPattern[]] :=
  Module[
   {maxit, p0, L, k, tv, u, l, p1},
   maxit = OptionValue[maxiterations];
   p0 = p;
   L = Reap[
      Sow[p];
      k = 0;
      While[Norm[q - p0] > 2 s && k < maxit,
       tv = gTvec[f, p0, x, y];
       u = p0 + s tv;
       l = line[u, {tv[[2]], -tv[[1]], 0}, x, y];
       p1 = {x, y} /. FindRoot[{f, l}, {x, u[[1]]},
          {y, u[[2]]}];
       Sow[p1];
       p0 = p1;
       k++];
      Sow[q]
     ][[2, 1]];
   If[k ≥ maxit,
    Print["Warning: iteration limit reached at ", p1]];
   L];
```

This function traces from point p to point q in the direction defined by `gTvec` with steps of size s. By default, it stops after 40 steps, but that can be changed by an option. If q is the wrong direction from p, this fails with a warning. The direction can be changed by replacing the curve f by $-f$. If there is a singular point in the path between p and q, then this will likely get hung up there. The key is that one can trace into a singularity, but not out. Normally, we use critical points for the endpoints p, q, but we may need to add points between singularities.

The bow curve is a good example of using path tracing.

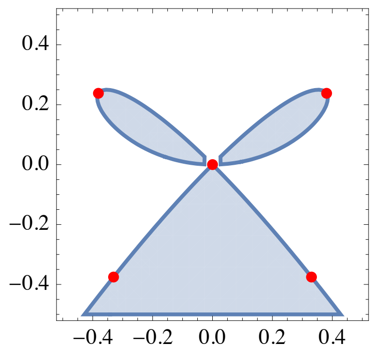
```
f3 = x^2 y - x^4 - y^3;
gp1 = Join[
  {x, y} /. NSolve[{f3, x^2 + y^2 - .25}, {x, y}, Reals],
  Chop[criticalPoints[f3, x, y]]
 ]

{{-0.330354, -0.375322}, {0.330354, -0.375322},
 {0.380892, 0.237985}, {-0.380892, 0.237985}, {0, 0}}
```

```
pta = Association[Table[i → gp1[[i]], {i, Length[gp1]}]]
```

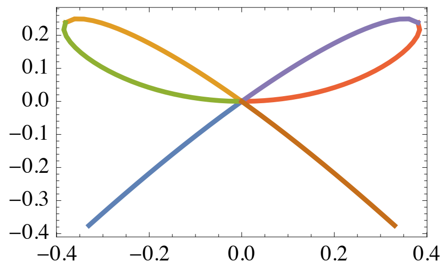
```
<| 1 → {-0.330354, -0.375322},
 2 → {0.330354, -0.375322}, 3 → {0.380892, 0.237985},
 4 → {-0.380892, 0.237985}, 5 → {0, 0} |>
```

```
RegionPlot[f3 > 0, {x, -.5, .5}, {y, -.5, .5},
 ImageSize → Small,
 Epilog → {Red, PointSize[Medium], Point[gp1]}]
```



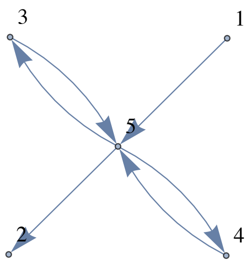
We proceed as follows with the positive direction clockwise around the positive region, but always into the singularity at {0, 0}.

```
P1 = pathFinderT[f3, pta[1], pta[5], .04, x, y];
P2 = pathFinderT[f3, pta[4], pta[5], .02, x, y,
    maxiterations → 100];
P3 = pathFinderT[-f3, pta[4], pta[5], .02, x, y,
    maxiterations → 100];
P4 = pathFinderT[f3, pta[3], pta[5], .02, x, y,
    maxiterations → 100];
P5 = pathFinderT[-f3, pta[3], pta[5], .02, x, y,
    maxiterations → 100];
P6 = pathFinderT[-f3, pta[2], pta[5], .04, x, y];
ListLinePlot[{P1, P2, P3, P4, P5, P6}, PlotStyle → Thick,
 Frame → True, Axes → False, ImageSize → Small]
```



In [1], we develop a number of utility functions to make tracing easier and do many examples, particularly of Gaussian curves. But the main point we are making is that a square-free curve can be reasonably approximated by a piecewise linear curve, and the instructions to do so can be given by a graph (network) consisting of the endpoints of each trace as vertices with the direction traveled, not traced, as directed edges. Here is the graph for the previous example.

```
Graph[{1 → 5, 5 → 4, 4 → 5, 5 → 3, 3 → 5, 5 → 2},
 VertexLabels → "Name"]
```



5. A Classical Interlude

In this section, we touch base with contemporary algebraic geometry. We operate in the *real and complex projective planes* $\mathbb{P}^2_{\mathbb{R}}$ and $\mathbb{P}^2_{\mathbb{C}}$.

Our construction follows our discussion on lines in Section 2. A point in the real (or complex) projective plane is a triple $\{a, b, c\}$ of real (or complex) numbers so that not all of a, b, c are zero. Two such triples that differ by a nonzero real (or complex) multiple are considered the same. For example, if $c \neq 0$, then loosely speaking, $\{a, b, c\} = \left(\frac{a}{c}, \frac{b}{c}, 1\right) = \left(\frac{a}{c}, \frac{b}{c}\right)$ is an affine point. We called points $(a, b, 0)$ infinite points; in the projective plane they are just points. Just as we added a variable z for the third coefficient in the equation of a line, in the projective plane we again add a third variable for equations. We call this *homogenization*. Now we want all of our monomials to have the same total degree. The next function homogenizes a bivariate polynomial.

```
homog[f_, x_, y_, z_] :=
 Expand[z^tDeg[f, x, y] (f /. Thread[{x → x / z, y → y / z}])]
```

That is, if we are working with a polynomial $f(x, y)$ of degree d, a monomial is converted by $x^i y^j \mapsto x^i y^j z^{d-i-j}$. There is a 1-1 correspondence between two-variable monomials of total degree less than or equal to d and three-variable monomials of degree exactly d.

If particular a, b, c with $h(a, b, c) = 0$, then $h(r\,a, r\,b, r\,c) = 0$ also, so being a zero is a property of the projective point $p = (a, b, c)$. Thus *projective curves* are the zero set of homogeneous polynomials in three variables. Also in the example for the bow curve `f3`, $(1, 0, 0)$ is a point of the homogenization of $f(x, y)$, which means $(1, 0, 0)$ is an infinite point of f.

The opposite of homogenization is *specialization*. We can substitute the number 1 for any of the three variables in a homogeneous polynomial and get a two-variable polynomial that is in general nonhomogeneous. For example, if we homogenize $f(x, y)$ and then specialize at z, we get back the original. But specializing at x or y produces a new polynomial.

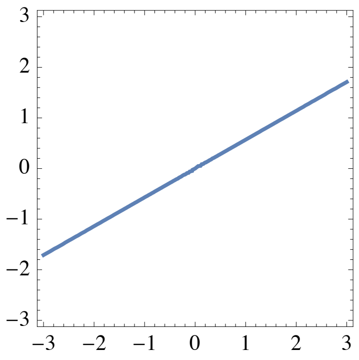
We say f is a singular curve if any complex projective singular point exists. So f may be a singular curve even though there are no affine singularities. We do this partly to be consistent with the algebraic geometers, but also because singular curves (even with infinite or complex singularities) do behave differently from regular curves.

Likewise, a curve f is *reducible* if its homogenization is reducible over the complex numbers. Because homogenization preserves polynomial multiplication, the homogeneous polynomial h is reducible if and only if all its specializations are reducible. It is fairly rare that a bivariate real polynomial has complex factors, but an important class of examples is the homogeneous functions in two variables. These always factor into linear factors, but some factors may be complex. Consider the next example.

```
g4 = x^3 - 2 x^2 y + x y^2 - y^3;
Factor[g4]
```

```
x^3 - 2 x^2 y + x y^2 - y^3
```

```
ContourPlot[g4 == 0, {x, -3, 3}, {y, -3, 3}, ImageSize → Small]
```



This seems to be irreducible, but the plot appears to be a straight line rather than a cubic. Furthermore, it is singular at $(0, 0)$. Think of this curve as a homogenization of a polynomial of one variable and specialize at $y = 1$.

```
g4y = g4 /. y → 1
```

```
-1 + x - 2 x^2 + x^3
```

```
sol5 = x /. NSolve[g4y];
g5 = Times @@ Table[(x - y sol5[[i]]), {i, 3}]
```

```
(x - 1.75488 y) (x - (0.122561 + 0.744862 i) y)
  (x - (0.122561 - 0.744862 i) y)
```

```
Chop[ComplexExpand[g5]]
```

```
x^3 - 2. x^2 y + 1. x y^2 - 1. y^3
```

So `g5` gives a complex numerical factorization of `g`; the two complex factors are invisible on the contour plot.

Related to singular points are intersection points. Here is an example.

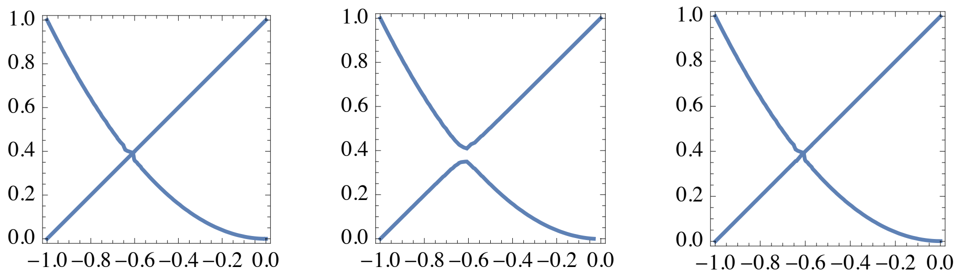
```
NSolve[{y^2 - x^4, y^2 + x^5}, {x, y}, Reals]
```

```
{{x → -1., y → -1.}, {x → -1., y → 1.},
 {x → 0., y → 0.}, {x → 0., y → 0.},
 {x → 0., y → 0.}, {x → 0., y → 0.}, {x → 0., y → 0.},
 {x → 0., y → 0.}, {x → 0., y → 0.}, {x → 0., y → 0.}}
```

We say the intersection of these curves at $(0, 0)$ has *multiplicity* 8. To explain what this means, particularly in the case of numerical curves, we use the formulation given in [4], which has been implemented numerically by Z. Zeng and the author. The implementation in the plane curve case is given in Appendix 1 of [1], the code and examples are in [2] and further information can be found in [5].

Intersections and singularities are connected, in that if f and g intersect at p, then the curve $f \times g$ has a singularity at p. However, there is an important difference. If we perturb a curve with a singularity by adding some terms with very small coefficients, the singularity often goes away. But if we perturb both of the curves intersecting at p, then locally we have the same multiplicity. Here is an example.

```
f = y - x^2;
g = y - x - 1;
Row[{
   ContourPlot[f g == 0, {x, -1, 0}, {y, 0, 1}, ImageSize → 150],
   ContourPlot[f g - .001 == 0, {x, -1, 0}, {y, 0, 1},
    ImageSize → 150],
   ContourPlot[(f - .001) (g + .001) == 0, {x, -1, 0},
    {y, 0, 1}, ImageSize → 150]
  }, Spacer[20]]
```



What this shows is that singularities are numerically unstable, but intersections are numerically stable. Thus in [1], which emphasizes the numerical point of view, we avoid getting deeply into singularities, but we can deal with intersections.

This leads to the most important theorem of complex projective plane algebraic geometry, Bézout's theorem.

Given complex algebraic curves $f(x, y)$ and $g(x, y)$ of degrees (respectively) m and n with no common nonconstant factor, there are exactly $m\,n$ complex projective points on both curves counting intersection multiplicity.

There are many proofs in the literature, and we will not give a complete proof here or in [1]. The complicating issue is when there are infinite or multiple intersection points. The typical proof involves use of the *resultant*. In the case of possibly infinite but not multiple points, one approach is to apply a random projective transformation. The resulting curves then, with high probability, will have no infinite intersection points and moreover, each intersection point will have a unique x coordinate. We can find these by applying the resultant with respect to x, which will then give a polynomial of degree $m\,n$ with distinct and hence non-multiple zeros. One can easily find the y coordinates of the transformed system by substituting each x in either equation and solving for y. Finally, transforming back will give the $m\,n$ solutions of the original system. We will study these transformations and find infinite intersection points by transforming, solving the affine system and transforming back in the next section.

As an example, consider the following Gaussian cubic and quadratic. There is one infinite solution. Applying the random projective linear transformation with matrix

$$\begin{pmatrix} 3 & 4 & -3 \\ 3 & 0 & 2 \\ -4 & 2 & 2 \end{pmatrix}$$

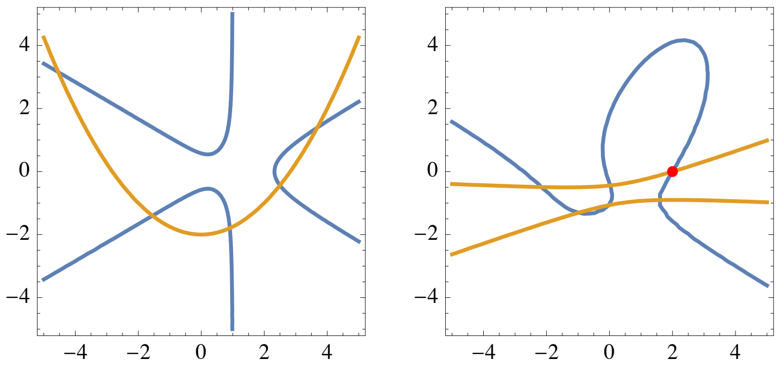
gives a system of equations that leads to polynomials with rational coefficients, no infinite solutions and unique x coordinates for the affine solutions.

```
f6 = -1 + 2 x - 3 x^2 + x^3 + 3 y^2 - 3 x y^2;
g6 = 4 y - x^2 + 8;
```

$$\texttt{f6r} = \frac{823}{137\,842} + \frac{13\,521\,x}{275\,684} - \frac{537\,x^2}{68\,921} - \frac{628\,x^3}{68\,921} + \frac{5073\,y}{275\,684} + \frac{2391\,x\,y}{68\,921} - \frac{567\,x^2\,y}{68\,921} + \frac{238\,y^2}{68\,921} + \frac{619\,x\,y^2}{68\,921} - \frac{583\,y^3}{68\,921};$$

$$\texttt{g6r} = \frac{428}{1681} - \frac{182\,x}{1681} - \frac{16\,x^2}{1681} + \frac{1372\,y}{1681} - \frac{276\,x\,y}{1681} + \frac{911\,y^2}{1681};$$

```
Row[{
  ContourPlot[{f6 == 0, g6 == 0}, {x, -5, 5}, {y, -5, 5},
   ImageSize → 200],
  Spacer[20],
  ContourPlot[{f6r == 0, g6r == 0}, {x, -5, 5}, {y, -5, 5},
   Epilog → {Red, PointSize[Medium], Point[{2, 0}]},
   ImageSize → 200]
 }]
```



Pictured are the original system and the transformed system. The indicated point in the second plot corresponds to the infinite solution {0, 1, 0} of the first plot. Even in this simple example with equations and transformation using one-digit integers, the resultant polynomial was a rational polynomial with numerators of 17 digits and a denominator of 21 digits!

Later in [1], Bézout's theorem is used in the discussion of Cayley's theorem and Harnack's theorem. In this section, we use Bézout's theorem to argue the *singularity theorem*:

An irreducible curve of degree d has at most $\frac{(d-1)(d-2)}{2}$ *complex projective singular points.*

In [1] I take a constructive point of view and show instead that a curve of degree d with $\frac{(d-1)(d-2)}{2}+1$ or more singular points is reducible. In the argument, we produce a polynomial of smaller degree that meets the given curve in too many points, so has a common factor with the given curve. In fact, in Appendix 1 of [1] we implement this argument with a function that factors the defining polynomial of any curve with $\frac{(d-1)(d-2)}{2}+1$ or more singularities.

Going back to the cubic, we homogenize and then specialize at $y = 1$.

```
f6 = -1 + 2 x - 3 x^2 + x^3 + 3 y^2 - 3 x y^2;
cp6 = criticalPoints[f6, x, y]
```

```
{{2.32472, 0.}, {0.096778, 0.554664}, {0.096778, -0.554664}}
```

The resulting plot shows the infinite points of f in the specialization where the dashed line is the original infinite line. The original infinite points are named `ip1`, `ip2`, `ip3`. The first critical point becomes the infinite point (0, 1, 0) in the x-z plane, and the other two go to the points `pcp2`, `pcp3`.

```
h6 = homog[f6, x, y, z] /. {y → 1}
```

$$-3x + x^3 + 3z - 3x^2 z + 2xz^2 - z^3$$

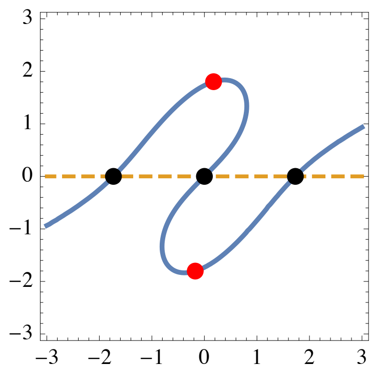
```
ip1 = {-Sqrt[3], 0};
ip2 = {0, 0};
ip3 = {Sqrt[3], 0};
cp6ha = Expand[Append[cp6[[2]], 1] / cp6[[2, 2]]][[{1, 3}]]
```

```
{0.17448, 1.80289}
```

```
cp6hb = Expand[Append[cp6[[3]], 1] / cp6[[3, 2]]][[{1, 3}]]
```

```
{-0.17448, -1.80289}
```

```
ContourPlot[{h6 == 0, z == 0}, {x, -3, 3}, {z, -3, 3},
 ContourStyle → {Thick, Dashed},
 Epilog → {{PointSize[Large], Black, Point[{ip1, ip2, ip3}]},
   {PointSize[Large], Red, Point[{cp6ha, cp6hb}]}},
 ImageSize → Small]
```



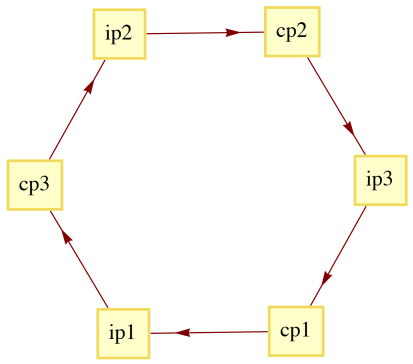
So in the projective plane, infinite points look just like affine points. We can trace projective paths just like affine paths. Thus, we can form graphs just like in the affine case; in particular, the projective graphs now have the property that every vertex is even. This gives my *fundamental theorem of real plane projective algebraic curves*, henceforth called just the *fundamental theorem*, which completely describes the topology of the projective curve.

Let $h(x, y, z) = 0$ *be a homogeneous real plane projective algebraic curve* C*. Then there is a finite set of points* V *in* C*, called vertices, and a set of edges* E *between pairs of vertices satisfying*:

1. *Each edge corresponds to a continuous arc (or path) in* C *connecting the two vertices.*
2. *Every singular point of* C *is a vertex.*
3. *The interiors of any two arcs corresponding to edges are disjoint; that is, arcs only meet at vertices.*
4. *Every point of* C *is either a vertex or an interior point of an arc.*
5. *The graph is an Euler graph; that is, every vertex is even.*

In the previous example, the graph can be rendered as follows, where the vertex names refer to the original affine specialization.

```
GraphPlot[{"ip2" → "cp2", "cp2" → "ip3", "ip3" → "cp1",
   "cp1" → "ip1", "ip1" → "cp3",  "cp3" → "ip2"},
 VertexLabeling → True, DirectedEdges → True,
 ImageSize → Small]
```



Several comments are in order. First, *critical points* are not a concept in the projective plane; they come from some affine specialization. They make good vertices, but in this context are somewhat arbitrary. The same is true of the *direction* of the curve, but these graphs can be given a directed Euler graph structure. The fact that these are Euler graphs implies they can be decomposed into (not necessarily disjoint) directed circuits.

Already in his 1799 proof of the fundamental theorem of algebra, Gauss essentially calculates the infinite points of Gauss curves coming from a monic polynomial $p(z)$ of degree d as

$$\mathrm{ip}_j = \left(\cos\left(\frac{(2\,j+d-1)\,\pi}{2\,d}\right),\ \sin\left(\frac{(2\,j+d-1)\,\pi}{2\,d}\right), 0\right) \text{ for } j = 1, \ldots, d.$$

Since the Gaussian curve already approximately intersects large circles about the origin in the affine points (and their antipodal points) given by the first two coordinates, one can infer that the graph will have edges pointing directly out from boundary points on a large circle to the appropriate infinite point. Thus by treating any two antipodal points of the curve on a large circle about the origin as the same infinite vertex, we convert the bounded graph to the projective graph.

A more interesting example of a Gaussian curve is Gauss's example, which has two components and a singular point.

```
f7 = gaussCurve[z^5 - 5 z^4 + 9 z^3 - 5 z^2, x, y, z]
```

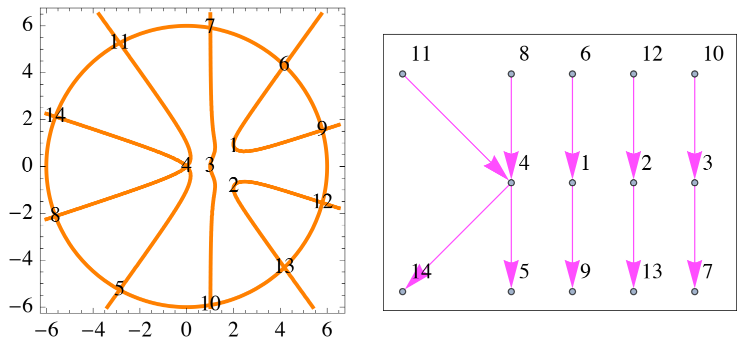
$$-5\,x^2 + 9\,x^3 - 5\,x^4 + x^5 + 5\,y^2 - 27\,x\,y^2 + 30\,x^2\,y^2 - 10\,x^3\,y^2 - 5\,y^4 + 5\,x\,y^4$$

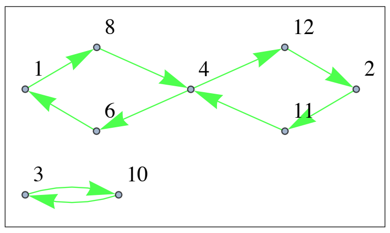
We find the critical and boundary points on a circle of radius 4 and put them in an association for labeling.

```
g7p = Chop@Join[
     criticalPoints[f7, x, y],
     {x, y} /. NSolve[{f7, x^2 + y^2 - 36}, {x, y}, Reals]
    ];
g7A = <|Table[i → g7p[[i]], {i, Length[g7p]}]|>;
```

We show a contour plot and the bounded graph. Then, by treating boundary points as infinite points and identifying pairs of antipodal points (7 ↔ 11, 8 ↔ 13, 9 ↔ 6, 5 ↔ 12, 14 ↔ 10), here is the projective graph.

```
Row[{
   Show[
    ContourPlot[{f7 == 0, x^2 + y^2 == 36}, {x, -6, 6.5},
     {y, -6., 6.5}, ContourStyle → Orange],
    Graphics[Table[{Text[i, g7A[i]]}, {i, 14}]],
    ImageSize → 150],
   Framed@
    Graph[{6 → 1, 1 → 9, 12 → 2, 2 → 13, 11 → 4, 4 → 14,
      8 → 4, 4 → 5, 10 → 3, 3 → 7}, VertexLabels → "Name",
     EdgeStyle → Magenta, ImageSize → 150],
   Framed@Graph[{6 → 1, 1 → 8, 12 → 2, 2 → 11, 11 → 4,
      4 → 12, 8 → 4, 4 → 6, 10 → 3 , 3 → 10},
     VertexLabels → "Name", EdgeStyle → Green, ImageSize → 150]
  }, Spacer[10]]
```





We mention the Riemann–Roch theorems, whose main subject is the concept of *genus*. These theorems are the backbone of complex curve theory and even real space curve theory. However, for real plane curves the important invariant of a curve is the degree, not genus, so we do not dwell on these theorems.

6. Fractional Linear Transformations

An important tool in [1] is utilizing the projective linear transformations. We follow Abhyankar [6] by keeping the discussion mainly in the affine realm, where it is easier to compute, viewing these as *fractional linear transformations*.

A *fractional linear transformation* is a function Φ defined by

$$\Phi(x, y) = \left(\frac{a_{11}\,x + a_{12}\,y + a_{13}}{d_1\,x + d_2\,y + d_3}, \frac{a_{21}\,x + a_{22}\,y + a_{23}}{d_1\,x + d_2\,y + d_3}\right),$$

where a_{ij} and d_k are real (or sometimes complex) numbers in the form of integers or machine numbers. Setting the common denominator to zero defines a line, so the domain of Φ is the affine plane minus this line.

The notation suggests describing the fractional linear transformation compactly by the matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ d_1 & d_2 & d_3 \end{pmatrix}.$$

This is more compact as well as useful, as the fractional linear transformation is actually given by the two-step procedure using matrix multiplication:

$$(x, y) \mapsto A\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = (u, v, w) \mapsto \left(\frac{u}{w}, \frac{v}{w}\right).$$

In the Wolfram Language, this becomes the function `flt`.

```
flt[{x_, y_}, A_] := Module[
  {u, v, w},
  {u, v, w} = A.{x, y, 1};
  If[Abs[w] < 1.*^-6, {}, {u / w, v / w}]
 ]
```

To the extent that we want to work completely in the affine domain, we note that the Wolfram Language also includes fractional linear transformation under the name *linear fractional transformation*. So one can also use the Wolfram Language `TransformationFunction` to evaluate a fractional linear transformation.

Here is an example.

```
A = {{1, 0, -2}, {0, -1, 0}, {0, 1, 1}};
A // MatrixForm
```

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

```
flt[{2, 1}, A]
```

$\left\{0, -\frac{1}{2}\right\}$

```
TransformationFunction[A][{2, 1}]
```

$\left\{0, -\frac{1}{2}\right\}$

In [1], to keep things simple we assume the matrix A is invertible. Matrix multiplication corresponds to composition of transformations; in particular, since our matrices are invertible, so are our fractional linear transformations.

Somewhat unique to [1], we have our transformations work on curves as well as points.

```
FLT[f_, A_, x_, y_] := Module[
  {fh, gh, z},
  fh = Expand[z^tDeg[f, x, y] (f /. {x → x / z, y → y / z})];
  gh = Expand[fh /. Thread[{x, y, z} → Inverse[A].{x, y, z}]];
  Chop[gh /. {z → 1}, 1*^-10]
 ]
```

The fractional linear transformation Φ takes the curve (i.e. the bivariate polynomial f) to a curve g such that $g(\Phi((x, y)) = 0$ whenever $f(x, y) = 0$.

Here is an example using A as defined before.

```
f8 = x^2 + y^2 - 1;
g8 = FLT[f8, A, x, y]
```

$3 + 4\,x + x^2 + 6\,y + 4\,x\,y + 4\,y^2$

The relationship between `FLT` and `flt` is shown by the following example; `flt` maps points to points and `FLT` maps curves to curves. The image of a point under `flt` is a point of the image of the curve under `FLT`.

```
g8 /. Thread[{x, y} → flt[{.6, .8}, A]]
```

-1.11022×10^{-16}

In this case, the transformation takes the circle to a conic, a parabola. One can use the various transformations given by the Wolfram Language. We provide some additional ones in [2] (such as `klRotation`, which takes line k to line l, and `kReflection`, the reflection about the line k) as Euclidean transformations and `kShear` as an affine transformation. As an example, we give `klRotation`.

```
klRotation[k_, l_, x_, y_] := Module[
  {a, b, p, T, R},
  a = Coefficient[k, {x, y}];
  b = Coefficient[l, {x, y}];
  If[Abs[Det[{a, b}]] < 1.*^-9,
    Print["No affine center"]; Return[Fail]
   ];
   p = {x, y} /. First@NSolve[{k, l}];
   T = {{1, 0, -p[[1]]}, {0, 1, -p[[2]]}, {0, 0, 1}};
   R = Join[Append[N[RotationMatrix[{a, b}]], {0, 0}],
     {{0}, {0}, {1}}, 2];
   Inverse[T].R.T
 ]
```

More importantly, we have two fractional linear transformations that act on the projective plane. The transformation `ip2z` takes the infinite point *p* to the origin and the original infinite line to the *x* axis. The transformation `iTransform` specializes the projective plane by removing the line *l* from the affine plane and making it the infinite line; the new *x* axis is the original infinite line.

```
ip2z[ip_, x_, y_] := Module[
  {p, B},
  If[Length[ip] ≠ 3,
   Print["not infinite point"]; Abort[],
   If[Abs[ip[[3]]] > 1.*^-6,
    Print["not infinite point"]; Abort[],
    p = Chop[ip, 1.*^-6]
   ]
  ];
  B = {
    {1.414213562373095`, 0.`, 0.`},
    {0.`, 0.`, 1.414213562373095`},
    {0.`, -0.7071067811865475`, -0.7071067811865475`}
   };
  If[Abs[First@p] < 1.*^-6, Return[B]];
  B.klRotation[line[p, {0, 0}, x, y], x, x, y]]
```

As an example, we are interested in the behavior of the infinite point of the preceding curve *g*8.

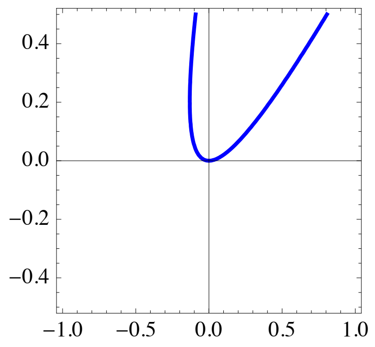
```
g8 = 3 + 4 x + x^2 + 6 y + 4 x y + 4 y^2;
ip8 = infiniteRealPoints[g8, x, y]
```

```
{{-0.894427, 0.447214, 0.}}
```

```
h8 = FLT[g8, ip2z[First@ip8, x, y], x, y]
```

```
2.5 x^2 - 0.894427 y - 3.57771 x y + 1.05279 y^2
```

```
ContourPlot[{h8 == 0}, {x, -1, 1}, {y, -.5, .5},
 ContourStyle → Blue, ImageSize → Small, Axes → True]
```



The transformation `ip2z` puts the infinite point at the origin of this plot, which shows the infinite line as the x axis. It appears that the parabola $g8$ is actually tangent to the infinite line at the infinite point. To check, we can calculate the tangent line to $h8$ at the origin to see it is the x axis, that is, $y = 0$.

```
tLine[h8, {0, 0}, x, y]
```

```
0. + 1. y
```

There are various alternate versions of the functions `flt` and `FLT` to handle working projectively. For example, `flti` accepts infinite points as input or returns them as output.

The main application for `iTransform` is that we can now find all complex projective singular points or intersection points in one step by picking a random line that, with high probability, will not go through any of the finite number of singular or intersection points. For details, see [1].

7. Applications to Geometry

In [1], the theory so far is applied to recover known results in lower-dimension geometry. First we consider nonsingular conics in the form

$$a_1 x^2 + a_2 x y + a_3 y^2 + a_4 x + a_5 y + a_6 = 0,$$

where the a_j are integers or machine numbers. We identify them as to type (hyperbola, parabola, ellipse) and write them in standard form. We parameterize them by rational or trigonometric functions, find their foci and directrix or conversely, construct them from arbitrary foci and another appropriate value such as the semilatus rectum.

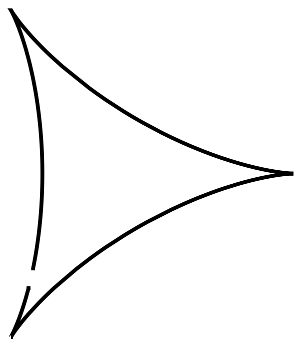
We then discuss the numerical theory for nonsingular cubics. Unlike the number theory case, which is one of the most difficult subjects in mathematics, the numerical case is very simple. We give a function to find the numerical inflection points; then, with a choice of inflection point, we have a deterministic black-box function to calculate the Weierstrass normal form and j-invariant. The j-invariant almost completely classifies numerical cubics relative to fractional linear transformations, that is, relative to the real projective linear group: there are two conjugate classes for each j. Under the complex projective linear group, the classification is complete.

We end with Cayley's theorem: an irreducible curve of degree d with $\frac{(d-1)(d-2)}{2}$ double points has a rational parameterization. This means that with parameter t, the parameterization components have the form of a polynomial in t divided by another polynomial in t. The coefficients are not, however, expected to be rational numbers; Cayley only promises algebraic numbers. Thus in practice, this parameterization works best with machine-number coefficients. We illustrate by parameterizing the hypocycloid.

```
ϕx =
  -((0.3946283194078104`
       (-0.19419306729140778` + 0.5008120016643439` t +
         2.342269796866374` t^2 + 2.100529395812897` t^3 +
         1.` t^4)) /
      (0.16466880387583177` + 0.6038407805584985` t +
        1.3651593716897787` t^2 + 1.4880474310771863` t^3 +
        1.` t^4));

ϕy =
  -((0.5577361971418063`
       (0.04150396033309846` + 0.5103350783662945` t +
         2.153089385284384` t^2 + 3.3532487723268503` t^3 +
         1.` t^4)) /
      (0.16466880387583177` + 0.6038407805584985` t +
        1.3651593716897787` t^2 + 1.4880474310771863` t^3 +
        1.` t^4));

ParametricPlot[{ϕx, ϕy}, {t, -20, 20}, PlotStyle → Black,
 PlotRange → All, Axes → False, ImageSize → Small]
```

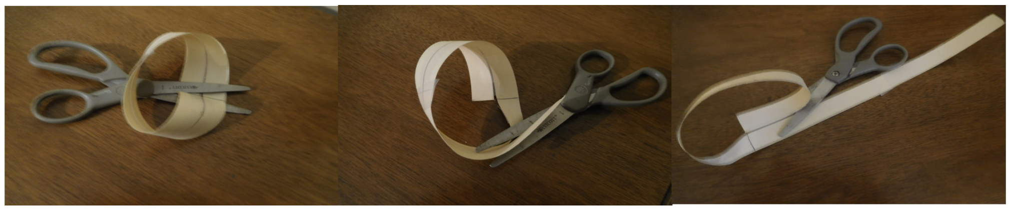


The gap occurs because we only plotted the parameter on a closed interval; in theory it should run from $-\infty < t < \infty$. Details of how the parametric functions were calculated are in [1].

8. The Möbius Band Model of the Real Projective Plane

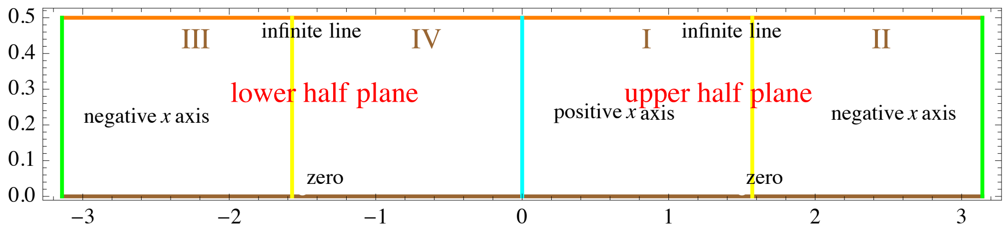
Topologists often think of the real projective plane as a Möbius band where the entire outer boundary is squashed to the affine origin. Alternatively, the Möbius band can be viewed as the real projective plane with a tiny disk about the affine origin removed, the boundary of that disk being the boundary of the Möbius band. In either case, the center line of the band is the *infinite line*.

It is common to construct a Möbius band out of a strip of paper. Here is a slightly different but useful way, but shown in Figure 1 by a physical deconstruction: cut from a boundary point to the center (infinite) line, then cut around the center line.



▲ **Figure 1.** Constructing a Möbius band.

This gives a long skinny strip that we can identify with the real projective plane shown in Figure 2. The vertical yellow lines are the negative and positive y axes, and the standard quadrants of the affine plane are numbered in Roman numerals.



▲ **Figure 2.** The real projective plane.

We implement the mappings from the projective plane to this strip, called the *rectangular hemisphere* in [1] for reasons given there, and from the strip to the Möbius band by the following functions.

1. Map from affine plane to rectangular hemisphere.

```
moebiusPhi[p_] := If[
  Length[p] == 2 && p[[2]] ≠ 0,
  {ArcTan @@ N[p], N[ArcTan[Norm[p]] / Pi]},
  Echo[p, "ambiguous point "]; {}
 ]
```

2. Map from rectangle to Möbius band.

```
moebiusBAux[t_] := Module[
   {a = N[Sqrt[2] - 1]},
   {
    Cos[2 t + Pi] / (1 + a Cos[t]),
    Sin[2 t + Pi] / (1 + a Cos[t]),
    1 / (1 + a Cos[t + Pi / 2])
   }
  ]
```

```
moebiusB[s_, t_] := t moebiusBAux[s] + (1 - t) moebiusBAux[s + Pi]
```

3. Plot Möbius band with infinite line.

```
moebiusPLMB := Show[
   ParametricPlot3D[moebiusB[t, s], {t, -Pi, Pi},
    {s, 0, .5}, Mesh → None, PlotStyle → Opacity[0.8]],
   ParametricPlot3D[moebiusB[t, .5], {t, 0, Pi},
    PlotStyle → {Thick, Blue, Dashed}],
   Boxed → False, Axes → False]
```

A simple example is the hyperbola $x^2 - y^2 - 4$; we give the construction. The infinite points are ip1 = $\{1, 1, 0\}$ and ip2 = $\{1, -1, 0\}$. Unfortunately, even this simple example takes up a great deal of space, so we will just get started. We consider the part of this hyperbola in the second quadrant of the affine plane and plot it on the Möbius band. We start at the infinite point $\{-1, 1, 0\}$ and trace to the infinite point $\{-2, 0\}$, which is an ambiguous point.

```
f9 = x^2 - y^2 - 4;
```

The affine part is well known, so we inspect the obvious infinite points $\{1, 1, 0\}, \{1, -1, 0\}$.

```
A9 = ip2z[{-1, 1, 0}, x, y]
```

```
{{1., 1., 0.}, {0., 0., 1.41421}, {0.5, -0.5, -0.707107}}
```

A technicality is that `ip2z` uses the line function, which could randomly differ by a multiple of -1. We need a specific choice, so we set the value of `A9`.

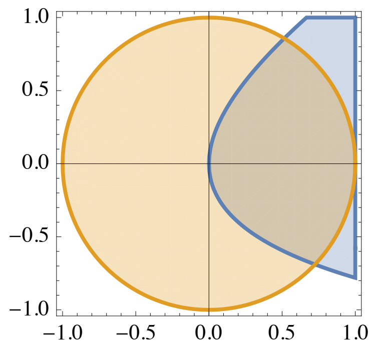
```
A9 = {{-1.0000000000000002`, -1.0000000000000002`, 0.`},
    {0.`, 0.`, 1.414213562373095`},
    {-0.5000000000000001`, 0.5000000000000001`,
     -0.7071067811865475`}};
```

```
h9 = FLT[f9, A9, x, y]
```

```
2. x + 1. x y - 2. y^2
```

Again, the x axis represents the infinite line for `f9`, and the origin the infinite point.

```
RegionPlot[{h9 > 0, x^2 + y^2 < 1}, {x, -1, 1}, {y, -1, 1},
 Axes → True, ImageSize → Small]
```



To connect this plot to the affine curve, find the points where `h9` intersects the circle of radius 1.

```
bp9 = {x, y} /. NSolve[{h9, x^2 + y^2 - 1}, {x, y}, Reals]
```

```
{{0.514587, 0.857438}, {0.724617, -0.689152}}
```

Now map these back to the affine plane to see that it is the first point in `bp9` that is related to a point in the second quadrant.

```
ap9 = flt[#, Inverse[A9]] & /@ bp9
```

```
{{-2.78082, 1.93209}, {2.08849, -0.601504}}
```

So the part of the hyperbola in the second quadrant can be traced using two parts: the part from the x intercept $\{-2, 0\}$ to the point `ap9[[1]]` and the image of the part from `bp9[[1]]` to the infinite point represented by $\{0, 0, 1\}$.

```
L91 = flt[#, Inverse[A9]] & /@
   pathFinderT[h9, {0, 0}, First@bp9, .05, x, y];
J91 = pathFinderT[f9, First@ap9, {-2, 0}, .2, x, y];
K91 = Join[L91, J91];
```

We see no error messages, so we assume the tracing went correctly. Now apply `moebiusPhi`.

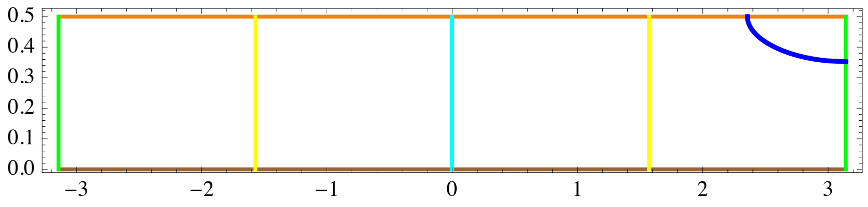
```
Q1 = moebiusPhi[#] & /@ K91;
```

```
» ambiguous point {}
» ambiguous point {-2, 0}
```

Here we get some warning messages. We have to set the ambiguous points correctly and can then draw Figure 3.

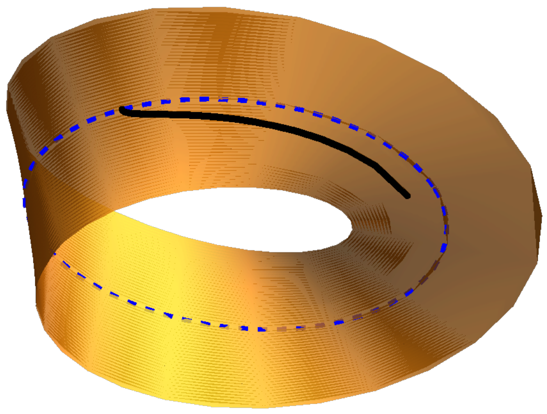
```
Q1[[1]] = {N[ArcTan[-1, 1]], .5};
Q1[[-1]] = N[{Pi, ArcTan[2] / Pi}];
```

So we have drawn this section of the hyperbola on the rectangular hemisphere.

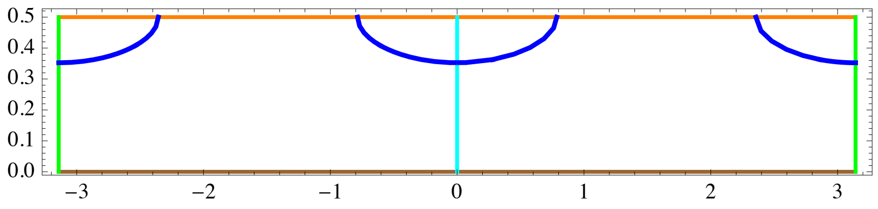


▲ **Figure 3.** The part of the hyperbola in the second quadrant on the rectangular hemisphere.

```
M1 = moebiusB @@@ Q1;
Show[Graphics3D[{Thick, Black, Line[M1]},
  ViewPoint → {-1.09, 2.53, -1.97}], moebiusPLMB,
 Boxed → False]
```

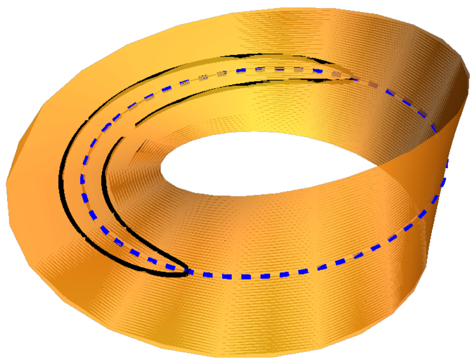


The reader may wish to attempt the other sections of the hyperbola; the one in the third quadrant is similar to `Line[Q1]`, and the parts in the first and fourth quadrants can be done together since the x intercept (2, 0) does not give an ambiguity (Figure 4).



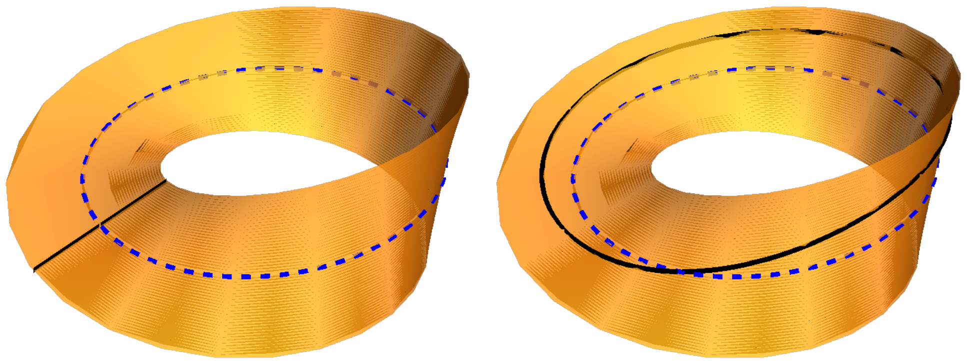
▲ **Figure 4.** The full plot of the hyperbola on the hemisphere looks like this.

Finally, we lift to the actual Möbius band using `moebiusB` and `plot` (Figure 5).



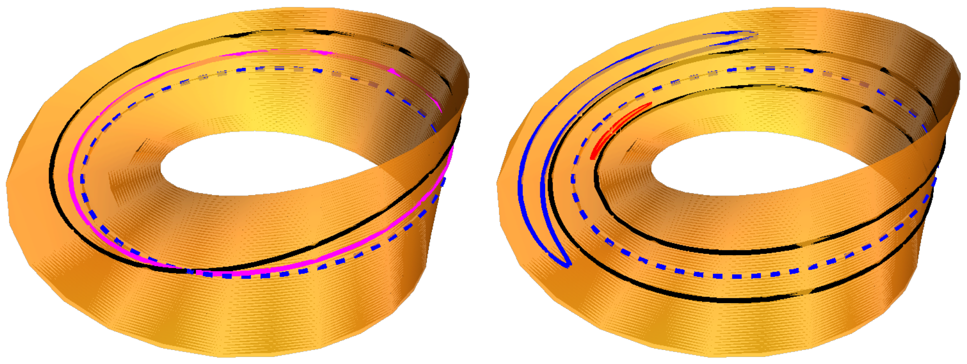
▲ **Figure 5.** The hyperbola on a Möbius band.

This last example was simple! From now on we just show the final output (Figure 6).



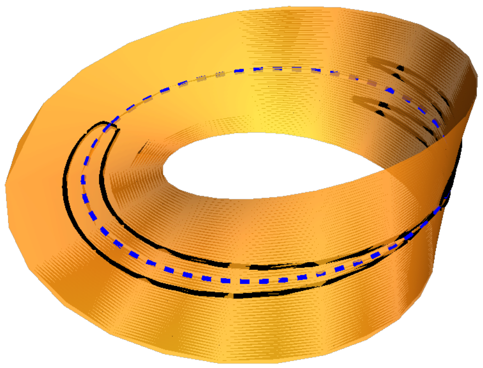
▲ **Figure 6.** Here are two Möbius plots of lines, the first a line through the origin, and then a typical line.

Next, Figure 7 shows two affinely parallel lines meeting at an infinite point and three circles; the black one contains the origin in its interior.



▲ **Figure 7.** Affinely parallel lines and three circles.

In Figure 8, we plot the rational function $y = \frac{4}{x(x^2-4)}$. This has two infinite points: a singular one at $(0, 1, 0)$ and a regular one at $(1, 0, 0)$.



▲ **Figure 8.** Plot of a rational function.

Experimenting with these plots we see, as the fundamental theorem tells us, that these curves are comprised of loops, that is, simple closed curves. Draw these yourself using the pattern in our hyperbola example that stops at the rectangle, print it out (preferably in landscape orientation with smaller aspect ratio), then cut it out, twist and tape together the two copies of the infinite line to make the Möbius band. Now cut out a loop. Two things can happen: either you get two pieces, one of topologically a disk and the other not, or only one piece, as in the classic example of cutting a Möbius band along the center line.

In the first case, we call the loop an *oval*, and the complementary piece shaped like a disk is called the *interior*. In the other case, we call this a *pseudo-line*. Notice both kinds of lines have this property. One easy way to tell, without going through the trouble of constructing a physical Möbius band, is that an oval meets the infinite line (or any other line for that matter, since up to fractional linear transformations all lines are the same) in an even number of points (possibly zero). A pseudo-line meets the infinite line in an odd number of points. Again, since in the projective plane all lines are equivalent, two pseudo-lines always meet in an odd number of points, in particular, at least one.

From Bézout's theorem, a curve of even degree meets any line in an even number of points. A consequence is that *a nonsingular curve can contain at most one pseudo-line. Further, if the degree is even, each loop of the curve must be an oval. On the other hand, each nonsingular curve of odd degree must contain exactly one pseudo-line and possibly some ovals.*

This last paragraph is in italics because it essentially tells us the topological structure of nonsingular plane curves.

9. Diamond Diagrams

We can now find the specific topological (and even some geometrical) structure of any particular real plane curve, at least up to degree six. We concentrate on the Newton's hyperbola family of curves introduced in Section 2. These are not well conditioned, so they present interesting problems. It may be necessary to go to arbitrary-precision numbers to get further with these, although for well-conditioned curves I have used the methods of [1] for curves up to degree nine.

We first consider Harnack's theorem [7] and related problems from Hilbert's 16th problem, Part 1 [8]. Harnack's first theorem states that a nonsingular curve of degree d can have at most $m = \frac{(d-1)(d-2)}{2} + 1$ topological components in $\mathbb{P}^2$. A rigorous proof requires advanced concepts in topology, but a heuristic proof is easy from Bézout's theorem, especially given the ideas of ovals and pseudo-lines in the last section.

As mentioned in the last section, an oval is a loop that cuts the Möbius band in two parts, one topologically a disk. That part is known as the *interior* of the oval. It is possible that the interior of an oval contains another oval. Consider the following example with fractional linear transformation given by matrix `ZapY` that cuts the y axis out of the affine plane.

```
ZapY = {{0, 1, 0}, {0, 0, 1}, {1, 0, 0}};
```

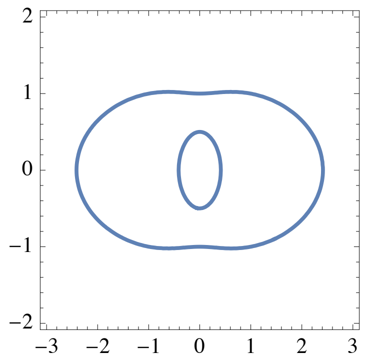
```
gc2 = gaussCurve[(z^2 - 1) (z^2 - 4), x, y, z]
```

$4 - 5 x^2 + x^4 + 5 y^2 - 6 x^2 y^2 + y^4$

```
d2 = FLT[gc2, ZapY, x, y]
```

$1 - 6 x^2 + x^4 - 5 y^2 + 5 x^2 y^2 + 4 y^4$

```
Labeled[ContourPlot[d2 == 0, {x, -3, 3}, {y, -2, 2},
  ImageSize → Small], Text@"oval inside an oval"]
```



oval inside an oval

We say the smaller oval has *depth* 2. If there were another oval inside that oval, it would have depth 3, and so on. It is easy to prove that the maximal depth of an oval in an irreducible curve of degree d is $d/2$; simply consider a line through a point in the interior of the deepest oval and apply Bézout's theorem. The next example generalizes this.

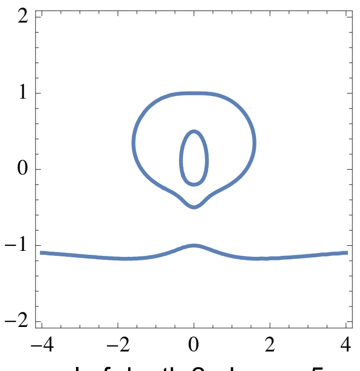
```
gd5 = gaussCurve[(z^2 - 1) (z^2 - 4) (z + 5), x, y, z];
d5 = FLT[gd5, ZapY, x, y]
```

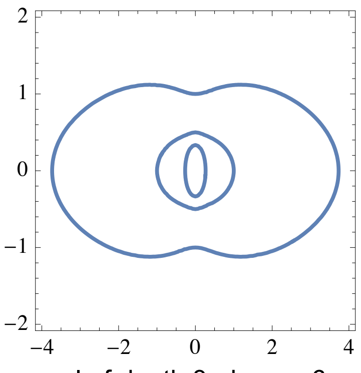
$$1 - 10 x^2 + 5 x^4 + 5 y - 30 x^2 y + 5 x^4 y - 5 y^2 + 15 x^2 y^2 - 25 y^3 + 25 x^2 y^3 + 4 y^4 + 20 y^5$$

```
gd6 = gaussCurve[(z^2 - 1) (z^2 - 4) (z^2 - 9), x, y, z];
d6 = FLT[gd6, ZapY, x, y]
```

$$1 - 15 x^2 + 15 x^4 - x^6 - 14 y^2 + 84 x^2 y^2 - 14 x^4 y^2 + 49 y^4 - 49 x^2 y^4 - 36 y^6$$

```
Text@
 Row[
  {Labeled[ContourPlot[d5 == 0, {x, -4, 4}, {y, -2, 2},
     ImageSize → Small], "oval of depth 2, degree 5"],
   Spacer[20],
   Labeled[ContourPlot[d6 == 0, {x, -4, 4}, {y, -2, 2},
     ImageSize → Small], "oval of depth 3, degree 6"]}]
```


oval of depth 2, degree 5


oval of depth 3, degree 6

Continuing this way, we can in principle construct an oval of depth $d/2$ using a curve of degree d for d even and a curve of depth $\frac{d-1}{2}$ for d odd.

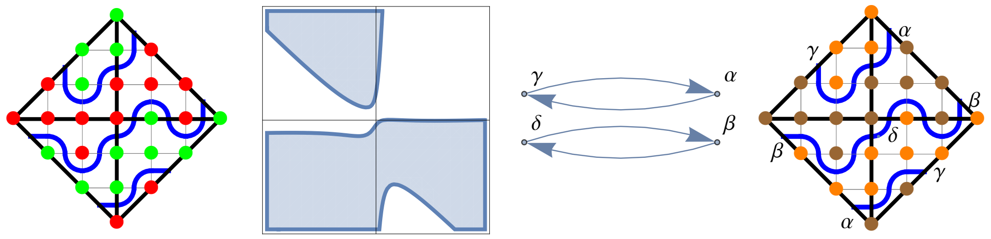
An M-curve is a nonsingular curve with the maximum number m of components. To best show the possible arrangements of the components of an M-curve, we use *diamond diagrams*. We have two main types, first the *Descartes–Viro diagrams* (or more simply the *Viro diagrams*), which depend on the signs of coefficients of the equation $f(x, y)$ of the curve [9]. These diagrams turn out to be in 1-1 correspondence with the Newton hyperbolas. We also use *Gauss diagrams*, which show the complementary positive and negative value sets of $f(x, y)$.

The code for drawing diamond diagrams is very long and explained in [1] and [2]; in this article we do not give code, only graphics.

For the Viro diagram in the first quadrant including the positive axes, the color of the dot at the point (i, j) is green if the coefficient of $x^i y^j$ is positive and red if it is negative. We do not allow equations with any term 0 for a Viro diagram. The curve then separates the red and green lattice points.

As an example, consider Newton hyperbola 413 (Figure 8).

```
nh413 = -1 + x - 0.2 x^2 + 0.008 x^3 - y - x y - 0.2 x^2 y + 0.2 y^2 -
    0.2 x y^2 + 0.008 y^3;
```



▲ **Figure 9.** The Viro diagram, region plot, graph and diamond diagram for the function `nh413`.

In this case, the Viro diagram and Gauss diagram (not shown) are the same, other than the color of the lattice points; orange indicates where $f > 0$ and brown where $f < 0$. A graph is given using only the infinite points, which are labeled $\alpha, \beta, \gamma, \delta$. The outer boundary of the diamond represents the infinite line. The diamond diagram indicates that: (1) on the positive x axis the curve crosses three times; (2) it does not cross the negative x axis; and (3) it crosses the positive y axis once and the negative y axis twice. The Viro diagram gives the maximal number of crossings according to Descartes's theorem on each positive and negative x, y and z axis, viewing f as a single-variable polynomial restricted to these lines. In the projective plane, the z axis is the line of infinite points $\{x, y, 0\}$ where infinite points in the first/third quadrant are positive and those in the second/fourth quadrant are considered negative in this context.

In this example, the crossing points are given as follows.

```
xCross = {x, y} /. NSolve[{nh413, y}, {x, y}, Reals]
```

```
{{1.33975, 0}, {5., 0}, {18.6603, 0}}
```

```
yCross = {x, y} /. NSolve[{nh413, x}, {x, y}, Reals]
```

```
{{0, -29.1421}, {0, -0.857864}, {0, 5.}}
```

Let α, β, γ be the infinite points.

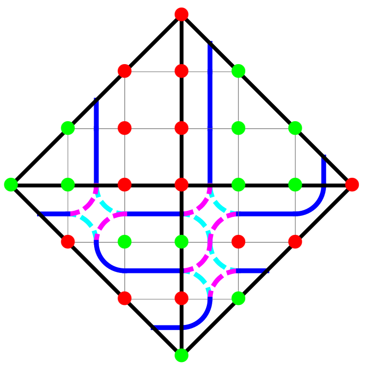
```
Normalize /@ infiniteRealPoints[nh413, x, y]
```

```
{{-0.0384901, -0.999259, 0.},
 {-0.999259, -0.0384901, 0.}, {-0.707107, 0.707107, 0.}}
```

The Newton hyperbola 613 is more complicated (Figure 10).

```
nh613 = newtonHyperbola[613, x, y]
```

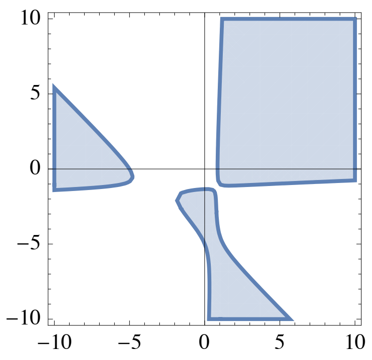
$-1 + x + 0.2\, x^2 - 0.008\, x^3 - y + x\, y + 0.2\, x^2\, y - 0.2\, y^2 + 0.2\, x\, y^2 - 0.008\, y^3$



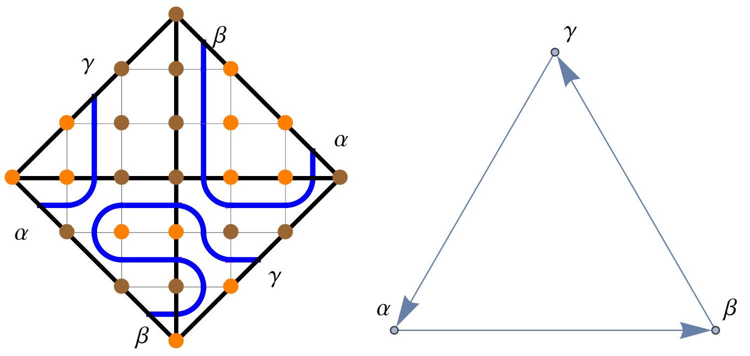
▲ **Figure 10.** The Viro diagram for the function `nh613`.

We see there are three ambiguous cells, that is, four lattice points $\{(i, j), (i+1, j), (i, j+1), (i+1, j+1)\}$ with $(i, j), (i+1, j+1)$ one color and $(i+1, j), (i, j+1)$ the other. There are two different possible ways to connect regions given by dashed curves in the colors aqua and magenta. Without further investigation, there is no a priori way to determine the correct choice; a slight perturbation of the curve can affect this. A `RegionPlot` suggests an answer.

```
RegionPlot[nh613 > 0, {x, -10, 10}, {y, -10, 10},
 Axes → True, ImageSize → Small]
```



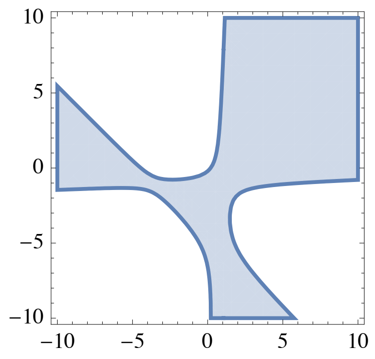
Checking infinite points and critical points confirms that there is nothing unexpected going on outside the region plot, so we get the Gauss diagram and graph (Figure 11).

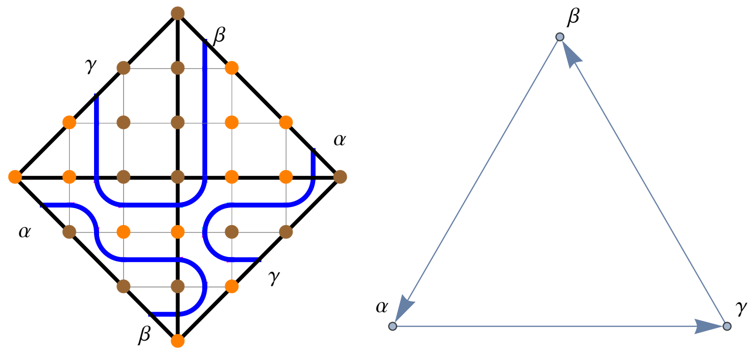


▲ **Figure 11.** Gauss diagram and graph.

In this case, a tiny perturbation changes the geometry and the Gauss diagram (Figure 12).

```
h10 = nh613 + .8;
RegionPlot[h10 > 0, {x, -10, 10}, {y, -10 , 10},
 ImageSize → Small]
```

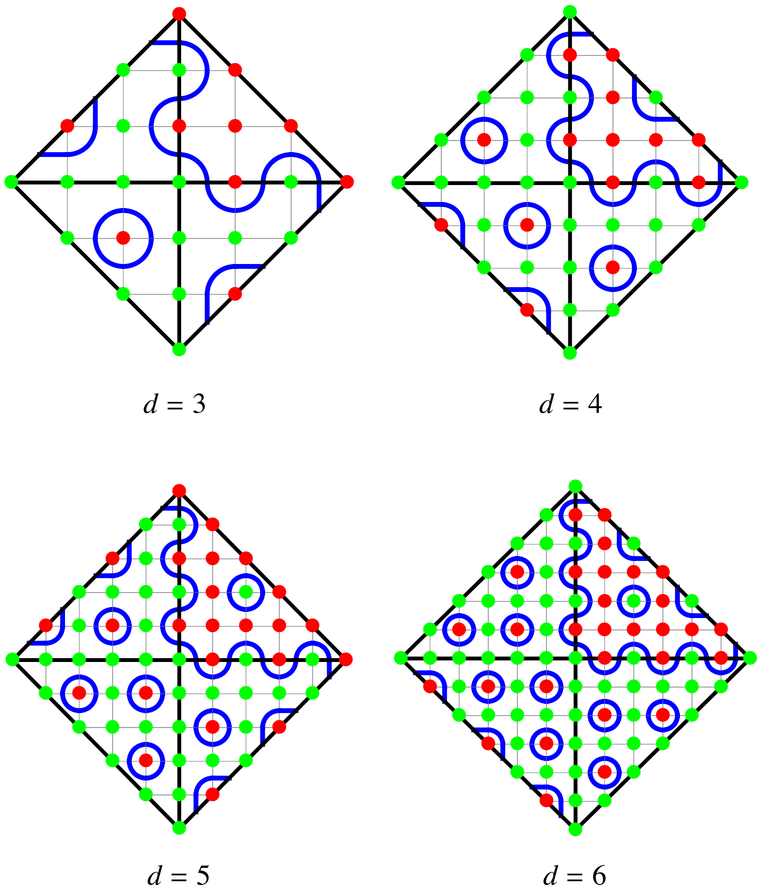




▲ **Figure 12.** The region plot and Gauss diagram for the perturbed `nh613` are different from `nh613`.

Originally, the negative complement was connected and the positive complement had three components; after the perturbation, it is the positive complement that is connected. Luckily, we did not need to change any values of the lattice points when changing from the Viro diagram to the Gauss diagram. In general, the user will need to do that. See [1] or some later examples.

Now that we have explained our diagrams, we can show some M-curves. Hilbert gave a series of M-curves for each degree that are given by Viro diagrams and hence exist by the work of Viro. We simply give the diagrams here (Figure 13). For more information see [1].



▲ **Figure 13.** Viro diagrams of M-curves of degree d.

Hilbert suggested other possibilities with more nesting in degree six.

Many more details on these diagrams and Hilbert's 16th problem [8] are given in [1].

We now have all of our tools. In [1] we illustrate more complicated examples, two of them the curve $(x^2 + y^2)^2 - 4x^3 y^2$ and the Newton hyperbola 336941. Both of these curves have interesting behavior at or near the infinite line, so a contour plot, even with large scale, cannot show everything.

10. Conclusion

At present, we have shown how to analyze and plot curves of degree up to six in various ways. For well-conditioned curves, these machine-number methods often work with higher degree; the author has had success with curves of degree eight and nine. To adequately deal with Newton hyperbolas of degree greater than six, one would perhaps like to rewrite some of the code to use arbitrary precision.

Our forthcoming book is a first attempt to apply numerical methods to a formerly abstract subject. There is a lot more that can be done in this area. We hope the book will be a starting point.

Acknowledgments

I want to thank the people at Wolfram Research for their help on the book project, especially Jeremy Sykes, Daniel Lichtblau and, for this article, George Beck.

References

[1] B. H. Dayton, *A Numerical Approach to Real Algebraic Curves with the Wolfram Language*, Champaign, IL: Wolfram Media, 2018. www.wolfram-media.com/products/dayton-algebraic-curves.html.

[2] Global Functions. (Jul 18, 2018) barryhdayton.space/curvebook/GlobalFunctionsTMJ.nb.

[3] E. W. Weisstein. “Bifurcation” from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/Bifurcation.html.

[4] F. S. Macaulay, *The Algebraic Theory of Modular Systems*, Cambridge: Cambridge University Press, 1916.

[5] B. H. Dayton, T.Y. Li, Z. Zeng, “Multiple Zeros of Nonlinear Systems,” *Mathematics of Computation*, **80**(276), 2011 pp. 2143–2168. www.ams.org/journals/mcom/2011-80-276/S0025-5718-2011-02462-2/S0025-5718-2011-02462-2.pdf.

[6] S. S. Abhyankar, *Algebraic Geometry for Scientists and Engineers*, Providence, RI: AMS, 1990.

[7] E. W. Weisstein. “Harnack’s Theorems” from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/HarnacksTheorems.html.

[8] E. W. Weisstein. “Hilbert’s Problems” from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/HilbertsProblems.html.

[9] E. W. Weisstein. “Descartes’ Sign Rule” from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/DescartesSignRule.html.

B. H. Dayton, “A Wolfram Language Approach to Real Numerical Algebraic Plane Curves,” *The Mathematica Journal*, 2018. dx.doi.org/doi:10.3888/tmj.20-7.

About the Author

Barry H. Dayton is Professor Emeritus at Northeastern Illinois University, where he taught for 33 years. His Ph.D. was in the field of algebraic topology, but he has done research in a variety of fields, including algebraic geometry and numerical algebraic geometry.

Barry H. Dayton
Department of Mathematics
Northeastern Illinois University
Chicago, IL 60625-4699
barry@barryhdayton.us
barryhdayton.space