Surface Story

Part III

Barry H Dayton
barryhdayton.space

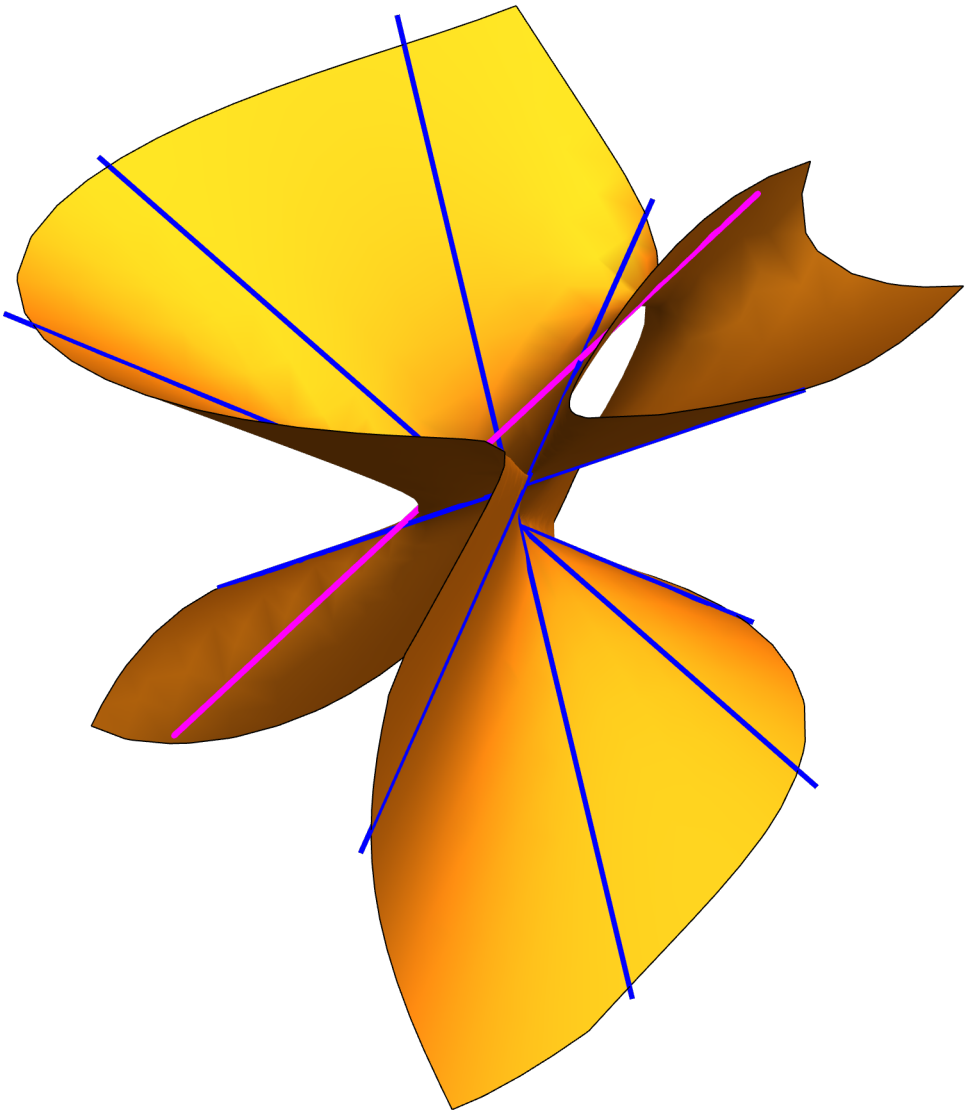


Table of Contents

5. Topology And Complex Curves

In this appendix we will change from algebraic geometry to topology, specifically the maps between surfaces may only be continuous, not necessarily projective linear transformations, eg. fractional linear transformations. Also, later in this appendix we will be interested in complex curves rather than just real ones. In general our global function naming convention will not follow our previous rules, however they will appear at the end of GlobalFunctionsNS.nb The reader interested in this topic may not need to read the previous chapters although some reference to earlier chapters of this book will occur.

In section 5.1 using the chromatic number we show that the sphere is not topologically equivalent to the torus or saddle surface. On the other hand, in 5.2 and 5.3 we show that the projective hyperboloid and saddle surface are topologically equivalent to the to the torus by giving explicit invertible continuous functions, known as homeomorphisms. In sections 5.4 and 5.5 we redo some material in my Plane Curve Book on normal forms correcting mistakes and extending them to the complex domain. And finally in sections 5.6 and 5.7 we give explicit descriptions of the known topological structure of the complex solution spaces of smooth plane conic and cubic curves.

This stand alone version of Chapter 5 uses some information from my *Plane Curve Book* and *Surface Story version 1* but can be read separately. The Mathematica **GlobalFunctionsNS.nb** notebook for my *Surface Story* has been updated to include the main functions in this appendix for those who want to experiment with my code.

5.1 Topology of Surfaces -- Chromatic Number

While I have viewed Algebraic Geometry as the study of projective linear transformations Topology is the study of continuous functions. In this book topological spaces will be subspaces of projective real or complex spaces. The main question is now *When are two surfaces topologically equivalent.* If one can find a bi-continuous invertible function between the surfaces, they are topologically equivalent. If we work in the affine domain the invertible functions should be extend to the projective closure of the affine surface.

But if we can't think of an obvious equivalence how can we prove that there is none? Topologists then depend on surface invariants preserved by topological equivalence. Often they rely on homology or homotopy groups to get these invariants. I don't want to go into this subject in this book. One visual invariant is easy to explain and see, even if it not necessarily easy to prove, is the *chromatic number of a surface.*

These derive from the famous *Four Color Theorem* which is normally stated on the affine plane. It derives from the number of colors a cartographer needs in coloring a map where geographic or political regions are separated by continuous curves or curve segments. These concept extends not just to the plane but any topological, two dimensional, surface. The number of colors necessary is called the

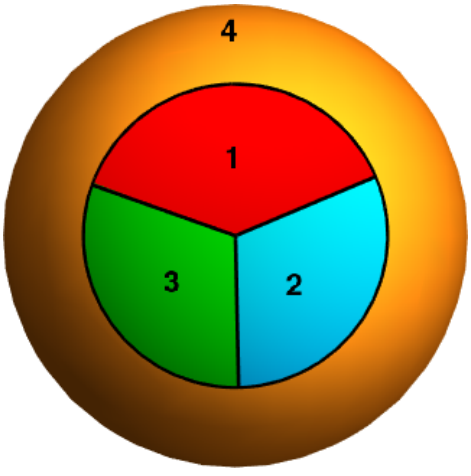
chromatic number .

For the plane or sphere the chromatic number is 4, this is known as the Four Color Theorem . This is not trivial, the problem goes back to the 1850' s when DeMorgan proposed this as a problem from cartography to the mathematical community . In 1879 Alfred Kemp proposed a proof which later was seen to have a flaw . However in 1890 Percy Heawood showed his method did show that at most 5 colors are needed for the plane, he, Heawood, also gave a formula for the number of colors needed for other surfaces depending on their genus which is correct except for the Klein bottle but the proof did not cover the plane and sphere.

But going from 5 colors to 4 is difficult, no one has ever displayed a map on the plane or sphere which can not be colored with 4 colors. But no one has ever come up with a theoretical reason why one couldn't do it. What mathematicians have done is to show that there are only finitely many configura - tions that could not have a 4 coloring so if people could color all of these with 4 colors then we would know 4 colors are sufficient. But there are a large number of of these configurations, they have many regions and are hard to enumerate. So it is impractical to color all these manually. In 1976 Kenneth Appel and Wolfgang Haken used a computer to attempt to color all of these and claimed the 4 color problem was solved. Your author was at a large math conference at the time and many mathemati - cians were skeptical, to them a computer calculation was not a proof. But many proofs do divide the problem into cases, just in this case there were thousands but, in my opinion, that should not affect the validity of the proof. But others felt the use of computer was fine but the program need to be carefully validated. In fact, it found to have errors which were introduced by the programmers, not the com - puter. Since then several more complicated computer methods have been used and most mathemati - cians are now satisfied that 4 colors are sufficient. I mention this example because it is the first exam - ple of the computer being an integral part of a proof.

Earlier there were some who claimed that even 3 colors are sufficient. But it is easy to show that is not correct, one needs only to show one partition that can not be colored by 3 colors. An example on the sphere is

Out[ ]=

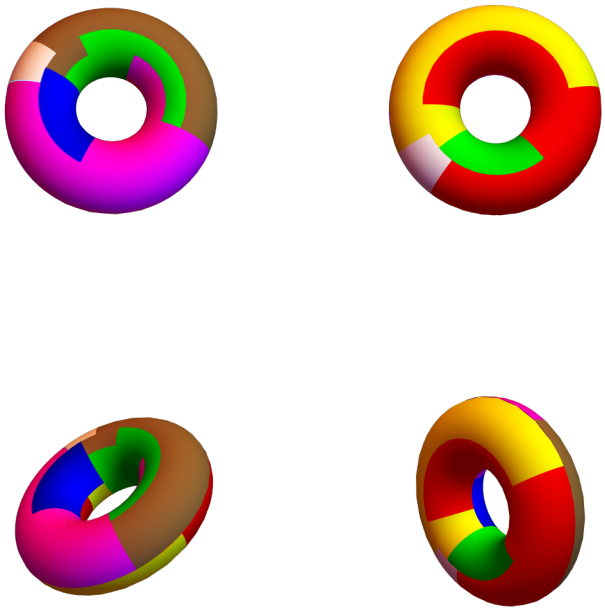


Region 4 is the outside of the circle containing 1,2,3. In this example each region meets each other region in an arc. Therefore each region must have a different color. This shows 4 colors are necessary.

The chromatic number of the sphere is then 4.

The point of this discussion is that if a surface is partitioned in *n* colors then any surface homeomorphic (topologically equivalent) also has a partition with *n* colors. Any homeomorphism will preserve the partition. So to show two surfaces are NOT homeomorphic it is enough to show they have different chromatic numbers. I use this to show that the sphere is not homeomorphic to a torus. The following partition shows the torus has chromatic number at least 7, it actually is 7 exactly but we don't need that fact.

Out[ ]=

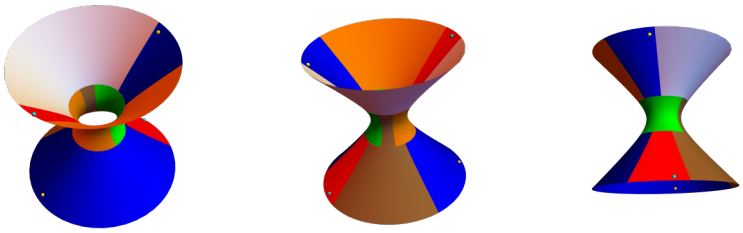


The 7 colors are white, maroon, blue, green, brown, red and yellow. These different views show each region meets each other region so we need at least 7 colors. We have proved the sphere is not topologically equivalent to a torus! FYI this graphic was drawn by Mathematica using a parameterization of the torus given in the next section.

In sections 5.2, 5.3 below I will show that projective hyperboloids are topologically equivalent to a torus. So we should be able to find a 7 coloring of these surfaces with each color touching each other color. Unfortunately because , unlike the torus, these are not affine surfaces it gets messy and I do not know of a nice example. But I do have nice 6 colorings of these spaces which show their chromatic number is at least 6, in particular they are not topologically equivalent to spheres by the 5 color theorem for the sphere which, as we saw, is considerably easier than the 4 color theorem.

For the standard hyperboloid $x^2 + y^2 - z^2 = 1$ here are some graphics.

Out[ ]=



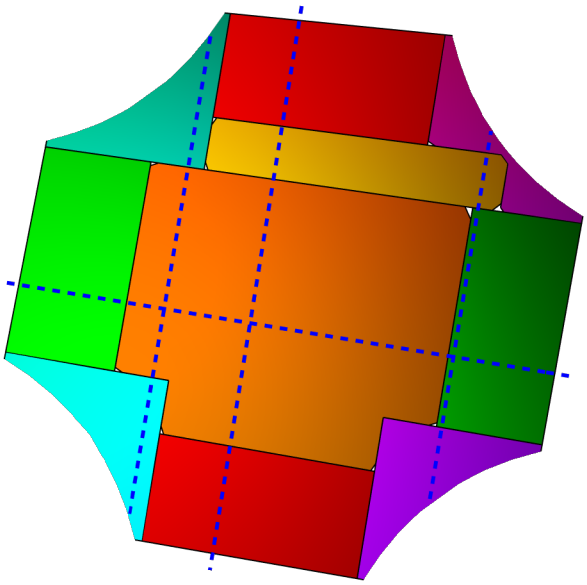
The small light colored balls show the red and blue regions are connected in projective space.

For the saddle surface $z = x\,y$ we have

Out[ ]= SSGraphic SSGraphic

Here it is less obvious that the red, green, cyan and magenta regions are connected projectively. The following graphic showing also 4 lines on the surface help.

Out[ ]=



I claim that the infinite points of these lines, which must lie in the projective surface, are arbitrarily near both affine components of these regions. For example the red region is given by $-2 \le y \le 2$ and $x \le -3$ or $3 \le x$.

In[ ]:=
```
lr = {4 - 8.420822433864302` t, -0.7`, -2.8` + 5.8945757037050095` t};
```

The infinite point is {–8.43082 , 0, 5.89458 , 0} from Section 1.10 .2

However if we consider the point in the lower component

In[ ]:=
```
lrp = lr /. t → 100
```

Out[ ]=
```
{-838.082 , -0.7 , 586.658}
```

which is equivalent to the projective point

```
In[ ]:= Append[lrp, 1] / 838.082 * 8.43082
Out[ ]= {-8.43082, -0.00704176, 5.90158, 0.0100597}
```

Likewise in the upper component

```
In[ ]:= lrn = lr /. t → -100
        Append[lrn, 1] / 846.082 * -8.42082
Out[ ]= {846.082, -0.7, -592.258}
Out[ ]= {-8.42082, 0.00696691, 5.89458, -0.00995272}
```

Both of these projective points are very close to the infinite point of this line as claimed .

5.2 Global Functions for Chapter 5

In this section we give some global functions that we will need in the sequel. These give continuous functions but are not, in general, algebraic. The reader is not expected to read this section now but these functions are needed in the sequel and are given here so as to not interrupt the story later with the definitions. These functions will be available to users of **Mathematica** in the **GlobalFunctionsN ⋱ S.nb** notebook which will be updated when this chapter is posted.

Hyperbola

Our standard hyperbola is

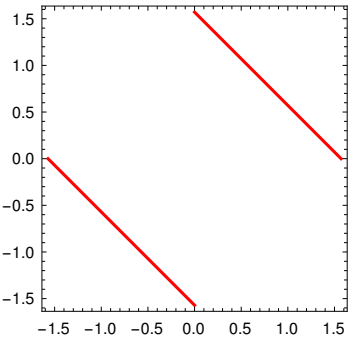
```
In[ ]:= hyp = x^2 + y^2 - z^2 - 1;
```

We now give a parameterization of this hyperbola on the square $-\pi/2 \le u, v \le \pi/2$

```
In[ ]:= HparSpace :=
         ContourPlot[{Tan[u] Tan[v] == 1}, {u, -Pi / 2, Pi / 2}, {v, -Pi / 2, Pi / 2}, ContourStyle → Red]
```

```
In[ ]:= Show[HparSpace, ImageSize → Small]
```

```
Out[ ]=
```



The red line will map to maps to the infinite line of hyperbola. This red line is given by x+y=Pi/2 or x+y=-Pi/2. One reason for this somewhat non-standard parameterization of the hyperboloid is that it takes

the square $-\pi/2 \le u, v \le \pi/2$ which is compact, to the entire projective hyperboloid which is also compact.

The parameterization is

```
In[ ]:= Hyp := {(1 + Tan[u] Tan[v])/(-1 + Tan[u] Tan[v]), (Tan[u] - Tan[v])/(-1 + Tan[u] Tan[v]), (Tan[u] + Tan[v])/(-1 + Tan[u] Tan[v])}
```

Usage: for point in HparSpace $-\text{Pi}/2 \le u, v \le \text{Pi}/2$ not on infinite red line the following places a point on the standard hyperbola

```
In[ ]:= hp = Hyp /. Thread[{u, v} → {.3, -.8}]
        hyp /. Thread[{x, y, z} → hp]
Out[ ]= {-0.51687, -1.01553, 0.546302}
Out[ ]= 1.11022 × 10^-16
```

An important feature of this parameterization is that we have an inverse function on the interior. If the point p is not on the hyperbola to a sufficient tolerance an error message will be produced.

```
In[ ]:= InvHyp[p_] := Module[{solab, a, b, c, d, ab, a1, b1, t0},
          solab = {{a → (-1 - Sqrt[1 - c^2 + d^2])/(c - d), b → (c/(c-d) - d/(c-d) + c Sqrt[1-c^2+d^2]/(c-d) - d Sqrt[1-c^2+d^2]/(c-d))/(c + d)},
            {a → (-1 + Sqrt[1 - c^2 + d^2])/(c - d), b → (c/(c-d) - d/(c-d) - c Sqrt[1-c^2+d^2]/(c-d) + d Sqrt[1-c^2+d^2]/(c-d))/(c + d)}};
          ab = solab[[1]] /. {c → p[[2]], d → p[[3]]};
          {a1, b1} = {a, b} /. ab;
          If[Norm[(Hyp /. {u → ArcTan[a1], v → ArcTan[b1]}) - p] < .0005,
           Return[{ArcTan[a1], ArcTan[b1]}]];
          ab = solab[[2]] /. {c → p[[2]], d → p[[3]]};
          {a1, b1} = {a, b} /. ab;
          If[Norm[(Hyp /. {u → ArcTan[a1], v → ArcTan[b1]}) - p] < .0005,
           Return[{ArcTan[a1], ArcTan[b1]}]];
          Echo[p, "Possible Numerical Error"];
          {}]
```

```
In[ ]:= InvHyp[hp]
Out[ ]= {0.3, -0.8}
```

```
In[ ]:= hp = {0, 1, 0}
        pq = InvHyp[hp]
        Hyp /. Thread[{u, v} → pq]
```

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

$$\text{Out[ ]= } \left\{-\frac{\pi}{4}, \frac{\pi}{4}\right\}$$

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

The other important thing is that for every *u* Hyp takes {u,-v} to the same point as {u,v}, and also takes {-u,v} to {u,v} for every u,v. Because zero denominators are involved we must use limits for this evaluation.

```
In[ ]:= rr = RandomReal[{-Pi / 2, Pi / 2}]
```

```
Out[ ]= 0.981721
```

```
In[ ]:= Limit[Hyp /. Thread[{u, v} → {rr, t}], t → -Pi / 2]
        Limit[Hyp /. Thread[{u, v} → {rr, t}], t → Pi / 2]
```

```
Out[ ]= {1., -0.668218, 0.668218}
```

```
Out[ ]= {1., -0.668218, 0.668218}
```

Likewise

```
In[ ]:= Limit[Hyp /. Thread[{u, v} → {t, rr}], t → -Pi / 2]
        Limit[Hyp /. Thread[{u, v} → {t, rr}], t → Pi / 2]
```

```
Out[ ]= {1., 0.668218, 0.668218}
```

```
Out[ ]= {1., 0.668218, 0.668218}
```

Already the topologists know the image of this parameterization will give topologically a torus so this invertible parameterization gives a proof that the projective hyperboloid is a topological torus, we will give a direct proof shortly.

Torus

We use this equation for our standard torus

$$\text{In[ ]:= toruspipi2 := } \frac{9\pi^4}{16} - \frac{5\pi^2 x^2}{2} + x^4 - \frac{5\pi^2 y^2}{2} + 2x^2y^2 + y^4 + \frac{3\pi^2 z^2}{2} + 2x^2z^2 + 2y^2z^2 + z^4$$

```
In[ ]:= TGraphic := ContourPlot3D[toruspipi2 == 0, {x, -5, 5}, {y, -5, 5}, {z, -5, 5},
          ContourStyle → Opacity[1], Mesh → None, Boxed → False, Axes → False]
```

In[ ]:=
```
Show[TGraphic , ImageSize → Tiny]
```

Out[ ]=

A parameterization is given on the entire square $-\pi \le r, t \le \pi$, t is an angle about {0,0,0} while r is an angle about the center circle of radius π.

In[ ]:=
```
Tpar := { {(π/2 - r) Cos[t], (π/2 - r) Sin[t], Sqrt[π^2/4 - (-π/2 - r)^2]}   r ≤ 0
        { {(π/2 + r) Cos[t], (π/2 + r) Sin[t], -Sqrt[π^2/4 - (-π/2 + r)^2]}  r > 0
        { 0                                                                  True
```

Usage

In[ ]:=
```
par = {2 Pi / 3, -Pi / 4}
tp = Tpar /. Thread[{r, t} → par]
N[tp]
```

Out[ ]= $\left\{\frac{2\pi}{3}, -\frac{\pi}{4}\right\}$

Out[ ]= $\left\{\frac{7\pi}{6\sqrt{2}}, -\frac{7\pi}{6\sqrt{2}}, -\frac{\sqrt{2}\,\pi}{3}\right\}$

Out[ ]= {2.59168, -2.59168, -1.48096}

In[ ]:=
```
toruspipi2 /. Thread[{x, y, z} → N[tp]]
```

Out[ ]= 1.42109×10^{-14}

Again an important property of this parameterization is we have an inverse

In[ ]:=
```
tangle[p_] := With[{a = VectorAngle[p, {1, 0}]}, If[N[{Cos[a], Sin[a]}] == p / Norm[p], a, -a]]
```

In[ ]:=
```
InvTor[{x_, y_, z_}] := Piecewise[{{{Pi / 2 - Norm[{x, y}], tangle[{x, y}]}, z ≥ 0},
    {{-Pi / 2 + Norm[{x, y}], tangle[{x, y}]}, z < 0 }}]
```

In[ ]:=
```
InvTor[tp]
```

Out[ ]= $\left\{\frac{2\pi}{3}, -\frac{\pi}{4}\right\}$

These parameterizations with similar (factor of 2) domains show directly that the standard hyperbola and standard torus are topologically equivalent, that is, homeomorphic. In fact we can map the torus to the hyperbola and hyperbola to torus directly

```
In[ ]:= T2H[{x_, y_, z_}] := Hyp /. Thread[{u, v} → .5 InvTor[{x, y, z}]]

In[ ]:= H2T[{x_, y_, z_}] := Tpar /. Thread[{r, t} → 2 InvHyp[{x, y, z}]]
```

Recall

```
In[ ]:= N[tp]
Out[ ]= {2.59168 , -2.59168 , -1.48096}

In[ ]:= hp1 = T2H[tp]
Out[ ]= {-0.164525 , -1.24969 , -0.767327}

In[ ]:= hyp /. Thread[{x, y, z} → hp1]
Out[ ]= -5.55112 × 10^-16

In[ ]:= H2T[hp1]
        H2T[hp1] - N[tp]
Out[ ]= {2.59168 , -2.59168 , -1.48096}
Out[ ]= {2.22045 × 10^-15, 4.44089 × 10^-15, -4.44089 × 10^-16}
```

Saddle Surface

Now consider the saddle surface z= x y. In Chapter 2 of the Surface Story it is shown that the saddle surface is algebraically equivalent to the standard hyperbola. Therefore we can use our parameteriza - tion of the hyperboloid above to give parameterization is given on $-\pi/2 \le s, t \le \pi/2$ by

```
In[ ]:= Ht := {{1.421753448878254` , 2.4001312247824407` ,
    -1.4217534488782626` , -2.4001312247824362` },
   {-1.3682399203220887` , 0.05585142943475707` , 0.7762628086454613` ,
    1.1281027939185155` }, {-2.550704470740892` , -1.499868080914514` ,
    0.08976727903245796` , 2.957640849193627` }, {1.1547005383792515` ,
    0.5773502691896257` , -1.1547005383792515` , -0.5773502691896266` }}

In[ ]:= Ht // MatrixForm
Out[ ]//MatrixForm=
```

$$\begin{pmatrix} 1.42175 & 2.40013 & -1.42175 & -2.40013 \\ -1.36824 & 0.0558514 & 0.776263 & 1.1281 \\ -2.5507 & -1.49987 & 0.0897673 & 2.95764 \\ 1.1547 & 0.57735 & -1.1547 & -0.57735 \end{pmatrix}$$

```
In[ ]:= SSpar[{s_, t_}] := Simplify[TransformationFunction [Ht][Hyp /. {u → s, v → t}]]
```

This is complicated

In[ ]:=
```
SSpar[{s, t}]
```

Out[ ]=
$$\left\{\frac{6.6197 + \text{Tan}[s](1.6946 - 1.6946\ \text{Tan}[t]) - 6.6197\ \text{Tan}[t]}{3. - 3.\ \text{Tan}[t] + \text{Tan}[s](-1. + 1.\ \text{Tan}[t])},\right.$$
$$\frac{-4.32379 + \text{Tan}[s](1.44126 - 0.41593\ \text{Tan}[t]) + 1.24779\ \text{Tan}[t]}{3. - 3.\ \text{Tan}[t] + \text{Tan}[s](-1. + 1.\ \text{Tan}[t])},$$
$$\left.\frac{-9.54073 + \text{Tan}[s](-2.44237 + 0.704834\ \text{Tan}[t]) + 2.75333\ \text{Tan}[t]}{3. - 3.\ \text{Tan}[t] + \text{Tan}[s](-1. + 1.\ \text{Tan}[t])}\right\}$$

The infinite part is $(x - \text{ssa})(y - \text{ssb}) = 0$ where

In[ ]:=
```
ssa = N[ArcTan[3]]
ssb = N[ArcTan[1]]
```

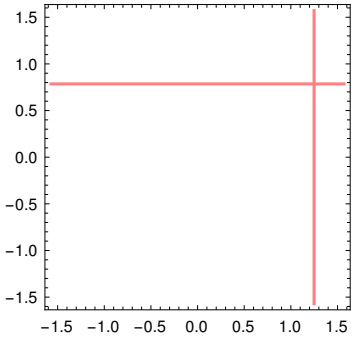
Out[ ]=
```
1.24905
```

Out[ ]=
```
0.785398
```

Thus the parameter space for our saddle surface looks like this where the red lines map to the infinite curve of the saddle surface .

In[ ]:=
```
SSparSpace = ContourPlot[{x == ssa, y == ssb}, {x, -Pi / 2, Pi / 2},
  {y, -Pi / 2, Pi / 2}, ContourStyle → Pink, ImageSize → Small]
```

Out[ ]=



Again we can find and inverse using the inverse we already have

In[ ]:=
```
InvSS[{x_, y_, z_}] := InvHyp[TransformationFunction [Inverse[Ht]][{x, y, z}]]
```

Warning : this only takes actual points given numerically . Does not give a formula. It is useful to note that we can easily find points on the saddle surface by picking real numbers a,b and the point is then $\{a, b, a\,b\}$.

In[ ]:=
```
inp = InvSS[{2, 3, 6}]
```

Out[ ]=
```
{-0.166184 , 0.915161}
```

In[ ]:=
```
SSpar[inp]
```

Out[ ]=
```
{2., 3., 6.}
```

Finally we get, for actual points, a map from the Torus to the Saddle Surface showing that these are also homeomorphic.

```
In[ ]:= T2SS[{x_, y_, z_}] := SSpar[.5 InvTor[{x, y, z}]]
```

The inverse is

```
In[ ]:= SS2T[{x_, y_, z_}] := Tpar /. Thread[{r, t} → 2 InvSS[{x, y, z}]]
```

```
In[ ]:= t236 = SS2T[{2, 3, 6}]
Out[ ]= {-0.488395, 1.83943, 0.966279}
```

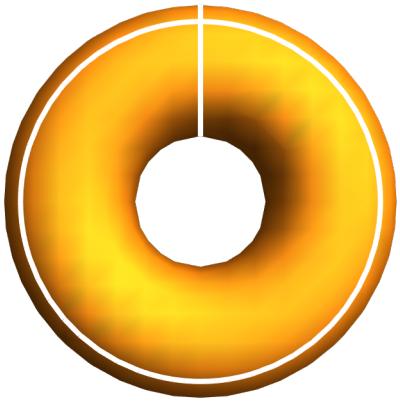
```
In[ ]:= toruspipi2 /. Thread[{x, y, z} → t236]
Out[ ]= 0.
```

```
In[ ]:= T2SS[t236]
Out[ ]= {2., 3., 6.}
```

Note the infinite curve on the saddle surface comes from

```
In[ ]:= TSSGraphic =
 Show[TGraphic, ParametricPlot3D [Tpar /. {r → 2 ssa}, {t, -Pi, Pi}, PlotStyle → White],
  ParametricPlot3D [Tpar /. {t → 2 ssb}, {r, -Pi, Pi}, PlotStyle → White],
  ViewPoint → Below]
```

Out[ ]=



5.3 Topology of Hyperboloids

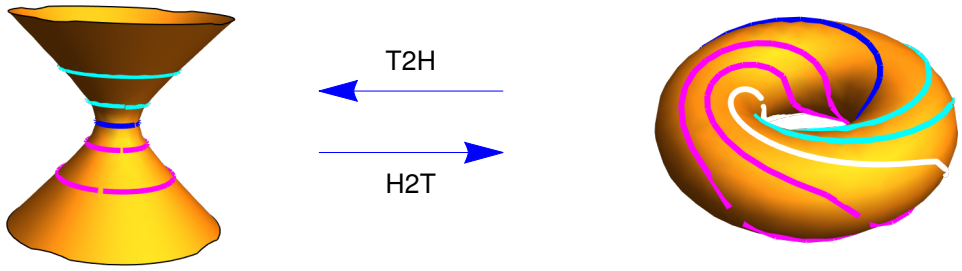
In this section I show that all projective hyperboloids are topologically equivalent to the torus. In 5.1 we showed the torus is not equivalent to the sphere so smooth projective quadric surfaces are not the

same topologically.

I suggested that the reader not read Section 5.2 carefully, but if you did then you may not wish to read this section which is a more readable but less computational version of 5.2.

I f first show the hyperboloid is topologically a torus. Here we have explicit invertible continuous functions defined in 5.2 **T2H** and **H2T**. See examples in 5.2.

In[ ]:=



Then since all hyperboloids are equivalent by projective linear transformations which are continuous they are all topologically equivalent to the torus.

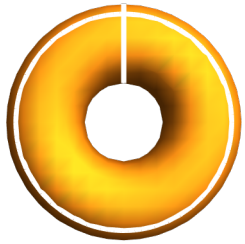
For the saddle surface $z = x\,y$ in section 5.2 we introduced a parameterization of the saddle surface **SSPar** on the square $-(\pi/2) \le x \le \pi/2$ and $-(\pi/2) \le y \le \pi/2$. We identified 2 lines in this square which map to the infinite curve x = **ssa** and y = **ssb** where **ssa** is approximately 1.24905 and **ssb** is approximately 0.785398. This parameterization has an inverse **InvSS.**

On the other hand we gave the parameterization of the Torus by `Tpar` on the square $-\pi \le x \le \pi$ and $-\pi \le y \le \pi$ with inverse **InvTor**. In this case there are no infinite points to worry about.

We can go back and forth between these parameter spaces by multiplication or division by 2. Then we can map the torus to the hyperboloid by sending a point on the torus by applying **InvTor**, dividing by 2 and applying **SSpar**. We called this **T2SS** with inverse SS2T. The points of the torus that land in the infinite part of the saddle surface come from the points of the torus parameter space with x = 2 ssa and y =2 ssb. So the picture of the points on the torus going to affine points of the saddle surface looks like this with the white curves removed. That is, this picture shows the actual domain of T2SS as defined in 5.2.

In[ ]:=
```
TSSGraphic =
 Show[TGraphic , ParametricPlot3D [Tpar /. {r → 2 ssa}, {t, -Pi , Pi}, PlotStyle → White],
  ParametricPlot3D [Tpar /. {t → 2 ssb}, {r, -Pi , Pi}, PlotStyle → White],
  ViewPoint → Below , ImageSize → Small]
```

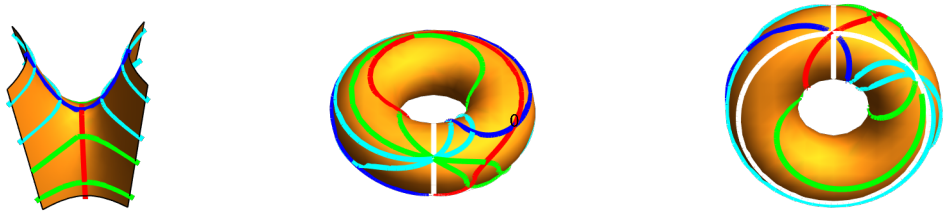
Out[ ]=



I claim that that the function T2SS can be extended on the whole torus as a function to the projective saddle surface.

Here is a graphic showing the map between a saddle surface and a torus. The curves correspond by color. The two graphics on the right of tori show different views of the same torus.

Out[ ]=



It is easy to find parametric functions giving the horizontal colored circles on the hyperboloid. The mapping sends them to the more complicated curves on the torus. The white curve on the torus is the image of the infinite curve of the hyperboloid. I leave the details for the reader.

5.3.2 A topological Construction

We leave our study of topology with one more well known result.

Theorem : *Suppose we have a continuous map from a plane square to a topological space $\mathcal{T}$ which sends points on opposite side of a plane square to the same point of $\mathcal{T}$ .Then this map factors through the torus. In particular f this map is 1-1 on the interior of the square and onto then $\mathcal{T}$ is homeomorphic to the torus.*

To be more specific we will use the square $\pi/2 \leq x,\ y \leq \pi/2$. If our map is called f then we are assum -

ing $f[\{x, -\pi/2\}] = f[\{x, \pi/2\}]$ for all x and $f[\{-\pi/2, y\}] = f[\{\pi/2, y\}]$ for all y.

A reference for this theorem is the book *Principles of Topology* by Fred H. Croom, Dover Publications, 1989. This theorem is essentially his Example 7.4.2.

Croom's argument uses the idea of a quotient space. For any equivalence relation ~ on a topological space $\mathcal{X}$, Croom defines a new topological space $\mathcal{X}/\sim$ whose elements are the equivalence classes of ~. From set theory there is an onto projection $p:\mathcal{X} \longrightarrow \mathcal{X}/\sim$ sending each element to its class. One defines a set in $\mathcal{X}/\sim$ to be open if its inverse image in $\mathcal{X}$ is open, making p continuous. Moreover, as Croom notices, any function $f:\mathcal{X} \longrightarrow \mathcal{Y}$ factors through p, that is there is a function $f^*:\mathcal{X}/\sim \longrightarrow \mathcal{Y}$ with $f = f^* \circ p$, our definition of the topology on $\mathcal{X}/\sim$ makes f^* continuous. Moreover Croom's Theorem 7.16 says that if f is onto then f^* is a homeomorphism.

Applying this to our map **Hyp** from the square to the torus then the relation that Hyp sends points {x, -$\pi/2$} and $\{x, \pi/2\}$ to the same point on the torus as well as with $\{-\pi/2, y\}$ and $\{\pi/2, y\}$ so defining the relation ~ by {x, -$\pi/2$} ~$\{x, \pi/2\}$, $\{-\pi/2, y\}$~$\{\pi/2, y\}$ with every interior point related only to itself shows that quotient space is homeomorphic to the torus. But the hypotheses of our theorem above says our continuous function also has the same quotient space so its image is also homeomorphic to the torus.

An easy modification of this is the classical result

Corollary : Suppose f is a continuous function on the real plane, or complex numbers, which is periodic in in each variable with period 2π, and let {a,b}~{c,d} if and only if $a = b + 2j\pi$ and $c = 2k\pi$ for integers j, k. Then the quotient space $\mathbb{R}^2/\sim$ is homeomorphic to a torus and thus f can be viewed as mapping the plane to a torus.

A useful tool here is to find a canonical point in our square equivalent, in the relation of the corollary, to a point in the original square $-\pi \le x, y \le \pi$.

Later we will encounter doubly functions with periods 2π in both directions, to reduce to our parame - ter square $-\pi \le u, v \le \pi$ we use the following global function which is initiated here and our Global Functions notebook.

In[ ]:=
```
reduce2pipi[{a_, b_}] := Module[{c, d},
  c = a; d = b;
  While[c < -Pi, c = c + 2 Pi];
  While[c > Pi, c = c - 2 Pi];
  While[d < -Pi, d = d + 2 Pi];
  While[d > Pi, d = d - 2 Pi];
  N[{c, d}]]
```

For example let

In[ ]:=
```
Clear[a, b]
a = {9.35, -17.12};
b = reduce2pipi[a]
```

Out[ ]=
```
{3.06681, 1.72956}
```

In[ ]:= **a – b**

Out[ ]= {6.28319 , –18.8496}

where

In[ ]:= **(a – b) / 2 / π**

Out[ ]= {1. , –3.}

5.4 Normal Forms in Complex Algebraic Geometry

5.4.1 The cTransform

We now temporarily change our focus back to algebraic geometry, but unlike most of the rest of my books we will look at complex algebraic geometry. We will take a new look at sections 7.3 and 7.5 in my *Plane Curve Book*.

Our first tool is the cTransform. Here is the updated version with a new name for conics as well as some other renamed subroutines. The important thing is that they all work in the complex case.

```
cTransform2[f_, p_, x_, y_] := Module[{fh, ph, nh, t, cs, A, d},
    d = tDegMD[f, {x, y}];
    fh = Expand[t^d * (f /. Thread[{x, y} → {x / t, y / t}])];
    ph = If[Length[p] == 2, N[Append[p, 1]], N[p]];
    ph = ph / Norm[ph];
    nh = {D[fh, x], D[fh, y], D[fh, t]} /. Thread[{x, y, t} → ph];
    nh = nh / Norm[nh];
    cs = Cross[nh, ph];
    cs = cs / Norm[cs];
    A = {cs, ph, nh}];
```

Here we have a curve *f* in 2 variables and a point *p* on the curve. This returns a 3×3 matrix such that the Transformation Function takes *p* to the infinite point {0,1,0} and the tangent line to the infinite tangent line at that point.

```
infiniteConicPoints2D[f_, x_, y_] := Module[{mf, rrl, sv}, mf = FromCoefficientRules[
      Select[CoefficientRules[f, {x, y}], Total[#[[1]]] == 2 &], {x, y}];
    rrl = RandomReal[{-5, 5}, 3].{x, y, 1};
    sv = NSolveValues[mf == 0 && RandomReal[{-1, 1}, 3].{x, y, 1} == 0, {x, y}, Complexes];
    {Append[sv[[1]], 0], Append[sv[[2]], 0]}];
```

In[ ]:=
```
flt[p_, A_] := TransformationFunction[A][p]
```

```
FLTC2[f_, A_, x_, y_] := Module[{fh, gh, AI, B, d, z},
    fh = Expand[z^2 (f /. {x → x/z, y → y/z})];
    AI = Inverse[A];
    B = AI.{x, y, z};
    gh = N[Expand[fh /. Thread[{x, y, z} → B]]];
    Chop[gh /. {z → 1}, 1*^-10]
  ];
```

In Section 3.1 of my *Plane Curve Book* there is a global function **tLine** which finds the tangent line to a curve at a point *p* on curve *f*. We change the name for convenience and add it to our global functions.

```
tangentLine2D[f_, p_, x_, y_] := line2D[p, {D[f, y], -D[f, x], 0} /. Thread[{x, y} → p], x, y]
```

We now note that this works for complex conics. Here is an example

```
Clear[f]
f = (.2 + i) x^2 - (3 + 8.32 i) x y + (4 - 2 i) y^2 + i x - 3 y + 4 - 5 i
```

$$(4 - 5\,i) + i\,x + (0.2 + 1.\,i)\,x^2 - 3\,y - (3. + 8.32\,i)\,x\,y + (4 - 2\,i)\,y^2$$

A point on this conic is

```
pf = {1, -0.007577844391223804` - 0.42500676635061163` i}
f /. Thread[{x, y} → pf]
```

$$\{1, -0.00757784 - 0.425007\,i\}$$

$$0. + 4.44089 \times 10^{-16}\,i$$

```
Cf = cTransform2D[f, pf, x, y]
```

$$\{\{-0.177415 - 0.334953\,i,\ 0.447326 - 0.484563\,i,\ 0.386747 + 0.521398\,i\},$$
$$\{0.677179,\ -0.00513155 - 0.287805\,i,\ 0.677179\},$$
$$\{-0.16655 + 0.232069\,i,\ -0.415163 - 0.625353\,i,\ 0.429183 - 0.413255\,i\}\}$$

First note this sends point **pf** to {0,1,0}. We use the extended **flt** function

```
Chop[fltiMD[pf, Cf]]
```

$$\{0,\ 1.23208 + 0.00436189\,i,\ 0\}$$

which is equivalent to {0, 1, 0} in the projective plane by homogeniety.

Now the infinite line in projective 2-space is the line which contains {0,1,0} and {1,0,0}. So we can indirectly check that the original tangent line goes to this line.

```
tlf = tangentLine2D[f, pf, x, y]
```

$$(0.478563 - 1.02142\,i) - (0.137113 - 0.523119\,i)\,x - (1.15776 + 0.82404\,i)\,y$$

Here **tangentLine2D** is our new, less confusing name for **tline** in our plane curves global functions, this is in the current GlobalFunctionsNS.nb.

```
In[ ]:= tangentLine2D[f_, p_, x_, y_] := line2D[p, {D[f, y], -D[f, x], 0} /. Thread[{x, y} → p], x, y]
```

Our check is to show that the point {1,0,0} of the infinite line is in the image of a point of tlf via the **TransformationFunction** determined by **Cf.**

```
In[ ]:=
```

```
In[ ]:= jf = fltiMD[{1, 0, 0}, Inverse[Cf]]
Out[ ]= {-0.577223 - 0.0878878 i, -0.188995 - 0.998123 i}
```

```
In[ ]:= tlf /. Thread[{x, y} → jf]
Out[ ]= 4.44089 × 10^-16 - 6.66134 × 10^-16 i
```

which does the trick .

5.4.2 The normal form for a conic -- Real Example

The normal form of a conic will be the parabola $y = x^2$. Consider, for purposes of replication, the pseudo random integer conic

```
In[495]:= conic1 := 1 + 10 x - 2 x^2 - 2 y + 8 x y + y^2
```

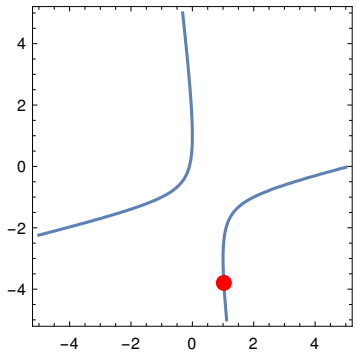
We can find a real point by using our plot to estimate a point we would like to use from this plot and finding the closest point actually on the conic. Thus we pick

```
In[496]:= c1p = {1.0259093563540405` , -3.7953394709624946` };
```

```
In[ ]:= conic1 /. Thread[{x, y} → c1p]
Out[ ]= 0.
```

```
In[ ]:= Show[ContourPlot[conic1 == 0, {x, -5, 5}, {y, -5, 5}, ImageSize → Small],
  Graphics[{Red, PointSize[.05], Point[c1p]}]]
```

```
Out[ ]=
```



We now apply the cTransform .

```
In[ ]:= Ac1 = cTransform2D[conic1, c1p, x, y]
Out[ ]= {{0.578058, 0.350419, 0.736923},
  {0.25289, -0.935565, 0.246504}, {-0.775819, -0.0438672, 0.629429}}
```

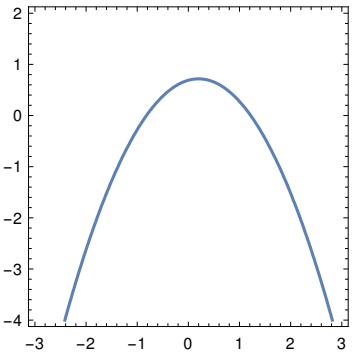
To see what we get we apply FLT.

In[ ]:=
```
f1 = FLT3D[{conic1}, Ac1, {x, y}][[1]]
```

Out[ ]=
$$-5.36143 - 2.14214\ x + 5.36143\ x^2 + 7.77378\ y$$

In[ ]:=
```
ContourPlot[f1 == 0, {x, -3, 3}, {y, -4, 2}, ImageSize → Small]
```

Out[ ]=



So we already have a conic with vertical axis We have to get rid of the linear part, move the vertex of this parabola to the origin and make the coefficients of *x*^2 and *y* negatives. Using the trick we learned in Chapter 2 rather than the method of Section 7.3 in the Plane Curves Book, a transform with matrix

In[ ]:=
```
S1 = {{1, 0, a}, {0, b, c}, {0, 0, 1}}
```

Out[ ]=
```
{{1, 0, {9.35, -17.12}}, {0, {3.06681, 1.72956}, c}, {0, 0, 1}}
```

should do the trick .

In[ ]:=
```
f2 = FLT3D[{f1}, S1, {x, y}][[1]]
```

MatrixRank : Argument {{1, 0, {9.35, −17.12}}, {0, {3.06681, 1.72956}, c}, {0, 0, 1}} at position 1 is not a non−empty rectangular matrix .

Inverse : Argument {{1, 0, {9.35, −17.12}}, {0, {3.06681, 1.72956}, c}, {0, 0, 1}} at position 1 is not a non−empty square matrix .

Out[ ]=
$$5.63164\ (\text{Inverse}[\{\{1, 0, \{9.35, -17.12\}\}, \{0, \{3.06681, 1.72956\}, c\}, \{0, 0, 1\}\}].\{x, y, 1\})^2$$

We see we want 5.36143 $b = -7.77378$ to make the coefficients of x^2 and y negatives so

In[ ]:=
```
bb = -7.7737816319831445` / 5.361426248281447`
```

Out[ ]=
```
-1.44995
```

In[ ]:=
```
f3 = f2 /. b → bb
```

Inverse : Argument {{1, 0, {9.35, −17.12}}, {0, −1.44995, c}, {0, 0, 1}} at position 1 is not a non−empty square matrix .

Out[ ]=
$$5.63164\ (\text{Inverse}[\{\{1, 0, \{9.35, -17.12\}\}, \{0, -1.44995, c\}, \{0, 0, 1\}\}].\{x, y, 1\})^2$$

Now to get rid of the linear part we need -2.14214 xc a x = 0

In[ ]:=
```
aa = -2.14214 / 10.7229
```

Out[ ]=
```
-0.199772
```

```
In[ ]:= f4 = f3 /. a → aa
Out[ ]= {5.63164 (0.199772 + 1. x)^2, 5.63164 (0. + 0.689681 c - 0.689681 y)^2, 5.63164}
```

And finally we can get rid of the constant part by

```
In[ ]:= cc = 5.575396993314872` /5.361426248281447`
Out[ ]= 1.03991

In[ ]:= f5 = f4 /. c → cc
Out[ ]= {5.63164 (0.199772 + 1. x)^2, 5.63164 (0.717205 - 0.689681 y)^2, 5.63164}
```

So we arrive at our conical form

```
In[ ]:= Chop[Expand[f5 / 5.361426248281447` ], 10 ^ -5]
Out[ ]= {0.0419205 + 0.419682 x + 1.0504 x^2, 0.540308 - 1.03915 y + 0.499633 y^2, 1.0504}
```

Note if

```
In[ ]:= S2 = S1 /. Thread[{a, b, c} → {aa, bb, cc}]
Out[ ]= {{1, 0, -0.199772}, {0, -1.44995 , 1.03991}, {0, 0, 1}}

In[ ]:= f6 = FLT3D[{conic1}, S2.Ac1, {x, y}][[1]]
Out[ ]= -7.08471 × 10^-6 x + 5.36143 x^2 - 5.36143 y
```

So if

```
In[ ]:= A1 = S2.Ac1
Out[ ]= {{0.733046 , 0.359182 , 0.61118},
         {-1.17346 , 1.3109 , 0.297132}, {-0.775819 , -0.0438672 , 0.629429}}

In[ ]:= Chop[Expand[FLT3D[{conic1}, A1, {x, y}][[1]] / 5.361426248281447` ], 1.*^-5]
Out[ ]= 1. x^2 - 1. y
```

showing that the Transformation Function with matrix Ac brings conic1 to canonical form, up to a constant multiple.

Following this example we can write down a general procedure. Note that this uses a random transformation so it will return a different transformation function each run. You should save the result with a unique name for future use rather than re-run this function when working with the same curve. This has a self check feature so the comment should give a normal form c x^2 -c y for some complex non-zero number. If this is not correct or there is an error message then re-run the function to get a different random transformation.

```
normalForm4Conic[ff_, pp_, x_, y_] := Module[{f, g, h, A, B, h0,
    h1, h2, h3, h4, h5, cfa, aa, bb, cc, cps, rm, p, S, S2, srm, a, b, c},
  rm = RandomReal[{-1, 1}, {3, 3}];
  f = FLTC2[ff, rm, x, y];
  p = flt[pp, rm];
  If[Abs[f /. Thread[{x, y} → p]] > 1.*^-11 , Echo["p not accurate point"];
   Abort[]];
  A = cTransform2[f, p, x, y];
  g = FLTC2[f, A, x, y];
  S = {{1, 0, a}, {0, b, c}, {0, 0, 1}};
  h = FLTC2[g, S, x, y];
  h1 = Expand[b h];
  bb = -Coefficient[h1, y] / Coefficient[h1, b x^2];
  h2 = h1 /. {b → bb};
  cfa = Coefficient[h2, x];
  aa = -cfa[[1]] / Coefficient[cfa, a];
  h3 = h2 /. {a → -cfa[[1]] / Coefficient[cfa, a]};
  cc := -h3[[1]] / Coefficient[h3, c];
  S2 = S /. Thread[{a, b, c} → {aa, bb, cc}];
  B = S2.A.rm;
  Echo[Chop[FLTC2[ff, B, x, y], 1.*^-8], "Normal Form is "];
  B]
```

Applying to our previous example

In[497]:=
```
normalForm4Conic[conic1, c1p, x, y]
```

» Normal Form is 3.10867 x^2 - 3.10867 y

Out[497]=
```
{{-0.636665, -0.453268, -1.06714},
 {-1.50554, 2.95103, 1.44589}, {-0.696565, -0.039386, 0.56513}}
```

Note rather than give normal form $y = x^2$ it gives a multiple of that. This is because FLT3D works only up to a constant.

If one runs a real conic but with a complex solution one will get a complex transformation.

In[499]:=
```
B = normalForm4Conic[x^2 + y^2 - 1, {Sqrt[2], -i}, x, y]
```

» Normal Form is (1.27771 - 1.92135 i) x^2 - (1.27771 - 1.92135 i) y

Out[499]=
```
{{-0.507268 - 0.383603 i, -0.681544 + 0.138805 i, 0.578581 - 0.139048 i},
 {0.317734 + 0.392038 i, -0.343997 - 3.13481 i, 0.382171 + 3.14761 i},
 {-0.118401 + 0.235166 i, 0.166288 + 0.0837225 i, 0.0837225 - 0.166288 i}}
```

In[501]:=
```
(y - x^2) /. Thread[{x, y} → fltMD[{.6, -.8}, B]]
```

Out[501]=
$$-1.06581 \times 10^{-14} - 1.06581 \times 10^{-14}\, i$$

5.4.6 A complex example

For our second example we start with a pseudo - random complex integer example . Here we leave off the linear part as it plays no essential role as complex conics are not divided into parabolas, hyperbolas and ellipses as real conics are.

In[502]:=
```
conic2 = -1 + (4 + i) x^2 - (6 - 5 i) x y - (3 - 2 i) y^2
```

Out[502]= $-1 + (4 + i)\, x^2 - (6 - 5\, i)\, x\, y - (3 - 2\, i)\, y^2$

In[503]:=
```
sol1 = NSolveValues[conic2 == 0, {x, y}]
```

NSolveValues : Infinite solution set has dimension at least 1. Returning intersection of solutions with

$$-\frac{92291\ x}{87992} - \frac{121001\ y}{175984} == 1.$$

Out[503]= {{1.5718 - 0.128716 i, -3.85212 + 0.19635 i}, {-0.822423 + 0.108657 i, -0.199829 - 0.165752 i}}

Mathematica Note: This is the same pseudo-random value I got earlier using Mathematica 12.3 instead of 13.1. In some ways this is comforting, but as we saw earlier in this chapter maybe not a good thing. So our random point is still

In[504]:=
```
p2 = sol1[[1]]
```

Out[504]= {1.5718 - 0.128716 i, -3.85212 + 0.19635 i}

Our algorithm **normalForm4Conic** works here.

In[505]:=
```
K = normalForm4Conic[conic2, p2, x, y]
```

» Normal Form is $(-7.58234 - 11.9463\ i)\, x^2 + (7.58234 + 11.9463\ i)\, y$

Out[505]= {{0.587288 - 0.87296 i, 0.278298 - 0.388537 i, 0.185011 - 0.103622 i},
{1.39576 - 5.06059 i, -0.826902 + 0.0268567 i, 1.11321 - 1.83181 i},
{0.214735 - 0.099931 i, 0.0834036 - 0.0438464 i, -0.0119869 - 0.000566177 i}}

We can use this to find many points on conic 2 simply by applying our transformation function to easily found points on the parabola $y = x^2$

In[506]:=
```
q1 = fltMD[{3, 9}, Inverse[K]]
```

Out[506]= {-0.260772 - 0.278993 i, 0.211966 - 0.318517 i}

In[507]:=
```
conic2 /. Thread[{x, y} → q1]
```

Out[507]= $0. - 4.66294 \times 10^{-15}\ i$

We can use complex numbers as well

In[508]:=
```
q2 = fltMD[{.357 - .218 i, (.357 - .218 i)^2}, Inverse[K]]
```

Out[508]= {-0.364464 - 0.0644493 i, 0.346904 - 0.0957222 i}

In[509]:=
```
conic2 /. Thread[{x, y} → q2]
```

Out[509]=
$$-3.33067 \times 10^{-15} - 3.27516 \times 10^{-15}\, i$$

5.5 Normal Form for Smooth Cubics

In[ ]:=
```
Clear[g1, g2, g3]
```

In section 7.5 of the Plane Curve Book I gave an algorithm for Weierstrass Normal form for a smooth plane cubic. Unfortunately I was thinking only of numerical cubics and did not worry about form of the coefficients. However in this Chapter we need the correct form $y\text{^}2 = 4\,x^3 - g_2\,x - g_3$ with coefficient of x^3 being 4.

Here is a correct function, it takes a smooth cubic *f* and inflection point **ipf** on *f* and it returns as a first component the normal form and the second component gives a transformation matrix taking a point on *f* to a point on the normal form. Note this differs from **weierstrassNormalForm2D** in my *Plane Curve Book* and earlier versions of *GlobalFunctions.nb*

In[ ]:=
```
weierstrassNormalForm [f_, ip_, x_, y_] :=
  Module[{B1, B2, B3, B4, B5, B, f1, f2, f3, f4, k, cy2},
   B1 = cTransform2D [f, ip, x, y];
   f1 = Chop[FLT2D[f, B1, x, y], 1.*^-9];
   cy2 = Coefficient [f1, y^2];
   If[Length[cy2] > 0, Print["Sorry, this example requires special handling"];
    Abort[]];
   k = Expand[Coefficient [f1, y] / cy2 / 2 + y];
   B2 = {{1, 0, 0}, {Coefficient [k, x], 1, k[[1]]}, {0, 0, 1}};
   f2 = FLT2D[f1, B2, x, y];
   B3 = homothety2D [CubeRoot [-(Coefficient [f2, x^3] / Coefficient [f2, y^2])], 1];
   f3 = FLT2D[f2, B3, x, y];
   f3 = -Expand[f3 / Coefficient [f3, y^2]];
   B4 = {{1, 0, Coefficient [f3, x^2] / 3}, {0, 1, 0}, {0, 0, 1}};
   B5 = {{1, 0, 0}, {0, 2, 0}, {0, 0, 1}};
   f4 = FLT2D[f3, B5.B4, x, y];
   B = B5.B4.B3.B2.B1;
   {Expand[4 f4], B}];
```

For example, let

In[ ]:=
```
f = y^2 - x^3 - x^2 + 2 x + 1 - 2 x y;
```

We find inflection points by **allInflectionPoints2D** in Section 70.1 of our *GlobalFunctionsNS.nb.*

In[ ]:=
```
aip = allInflectionPoints2D [f, x, y]
```

Out[ ]=
```
{{0., 1., 0.}, {1.73471, 4.33647}, {1.73471, -0.867047}}
```

```
In[ ]:= ip = aip[[3]]
Out[ ]= {1.73471 , -0.867047}

In[ ]:= {wfn, Afn} = weierstrassNormalForm [f, ip, x, y]
Out[ ]= {0.193765 - 1.86508 x + 4. x^3 - 1. y^2, {{0.152783 , -0.316429 , -0.539393},
          {1.54482 , -0.419326 , 1.32055}, {-0.593323 , -0.352319 , 0.723767}}}

In[ ]:= ip2 = aip[[2]]
Out[ ]= {1.73471 , 4.33647}
```

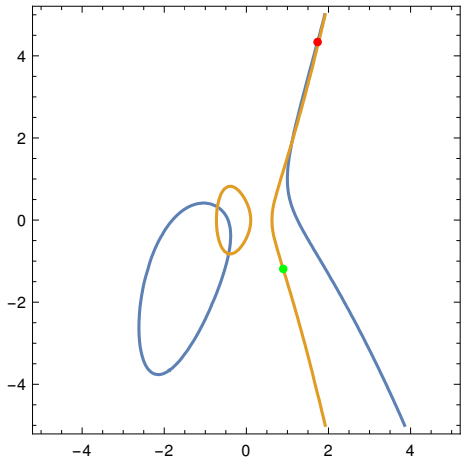
Note that the transformation function takes ip to a point on wfn

```
In[ ]:= iq = fltMD[ip2, Afn]
        wfn /. Thread[{x, y} → iq]
Out[ ]= {0.898131 , -1.19019}

Out[ ]= -8.48255 × 10^-12
```

Caution : remember the inflection point ip goes to an infinite point under this transformation.

```
In[ ]:= Show[ContourPlot[{f == 0, wfn == 0}, {x, -5, 5}, {y, -5, 5}],
          Graphics[{PointSize[.02], {Red, Point[ip2]}, {Green, Point[iq]}}]]
```

Out[ ]=



5.6 Topology of the Complex Solution Set for smooth plane Conics

In these last two subsections we will explore complex curves. I briefly mention at the end of Chapter 5 of my *Plane Curve Book* that complex solutions of curves form compact orientable surfaces. However the final pictures there are wrong and I can finally correct them. In these last sections we will show, in the case of smooth conic and cubic how to explicitly give invertible continuous maps from the complex solution set to a sphere or torus. In these sections we will think of the complex solution space is *up* while the surface is *down.*

5.6.1 The Parabola

My viewpoint in this note is that the complex solution space is *up* and we will map *down* to our target. So for each case we will define a *down* mapping and an inverse *up* mapping.

We first do the simple case of the parabola $y = x^2$. From our earlier results in this book we will see that using our projective linear transformations we will able to then get the general case.

This is quite simple for the affine part since every complex value for x gives a unique solution . So it is immediate that the affine complex solution space is simply the complex plane. But to get the correct infinite part we will instead use the equivalent affine space the paraboloid $z = x^2 + y^2$.

We need the **Mathematica** built in functions `ReIm` and `Abs`. The first changes a specific complex number $\alpha + \beta i$ to the real ordered pair $\{\alpha, \beta\}$. It may do some expansion:

```
In[ ]:= ReIm[(2 + 3 I) ^ 2]
```

```
Out[ ]= {-5, 12}
```

The second is just the absolute value $\sqrt{(\alpha^2 + \beta^2)}$.

```
In[ ]:= Abs[4 - 5 I]
```

```
Out[ ]= √41
```

So our up and down functions are defined simply by

```
In[ ]:= Pdown[{u_, v_}] := Append[ReIm[u], Abs[v]]
        Pup[{x_, y_, z_}] := {x + I y, (x + I y) ^ 2}
```

where u, v are any complex numbers and x, y, z are real numbers satisfying $z = x^2 + y^2$. To illustrate that these are inverse functions between the correct domains

```
In[ ]:= pu1 = RandomComplex [{-10 - I, 10 + I}];
        pu = {pu1, pu1^2}
        pd = Pdown[pu]
        (x ^ 2 + y ^ 2 - z) /. Thread[{x, y, z} → pd]
        Pup[pd]
```

```
Out[ ]= {8.83185 - 0.426785 i, 77.8195 - 7.53861 i}
```

```
Out[ ]= {8.83185, -0.426785, 78.1838}
```

```
Out[ ]= 0.
```

```
Out[ ]= {8.83185 - 0.426785 i, 77.8195 - 7.53861 i}
```

```
In[ ]:= qd1 = RandomReal[{-10, 10}, 2];
        qd = Append[qd1, qd1[[1]]^2 + qd1[[2]]^2]
        qu = Pup[qd]
        (y - x^2) /. Thread[{x, y} → qu]
        Pdown[qu]
Out[ ]= {-9.64215, 8.32047, 162.201}
Out[ ]= {-9.64215 + 8.32047 i, 23.7408 - 160.454 i}
Out[ ]= 0. + 0. i
Out[ ]= {-9.64215, 8.32047, 162.201}
```

The observant reader may have noticed that the third coordinate z is not used in the definition of **Pup**. But this is because z is assumed be already a function of x and y. The function **Pup** will still give a result even if $\{x, y, z\}$ is not in the domain of **Pup** and this result is in the domain of Pdown this will come in handy later to refine points in the domain of Pdown.

```
In[ ]:= Pdown[Pup[{2, 3, 11}]]
Out[ ]= {2, 3, 13}
```

We still must consider the infinite points. As mentioned in my *Plane Curve book* and Section 1.10 of this book, it is enough to find zeros of the top form where the equation is of the form $f = 0$. The top form for $x^2 - y = 0$ is x^2. So the only infinite complex solution of $x^2 - y = 0$ is {0, 1, 0} in homogeneous coordinates. But the top form for the paraboloid is $x^2 + y^2$ and the only real infinite solution is {0, 0, 1, 0}. So these two will map to each other. One just needs to check continuity in the homoge - neous space. By example

```
In[ ]:= Pdown[{100 + 100 I, (100 + 100 I)^2}]
Out[ ]= {100, 100, 20 000}
```

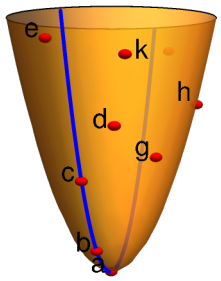
But by homogeneity this is {100,100,20000}/20000 which is essentially the homogeneous point {0,0,1,0}

```
In[ ]:= Pup[{100, 100, 10 000}]
Out[ ]= {100 + 100 i, 20 000 i}
```

which is close to the homogeneous point {0,1,0}. A complete proof of continuity is left to the reader.

So now we can conclude that the complex projective solution space of the parabola $y = x^2$ is topologi - cally the real projective space of the paraboloid. Here is an illustration where the points on the parabola are the points Pdown of the correspondingly named points of the solution set on the right.

Out[ ]=



a = {0, 0}
b = {1, 1}
c = {2, 4}
d ={2.2 + 1.2 i, 3.4 + 5.28 i}
e = {3.07281 - 0.638339 i, 9.03471 - 3.923 i}
f = {-3.07281 + 0.638339 i, 9.03471 - 3.923 i}
g = {1 + 2 i, -3 + 4 i}
h = {-1. + 2.5 i, -5.25 - 5. i}
k = {2.5 + 1.7 i, 3.36 + 8.5 i}

From our observation in Section 1 that the projective paraboloid is algebraically equivalent, hence topologically equivalent, to the sphere we can conclude that the complex solution space of the parabola is topologically equivalent to the sphere. But we can do better since we know the algebraic equivalence and describe the maps. Recall the global function defined in Section 2.1

In[ ]:=
```
PB2SP = paraboloid2sphere
```

Out[ ]= $\left\{\left\{0, 0, \frac{1}{2}, -\frac{1}{2}\right\}, \{1, 0, 0, 0\}, \{0, 1, 0, 0\}, \left\{0, 0, \frac{1}{2}, \frac{1}{2}\right\}\right\}$

In[ ]:=
```
PSdown[{u_, v_}] := TransformationFunction [PB2SP][Pdown[{u, v}]]
PSup[{x_, y_, z_}] := Pup[TransformationFunction [Inverse[PB2SP]][{x, y, z}]]
```

Illustrating with simple examples

In[ ]:=
```
PSdown[{3 I, 9}]
```

Out[ ]= $\left\{\frac{4}{5}, 0, \frac{3}{5}\right\}$

In[ ]:=
```
PSup[{4 / 5, 0, 3 / 5}]
```

Out[ ]= {3 i, -9}

In[ ]:=
```
PSup[{2 / 3, 1 / 3, 2 / 3}]
```

Out[ ]= {1 + 2 i, -3 + 4 i}

In[ ]:=
```
PSdown[{1 + 2 i, -3 + 4 i}]
```

Out[ ]= $\left\{\frac{2}{3}, \frac{1}{3}, \frac{2}{3}\right\}$

In[ ]:= Note that

In[ ]:=
```
PSdown[{0, 0}]
```

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

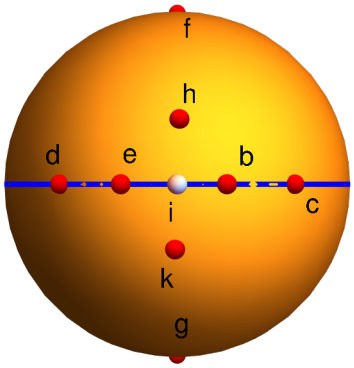
In[ ]:=
```
N[PSdown[{200, 40 000}]]
```

Out[ ]=
```
{0.99995, 0.00999975, 0.}
```

suggests that the infinite point of the parabola goes to {1, 0, 0} of the sphere. In fact, note that all real points on the parabola go to the equator of the sphere with solution {0,0} at {−1, 0, 0} in back of this sphere.

Illustrating by graphic

Out[ ]=




```
a = {0, 0}
b = {8, 64}
c = {3, 9}
d ={-3, 9}
e = {-7, 49}
i   infinite point
f = {i, -1}
g = {-i, 1}
h = {0.2 + 6. i, -35.96 + 2.4 i}
k = {-0.2 - 6. i, -35.96 + 2.4 i}
```

5.6.2 The Circle

Now it is easy to write down the mapping from the complex solution set of $x\text{^}2 + y\text{^}2 = 1$ using the global transformation P2C in Section 1.

In[ ]:=
```
P2C = p2cTransform2D
```

Out[ ]=
```
{{1, 0, 0}, {0, -0.5, 0.5}, {0, -0.5, -0.5}}
```

In[ ]:=
```
Cdown[{u_, v_}] := PSdown[TransformationFunction [Inverse[P2C]][{u, v}]]
Cup[{x_, y_, z_}] := TransformationFunction [P2C][PSup[{x, y, z}]]
```

The specified domain for `Cdown` is the complex solution set of the circle while the domain for **`Cup`** is the real Sphere. If these functions are called with other arguments one may still get a result but it may not be in the other domain and these functions may not be invertible for these values.

We check, using first the fact that the complex trigonometric functions sin and cosine also satisfy $\sin[\theta]^2 + \cos[\theta]^2 = 1$ for any complex θ.

```
θ = RandomComplex [{-8 - I, 8 + 8 I}]
(x^2 + y^2 - 1) /. Thread[{x, y} → {Cos[θ], Sin[θ]}]
p = {Cos[θ], Sin[θ]}
q = Cdown[p]
(x^2 + y^2 + z^2 - 1) /. Thread[{x, y, z} → q]
p - Cup[q]
```

```
3.5443 + 1.81567 i
```

```
1.77636 × 10^-15 + 8.88178 × 10^-16 i
```

```
{-2.90166 + 1.17228 i, -1.23606 - 2.75195 i}
```

```
{-0.124258, 0.291698, -0.948405}
```

```
0.
```

```
{-8.88178 × 10^-16 + 0. i, 0. - 8.88178 × 10^-16 i}
```

Next we check using the traditional trigonometric parameterizations of the real sphere

```
{s, t} = RandomReal [{-Pi, Pi}, 2];
pd = {Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}
qu = Cup[pd]
(x^2 + y^2 - 1) /. Thread[{x, y} → qu]
Cdown[qu]
```

```
{-0.322523, 0.706684, -0.629743}
```

```
{-1.17112 + 0.33659 i, -0.534488 - 0.737508 i}
```

```
5.55112 × 10^-17 + 1.11022 × 10^-16 i
```

```
{-0.322523, 0.706684, -0.629743}
```

We see that these functions on the specified domains give results in the required range and are inverses.

If we take a real solution of the circle and apply `Cdown` we get a point on the equator of the sphere, that is $z = 0$.

```
t0 = RandomReal [{-Pi, Pi}];
Cdown[{Cos[t0], Sin[t0]}]
```

```
{0.74778, -0.663946, 0.}
```

Some other values are

```
In[ ]:= Cdown[{1, 0}]
        Cdown[{-1, 0}]
        Cdown[{0, -1}]
Out[ ]= {0., -1., 0.}
Out[ ]= {0., 1., 0.}
Out[ ]= {-1., 0., 0.}
```

But note that Cdown will not work correctly on the point {0,1} due to a zero denominator in the calculation. There is also a problem with Cup at {1, 0, 0}. A work around is

```
In[ ]:= Limit[Cdown[{Cos[t], Sin[t]}], t → Pi / 2]
        Limit[Cup[{Cos[t], Sin[t], 0}], t → 0]
Out[ ]= {1., 0., 0.}
Out[ ]= {0., 1.}
```

While the real solution set of $x^2 + y^2 = 1$ is bounded the complex solution set is not. To find infinite points we take the maximal form which here is already homogeneous and solve. But note that $x^2 + y^2 = (x + i\,y)\,(x - i\,y)$ which has two complex solutions $x = -i\,y$ or $x = i\,y$. Normalizing at $y = 1$ we get solutions {-i, 1} or {i, 1}. So, in homogeneous coordinates there are two infinite points {i,1,0} or {$-i$, 1, 0}. Unfortunately our formula only works for affine points but note

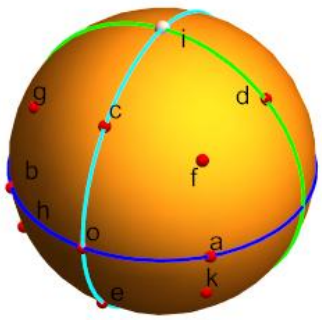
```
In[ ]:= (x^2 + y^2 - 1) /. Thread[{x, y} → {t i, Sqrt[1 - (t i)^2]}]
Out[ ]= 0
```

so these points are in our complex solution set for all t. Then our points map to sphere points by continuity

```
In[ ]:= Limit[ Cdown[{t i, Sqrt[1 - (t i)^2]}], t → ∞]
        Limit[ Cdown[{t i, Sqrt[1 - (t i)^2]}], t → -∞]
Out[ ]= {0., 0., 1.}
Out[ ]= {0., 0., -1.}
```

Here is an illustration of selected points on the complex circle and their images in the sphere.

In[ ]:=




```
o = {1., 0.}
a = {0.707107, 0.707107}
b = {0.707107, -0.707107}
c = {1.41421, 0. - 1. i}
d = {0. + 1. i, 1.41421 + 0. i}
e = {1.25, 0. + 0.75 i}
f = {0.866025 + 0.5 i, 0.866025 - 0.5 i}
g = {0.866025 - 0.5 i, -0.866025 - 0.5 i}
h = {0.75 + 0.25 i, -0.75 + 0.25 i}
i   infinite point
k = {0.75 - 0.25 i, 0.75 + 0.25 i}
```

The three curves in the sphere, $x = 0$, $y = 0$, $z = 0$ separate the sphere into 8 connected regions. In this example, since there is a strong relationship between the sphere and the circle which is embedded as the equator we can identify the regions with certain sets of solutions. For example the region containing the solution **f** will contain all solutions of the form $\{\alpha + \beta i, \gamma - \delta i\}$ for $\alpha, \beta, \gamma, \delta$ positive real numbers. It should be noted that rotations and reflections of the sphere take each of these regions to another. For example we can map the point **f** to the point **g** by

In[ ]:=
```
Cup[TransformationFunction [{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}][
  Cdown[{0.866025 + .5 I, 0.866025 - .5 I}]]]
```

Out[ ]=
```
{0.866025 - 0.499999 i, -0.866025 - 0.499999 i}
```

5.6.3 The Hyperbola

The first example were standard real conics we now look at one representative of an arbitrary real conic. So that my work can be replicated we use integer coefficients.

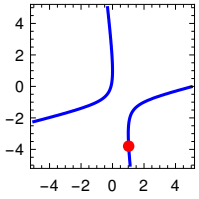
In[ ]:=
```
h1 = 1 + 10 x - 2 x^2 - 2 y + 8 x y + y^2;
```

This gives a hyperbola which contains, among many other points, the point **ph**:

In[ ]:=
```
ph = {1.0259094757506706` , -3.795341582563674` };
```

In[ ]:=
```
Show[ContourPlot[h1 == 0, {x, -5, 5}, {y, -5, 5}, ContourStyle → Blue, ImageSize → Tiny],
 Graphics[{Red, PointSize[.07], Point[ph]}]]
```

Out[ ]=



In Section 7.3 of my *Plane Curve Book* I show how to transform any plane conic to a parabola. In this case I know the transform so will not go through the steps. The transform from this hyperbola to the parabola is

```
In[ ]:= H2P = {{1.`, 1.`, 2.`}, {3.`, 0.`, -3.`}, {1.`, -2.`, -1.`}};
```

To check a pseudo -random point

```
In[ ]:= q = TransformationFunction [H2P][ph]
Out[ ]= {-0.101021, 0.0102051}

In[ ]:= (y - x^2) /. Thread[{x, y} → q]
Out[ ]= 1.04083 × 10^-17
```

Conversely

```
In[ ]:= rr = RandomReal[{-5, 5}]
Out[ ]= 1.52491

In[ ]:= rh = TransformationFunction [Inverse[H2P]][{rr, rr^2}]
Out[ ]= {3.69689, -0.39122}

In[ ]:= h1 /. Thread[{x, y} → rh]
Out[ ]= -1.42109 × 10^-14
```

Now we can write down the maps from the complex solution space of h1 = 0 to and from the real sphere.

```
In[ ]:= Hdown[{x_, y_}] := PSdown[TransformationFunction [H2P][{x, y}]]
        Hup[{x_, y_, z_}] := TransformationFunction [Inverse[H2P]][PSup[{x, y, z}]]
```

For the curious these functions are not terribly complicated

```
In[ ]:= Hdown[{x, y}]
Out[ ]= {(-1/2 + 1/2 Abs[(-3.+3. x)/(-1.+1. x-2. y)]) / (1/2 + 1/2 Abs[(-3.+3. x)/(-1.+1. x-2. y)]),
         Re[(2.+1. x+1. y)/(-1.+1. x-2. y)] / (1/2 + 1/2 Abs[(-3.+3. x)/(-1.+1. x-2. y)]),
         Im[(2.+1. x+1. y)/(-1.+1. x-2. y)] / (1/2 + 1/2 Abs[(-3.+3. x)/(-1.+1. x-2. y)])}

In[ ]:= Hup[{x, y, z}]
Out[ ]= {(0.166667 + 0.333333 (y/(1-x) + i z/(1-x)) + 0.166667 (y/(1-x) + i z/(1-x))^2) /
          (0.166667 + 0.333333 (y/(1-x) + i z/(1-x)) - 0.166667 (y/(1-x) + i z/(1-x))^2),
         (-0.5 - 9.25186 × 10^-18 (y/(1-x) + i z/(1-x)) + 0.166667 (y/(1-x) + i z/(1-x))^2) /
          (0.166667 + 0.333333 (y/(1-x) + i z/(1-x)) - 0.166667 (y/(1-x) + i z/(1-x))^2)}
```

which would look worse if our coefficients were not rational numbers. Note the built in function Abs is given by

Abs[$u + I v$] = Sqrt[$u^2 + v^2$] so square roots are involved.

Once again we see that real solutions of h1 = 0 go to the equator. Recall our random point on h1 was rh

```
In[ ]:= Hdown[rh]
Out[ ]= {0.39856 , 0.917142 , 0.}
```

To find in the infinite points we use the method given in Section 3.3 of my Plane Curve book.

```
In[ ]:= maxFormh1 = -2 x^2 + 8 x y + y^2
Out[ ]= -2 x^2 + 8 x y + y^2

In[ ]:= Reduce[maxFormh1 == 0, {x, y}]
Out[ ]= y == -4 x - 3 √2 x || y == -4 x + 3 √2 x
```

We can write down our infinite points

```
In[ ]:= N[{1, -4 - 3 Sqrt[2], 0}]
        N[{1, -4 + 3 Sqrt[2], 0}]
Out[ ]= {1. , -8.24264 , 0.}

Out[ ]= {1. , 0.242641 , 0.}
```

As an illustration

```
In[ ]:= sol4 = NSolveValues [h1 == 0 && x == 100, {x, y}]
Out[ ]= {{100. , -821.137}, {100. , 23.1374}}
```

Writing these as homogeneous points and dividing by 100 we see this is an approximation of the infinite point.

```
In[ ]:= Evaluate[{{100.` , -821.1374183841087` , 1}, {100.` , 23.137418384108656` , 1}} / 100]
Out[ ]= {{1. , -8.21137 , 1/100}, {1. , 0.231374 , 1/100}}
```

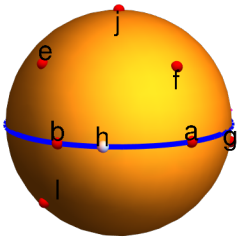
The coordinates on the sphere will be approximately

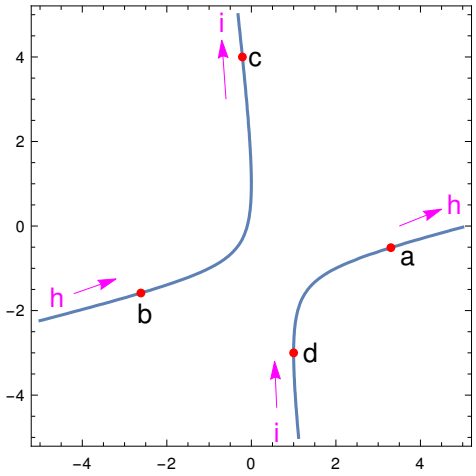
```
In[ ]:= ihd = Hdown[sol4[[1]]]
        hhd = Hdown[sol4[[2]]]
Out[ ]= {-0.708577 , -0.705633 , 0.}

Out[ ]= {0.698477 , 0.715633 , 0.}
```

To find **Hup[{1, 0, 0}**] we need to take a limit

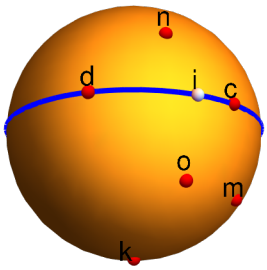
```
In[ ]:= Clear[t]

In[ ]:= Limit[Hup[{t, Sqrt[1 - t^2], 0}], t → 1]
Out[ ]= {-1. , -1.}
```

Here is the graphic showing this data and other selected points. The arrows on the plane graph show the direction of travel on the real plane curve which then is also on the spheres with infinite point **h** between **a,b** and infinite point **i** between **c** and **d**.





Out[•]=




```
a = {3.30081 , -0.511291}
b = {-2.61344 , -1.584}
c = {-0.212143 , 4}
d = {1. , -3.}
e = {-0.3235 + 0.7941 i, -0.8824 + 0.5294 i}
f = {1.324 + 0.7941 i, -1.118 + 0.5294 i}
g = {-0.2 - 0.0707107 i, -0.3 - 0.141421 i}
h   infinite point
i   infinite point
j = {0.5 + 0.5 i, -1 + i}
k = {0.5 - 0.5 i, -1 - i}
l = {-0.3235 - 0.7941 i, -0.8824 - 0.5294 i}
m = {0.1471 - 0.08824 i, 0.05882 - 1.235 i}
n = {0.6803 - 0.1949 i, -2.014 + 2.944 i}
o = {0.5 + 0.133975 i, -1. - 2.73205 i}
```

5.6.4 A complex conic

As our last example we consider the complex conic of section 5.4.6 where we called it conic2. Here we will call it just *k*. The linear terms just shear the conic so we can leave them off with out destroying our pseudo-randomness.

In[•]:=
```
k = -1 + (4 + i) x^2 - (6 - 5 i) x y - (3 - 2 i) y^2 ;
```

In 5.4 .6 we calculated a transformation matrix K taking this to normal form $y = x^2$ for reference we repeat this here

In[•]:=
```
K := ( -0.5123539884590109` + 0.9071396873769841` i  -0.40717365502919217` + 0.489428673
        3.855326236452737` - 2.611164128247496` i     -3.4893438886784645` + 7.309745651
       0.8220704880423995` - 0.4308659945517181` i    0.3183638426493982` - 0.1870499580
```

```
In[ ]:= K // MatrixForm
```

```
Out[ ]//MatrixForm=
```

$$\begin{pmatrix} -0.512354 + 0.90714\,i & -0.407174 + 0.489429\,i & -0.783827 + 0.473494\,i \\ 3.85533 - 2.61116\,i & -3.48934 + 7.30975\,i & 1.13508 - 1.87453\,i \\ 0.82207 - 0.430866\,i & 0.318364 - 0.18705\,i & -0.0470254 + 0.\,i \end{pmatrix}$$

Note

```
In[ ]:= Chop[Expand[FLT3D[{k}, K, {x, y}] / (0.5004363187858398` + 0.9368983792763038` i)]]
Out[ ]= {-1. x^2 + 1. y}
```

Our topological equivalences are from the complex solution space of k to the sphere are

```
In[ ]:= Kdown[{u_, v_}] := PSdown[TransformationFunction [K][{u, v}]]
        Kup[{x_, y_, z_}] := TransformationFunction [Inverse[K]][PSup[{x, y, z}]]
```

We can find a solution point of $k = 0$ by

```
In[ ]:= pk1 = fltMD[{2, 4}, Inverse[K]]
Out[ ]= {-0.396008 - 0.0322832 i, 0.199918 + 0.0146187 i}
```

Check

```
In[ ]:= k /. Thread[{x, y} → pk1]
Out[ ]= -1.11022 × 10^-16 + 7.21645 × 10^-16 i
```

```
In[ ]:= So
```

```
In[ ]:= ps1 = Kdown[pk1]
        (x^2 + y^2 + z^2 - 1) /. Thread[{x, y, z} → ps1]
Out[ ]= {0.6, 0.8, -1.77636 × 10^-16}
Out[ ]= -4.44089 × 10^-16
```

On the other hand if we take a point on the sphere

```
In[ ]:= ps1 = {1 / 3, 2 / 3, -2 / 3}
Out[ ]= {1/3, 2/3, -2/3}
```

```
In[ ]:= pk2 = Kup[ps1]
Out[ ]= {-0.397892 - 0.138114 i, 0.206391 - 0.191069 i}
```

```
In[ ]:= k /. Thread[{x, y} → pk2]
Out[ ]= -3.33067 × 10^-16 + 5.55112 × 10^-16 i
```

we get a solution point for $k = 0$.

We want to find the infinite points of k and their image in the sphere. We notice k is almost homoge - neous, in fact **kMax** is just $k + 1$, so we solve

```
In[ ]:= pinfs1 = NSolveValues [{k + 1, RandomReal [{- 10, 10}, 2].{x, y} + 1}, {x, y}]
Out[ ]= {{- 0.185944 - 0.0966776 i, - 0.028371 - 0.092407 i},
         {- 0.0434693 + 0.0012407 i, 0.10781 + 0.00118589 i}}
```

The following are then good estimates of the infinite points of *k*

```
In[ ]:= pinf1a = 2000 pinfs1[[1]]
        pinf1b = 2000 pinfs1[[2]]
Out[ ]= {- 371.889 - 193.355 i, - 56.7419 - 184.814 i}
Out[ ]= {- 86.9385 + 2.4814 i, 215.621 + 2.37179 i}
```

These go to

```
In[ ]:= Kdown[pinf1a]
        Kdown[pinf1b]
Out[ ]= {0.147137 , - 0.875366 , 0.460531}
Out[ ]= {0.998712 , 0.0435125 , - 0.0261242}
```

We can check for more infinite points or consistency of our estimates by trying again

```
In[ ]:= pinfs2 = NSolveValues [{k + 1, RandomReal [{- 10, 10}, 2].{x, y} + 1}, {x, y}]
Out[ ]= {{0.735128 + 0.922918 i, - 0.0655977 + 0.540252 i},
         {0.161756 - 0.00517528 i, - 0.401234 - 0.00302947 i}}
```

```
In[ ]:= pinf2a = 2000 pinfs2[[1]]
        pinf2b = 2000 pinfs2[[2]]
Out[ ]= {1470.26 + 1845.84 i, - 131.195 + 1080.5 i}
Out[ ]= {323.512 - 10.3506 i, - 802.469 - 6.05894 i}
```

These go to

```
In[ ]:= Kdown[pinf2a]
        Kdown[pinf2b]
Out[ ]= {0.149228 , - 0.874863 , 0.460809}
Out[ ]= {0.99877 , 0.0424832 , - 0.0255804}
```

These are close enough for plotting purposes, but we can get perhaps slightly better taking the average

```
In[ ]:= ik = Kdown[(pinf1a + pinf2a) / 2]
        jk = Kdown[(pinf1b + pinf2b) / 2]
Out[ ]= {0.149565 , - 0.874677 , 0.461054}
Out[ ]= {0.998791 , 0.0421058 , - 0.0253805}
```

We now would like to know any real points in the complex projective solution set of k. An interesting way to solve this is to note **Mathematica** has some very good very general minimization routines,

especially designed to work with Artificial Intelligence software. We use our trigonometric parameteriza tion of the sphere and minimize the norm of the imaginary part of **Kup** of a point.

```
In[ ]:= Clear[t, s];
        fff := Norm[Im[Kup[{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]}]]]
```

One run is

```
In[ ]:= Timing[Minimize[{fff, -Pi / 2 < s < Pi / 2, -Pi < t < Pi}, {t, s}]]
Out[ ]= {0.644184 , {1.6184 × 10^-10 , {t → -1.95586 , s → -1.36176}}}
```

Note this took my old Intel® Core™ i7-6800K CPU @ 3.40GHz × 12 computer running on Ubuntu 22.04 with Mathematica 12.3 less than .2 second to do this calculation. So one real point of *k* is

```
In[ ]:= gk = Chop[Kup[{Sin[s] Cos[t], Sin[s] Sin[t], Cos[s]} /.
              {t → -1.9558593848524621` , s → -1.3617638790835194` }], 10^-9]
Out[ ]= {-0.43975 , 0.0964036}
```

Checking:

```
In[ ]:= k /. Thread[{x, y} → gk]
Out[ ]= 2.61101 × 10^-11 + 7.32226 × 10^-10 i
```

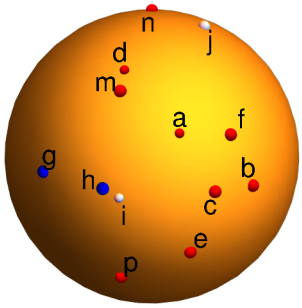
Another, less interesting but more accurate way to find real values is just to find a common solution to the real and imaginary parts .

```
In[ ]:= ComplexExpand [k]
Out[ ]= -1 + 4 x^2 - 6 x y - 3 y^2 + i (x^2 + 5 x y + 2 y^2)

In[ ]:= solre = NSolveValues [-1 + 4 x^2 - 6 x y - 3 y^2 == 0 && x^2 + 5 x y + 2 y^2 == 0, {x, y}, Reals]
Out[ ]= {{0.43975 , -0.0964036}, {-0.43975 , 0.0964036}, {0.693569 , -1.58188}, {-0.693569 , 1.58188}}

In[ ]:= k /. Thread[{x, y} → solre[[1]]]
Out[ ]= 1.4988 × 10^-15 - 4.996 × 10^-16 i
```

Thus there are 4 real values in this solution set .

We can picture selected points of the solution set and their image on the sphere by

Out[ ]=




```
a = {-0.2446 + 0.01095 i, 0.09187 + 0.337 i}
b = {-0.396 - 0.03228 i, 0.1999 + 0.01462 i}
c = {-0.3621 - 0.04404 i, 0.3133 + 0.1008 i}
d = {-0.1261 + 0.3381 i, -0.1843 + 0.2902 i}
e = {-0.3432 - 0.05874 i, 0.3981 - 0.167 i}
f = {-0.3607 + 0.02692 i, 0.1152 + 0.1399 i}
g = {0.43974998 , -0.096403558 }
h = {0.69356944 , -1.5818768 }
i   infinite  point
j   infinite  Point
m = {-0.04568 + 0.2439 i, -0.1703 + 0.3354 i}
n = {-0.6839 + 0.4244 i, -0.2995 + 0.01822 i}
p = {-0.1794 - 0.09911 i, 0.2826 - 0.3779 i}
```

5.7 The complex projective solution set of a smooth Cubic.

As mentioned in my Plane Curve Book in Section 5.5 the motivation for the concept of genus was the observation by Able and Jacobi in the early 1800's that certain functions involved in the indefinite integration of

$$\int \frac{dx}{\sqrt{4\,x^3 - g_2\,x - g_3}}$$

produced doubly periodic complex functions. It is not clear whether they realized that the image of these functions was a torus although we now know this as the Corollary in Section 5.3. By midcentury, however, Weierstrass must have realized this and came up with a parameterization of cubic functions of our normal form $y^2 = 4\,x^3 - g_2\,x - g_3$ using doubly periodic functions. Since we have shown that every smooth cubic has such a normal form we can use Weierstrass's parameterization to illustrate how to construct an explicit invertible map from the solution set of a cubic function to the torus.

Fortunately for us, Mathematica has a nice implementation of Weierstrass's functions. The intention of Mathematica is for finding closed forms for the integral above. However if you read the documentation carefully our use is at least partially covered.

We will be using the built in **WeierstrassP** and related functions to get our parameterization. Here is a general overview. We start with a smooth, meaning non-singular, cubic function f. We apply our **weierstrassNormalForm** procedure of Section 5.4.6 to get a function in Weierstrass normal form **wfn** = $4\,x^3 - g_2\,x - g_3 - y^2$ and transformation matrix **Af** We can extract g_2, g_3 to get a vector wfv using the function

In[ ]:=

```
weierstrassVector [wfn_] := {-Coefficient [wfn, x], -wfn /. Thread[{x, y} → 0]}
```

The Weierstrass parameterization of **wfn** is then given for complex t by

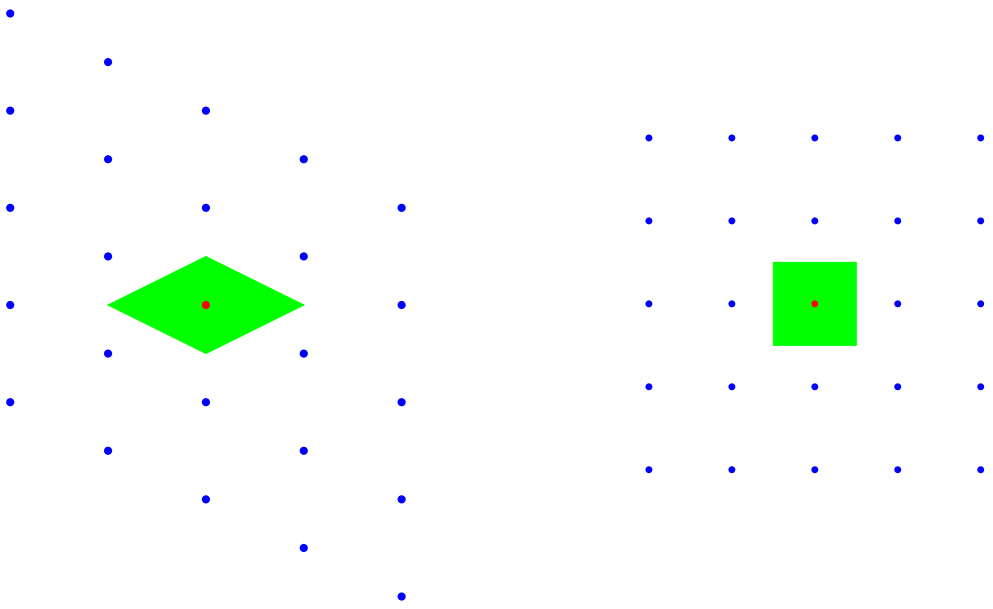
In[ ]:=
```
wPar[t_, wfv_] := {WeierstrassP[t, wfv], WeierstrassPPrime[t, wfv]}
```

If we want points on our original curve f they can be obtained from points $\{x, y\}$ on **wfn** using **fltMD[{x,y}, Inverse[Af]].**

Although **Mathematica** is necessarily thinking of our use of **WeierstrassP** they do give a built in inverse function with optional input **InverseWeierstrassP[{x,y}, wfv]** which will return t so that **wPar[t,wfv]= {x,y}** provided, of course, that $\{x, y\}$ is on the curve **wfn** with a very tight tolerance, eg. 10^{-16}.

wPar will be a doubly periodic function, the periods will depend on the vector **wfv** . To find these we use the built in **WeierstrassHalfPeriods [wfv]** . This will return 2 complex numbers which we can interpret as vectors using built in ReIm. These vectors may not be orthogonal, but they should be independent. We can then use these vectors to form a lattice and hence tiling of the plane so that each tile could be a complete domain for **wpar.** For example
where one tile, called the *fundamental domain* is shown . Now one can define a **TransformationFunction** on the plane to transform this to a tiling where each tile is a square of side 2π. Thus on one hand we can parameterize our curves wfn, including all complex values, hence f on this fundamental domain but because **wpar** is periodic over the entire region we can also parameterize our torus or even saddle surface from this same region.

In[ ]:=



Overall our goal is the map defined by the composition of the top arrows in the graphic below. This takes the solution set of our cubic curve to the torus. The inverse mapping is the composition of the lower arrows.

In[ ]:=

The orange maps come from the Weierstrass normal form projective linear transformation, the cyan map is the Inverse Weierstrass map with inverse **wPar**, the magenta map is the linear transformation shown above and the Gray maps are the Tpar transformation. The last maps could be replaced by the SSpar maps to the saddle surface. It should be noted that that the cyan maps between the Weierstrass normal form and the Weierstrass fundamental domain are where the hardest non-linear work is being done.

Unfortunately if the input is not accurate enough to give a point on the original curve or not accurate enough in the transformation to normal form then **InverseWeierstrassP** will fail to evaluate and will not give a warning message. For this reason, since we are using this inside a composition I recommend the following version of the inverse to WeierstrassP for this use. This check to see if the original **InverseWeierstrassP** works and if not finds a close point on the Weierstrass normal form of sufficient accuracy. The end result may be slightly inaccurate but will still end up on the torus. We could use the closest point algorithm if we are dealing only with real values, but this works for complex points as well.

In[ ]:=
```
inverseWP[pe_, wfv_] := Module[{w, ue, le, gve, p1, f, s, t},
  w = InverseWeierstrassP [pe, wfv];
  If[NumberQ[w], Return[w]];
  f = 4 s^3 - wfv[[1]] s - wfv[[2]] - t^2;
  gve = gtVec2D[f, pe, t, s];
  ue = pe + 10^-6 * gve;
  le = line2D[ue, {gve[[2]], -gve[[1]]}, s, t];
  p1 = {s, t} /. FindRoot[{f, le}, {s, ue[[1]]}, {t, ue[[2]]}];
  w = InverseWeierstrassP [p1, wfv];
  If[NumberQ[w], Return[w], Echo[pe, "Fail at InverseWeierstrassP "]];
 ]
```

Here is an example, note that the actual normal cubic is not entered.

In[ ]:=
```
fn = -y^2 + 4 x^3 - 3 x + 2;
wfvn = weierstrassVector [fn]
p = {-0.7054100411010285` + 0.8618642458406451` i, 3.`}
```

Out[ ]= {3, -2}

Out[ ]= {-0.70541 + 0.861864 i, 3.}

In[ ]:=
```
q = inverseWP[p, wfvn]
```

Out[ ]= 0.432544 - 0.800578 i

We wrongly use the display value of **p**

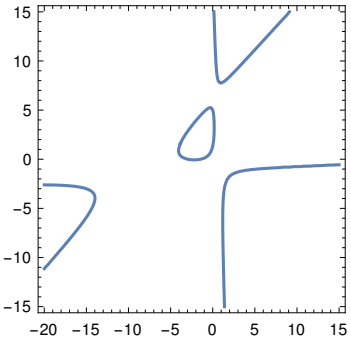
```
In[ ]:= pe = {-.70541` + .861864` I, 3.`};

In[ ]:= qe = inverseWP[pe, wfvn]
Out[ ]= 0.432545 - 0.800578 i

In[ ]:= Norm[q - qe]
Out[ ]= 5.43521 × 10^-7
```

5.7.1 First Example

Isaac Newton first described a cubic plane curve with three asymptotes (infinite points) and two ovals. A nice example is my Newton Hyperbola 836 described in my Plane Curve Book. My actual curve and plot is

```
In[ ]:= nh836 = 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;

In[ ]:= ContourPlot[nh836 == 0, {x, -20, 15}, {y, -15, 15}, ImageSize → Small]
```

Out[ ]=



I will first show, without going too deeply into the subject, how we can use our Weierstrass parameterization to plot the small oval. Then I will show how to plot the complex curve, with the equation above, on a torus.

The first step is to find the Weierstrass Normal Form . I first find inflection points

```
In[ ]:= infl836 = allInflectionPoints2D[nh836, x, y]
Out[ ]= {{-0.12874, 22.0683}, {6.57711, 12.6084}, {15.8884, -0.527148}}
```

By trial I find that the following one is the easiest to work with

```
In[ ]:= infl836 = {15.888428098023551` , -0.5271483543763018` };

In[ ]:= {wfn836, A836} = weierstrassNormalForm[nh836, infl836, x, y]
Out[ ]= {3.71474 - 7.25646 x + 4. x^3 - 1. y^2, {{0.0847194, 0.751728, -0.949785},
          {1.73264, -0.689376, 3.96493}, {-0.025385, 0.659709, 0.751093}}}
```

I can find points on the small oval

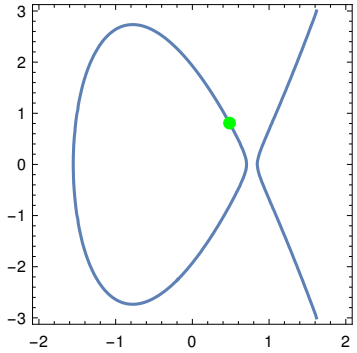
```
In[ ]:= sops = NSolveValues [nh836 == 0 && y == 3, {x, y}]
Out[ ]= {{-2.8511 , 3.}, {0.177637 , 3.}, {102.673 , 3.}}
```

The first two appear to be on this oval, the third is not .

```
In[ ]:= wsop = fltMD[sops⟦2⟧, A836]
Out[ ]= {0.484442 , 0.80881}
```

```
In[ ]:= Show[ContourPlot [wfn836 == 0, {x, -2, 2}, {y, -3, 3}],
  Graphics [{Green , PointSize [.04], Point[wsop]}], ImageSize → Small]
```

Out[ ]=



Now we find the Weierstrass vector from the normal form

```
In[ ]:= wfv836 = weierstrassVector [wfn836]
Out[ ]= {7.25646 , -3.71474}
```

The point wsop can be recovered by the Weierstrass parameter function

```
In[ ]:= inwsop = InverseWeierstrassP [wsop , wfv836]
Out[ ]= 1.11846 - 1.02932 i
```

```
In[ ]:= wPar[inwsop , wfv836]
Out[ ]= {0.484442 - 3.60822 × 10^-16 i , 0.80881 + 1.9984 × 10^-15 i}
```

which agrees with **wso**p after a chop . Now I claim, from experience, that all points on the small oval will have parameters with the same complex part

```
In[ ]:= cpsop = Im[inwsop] I
Out[ ]= 0. - 1.02932 i
```

We then guess by trial and error, our using the period information below, that some points on the small oval in the normal form are given by

In[ ]:=
```
smOvalW836 = Re[Table[wPar[t + cpsop, wfv836], {t, -1.8, 1.8, .2}]]
```

Out[ ]=
```
{{0.705235, -0.0157043}, {0.688668, -0.154575}, {0.639967, -0.345346},
 {0.543429, -0.642888}, {0.371722, -1.10581}, {0.0884376, -1.75379},
 {-0.333574, -2.4468}, {-0.863066, -2.7214}, {-1.349, -1.9194},
 {-1.55309, 0.}, {-1.349, 1.9194}, {-0.863066, 2.7214}, {-0.333574, 2.4468},
 {0.0884376, 1.75379}, {0.371722, 1.10581}, {0.543429, 0.642888},
 {0.639967, 0.345346}, {0.688668, 0.154575}, {0.705235, 0.0157043}}
```

In[ ]:= So

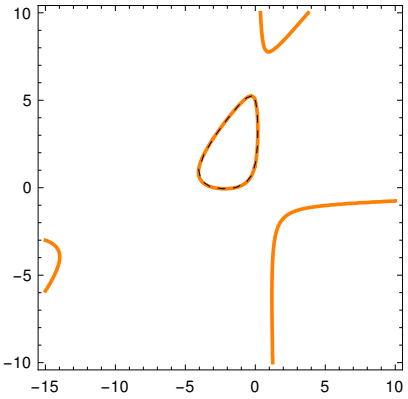
In[ ]:=
```
smOvalfh836 = fltMD[#, Inverse[A836]] & /@ smOvalW836
```

Out[ ]=
```
{{-0.237931, 5.24962}, {-0.613045, 5.14342}, {-1.19061, 4.70561}, {-1.98105, 3.9497},
 {-2.88308, 2.96237}, {-3.66224, 1.9249}, {-4.04775, 1.03715}, {-3.88672, 0.413041},
 {-3.23152, 0.0607155}, {-2.31345, -0.0630182}, {-1.41128, 0.00475001},
 {-0.705143, 0.261776}, {-0.237742, 0.738193}, {0.0290328, 1.46194}, {0.152364, 2.3981},
 {0.179131, 3.4085}, {0.13641, 4.29903}, {0.027237, 4.92339}, {-0.171109, 5.22568}}
```

Checking

In[ ]:=
```
Show[ContourPlot[nh836 == 0, {x, -15, 10},
  {y, -10, 10}, ContourStyle → Directive[{Orange, Thickness[.01]}]],
 Graphics[{Black, Dashed, Line[smOvalfh836]}]]
```

Out[ ]=



To go farther we need the Weierstrass Half Periods

In[ ]:=
```
WHalfPer = ReIm[WeierstrassHalfPeriods [wfv836]]
```

Out[ ]=
```
{{0., -1.02932}, {1.82442, 0.}}
```

These are easy to work with since they are orthogonal as real vectors and in the direction of the axes. Thus the transformation that gives the desired lattice is given by transformation matrix

```
In[ ]:= B836 = {{Pi / 1.8244232447332889` , 0, 0}, {0, Pi / -1.0293247723937833` , 0}, {0, 0, 1}};
        B836 // MatrixForm
```

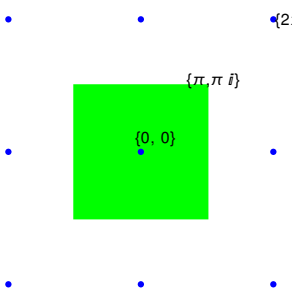
```
Out[ ]//MatrixForm=
        ⎛ 1.72196      0      0 ⎞
        ⎜    0     -3.05209  0 ⎟
        ⎝    0         0      1 ⎠
```

```
In[ ]:= fltMD[#, B836] & /@ WHalfPer
Out[ ]= {{0. , 3.14159}, {3.14159 , 0.}}
```

with the fundamental domain

```
In[ ]:= Graphics[{{Green , Polygon[{{-Pi , Pi}, {Pi , Pi}, {Pi , -Pi}, {-Pi , -Pi}}]},
          {Blue , PointSize[.02], Point[Flatten[Table[2 Pi {i , j}, {i , -1 , 1}, {j , -1 , 1}], 1]]},
          {Black , Text[{0 , 0}, {.7 , .7}], Text["{π,π i}", {π , π} + {.3 , .3}],
           Text["{2π,2π i}", {2 π + 1.8 , 2 π}]}}, ImageSize → Small]
```

Out[ ]=



Note we can modify the routine **reduce2pipi** in Global Functions to transform any complex number into a unique point of this domain.

```
In[ ]:= reduce2pipi [{12.35 , -8.756}]
Out[ ]= {-0.216371 , -2.47281}
```

We can now define a period function of period 2π from the fundamental square $-\pi \le x, y \le \pi$ to the plane

```
In[ ]:= WPpar[{x_ , y_}] :=
         fltMD[wPar[With[{z = fltMD[{x , y}, Inverse[B836]]}, z[[1]] + z[[2]] i], wfv836], Inverse[A836]]
```

Checking for periodicity

```
In[ ]:= r2 = RandomReal [{-5 , 5}, 2]
Out[ ]= {1.33713 , -2.96648}

In[ ]:= cr2 = WPpar[r2]
Out[ ]= {0.0123326 - 0.0580076 i, 1.3548 - 0.234791 i}
```

```
In[ ]:= WPpar[reduce2pipi[r2]]
Out[ ]= {0.0123326 - 0.0580076 i, 1.3548 - 0.234791 i}
```

Then we have a function from the standard torus to the complex projective solution set of fh836.

```
In[ ]:= Tor2nh836[{x_, y_, z_}] := WPpar[InvTor[{x, y, z}]]
```

For example consider the exact standard torus point

```
In[ ]:= pt1 = {7 π/(6 √2), -7 π/(6 √2), -√2 π/3};
```

```
In[ ]:= qc = Tor2nh836[pt1]
Out[ ]= {0.4043 + 0.23137 i, 6.59127 - 6.71793 i}
```

```
In[ ]:= nh836 /. Thread[{x, y} → qc]
Out[ ]= 6.38156 × 10^-12 - 5.61773 × 10^-12 i
```

Our inverse function is then

```
In[ ]:= nh2Tor836[{x_, y_}] := Module[{s, w, u, v},
          {s, w} = fltMD[{x, y}, A836];
          {u, v} = ReIm[inverseWP[{s, w}, wfv836]];
          {s, w} = reduce2pipi[fltMD[{u, v}, B836]];
          Tpar /. Thread[{r, t} → {s, w}]]
```

Checking we see for the example just above

```
In[ ]:= nh2Tor836[qc]
Out[ ]= {2.59168, -2.59168, -1.48096}
```

but

```
In[ ]:= N[pt1]
Out[ ]= {2.59168, -2.59168, -1.48096}
```

So this inverse does work .

Earlier we saw a large number of points on the small oval of nh836 was given by the list **smOvalfh836.** Applying this inverse function to random points in this set

```
In[ ]:= at = nh2Tor836[smOvalfh836[[2]]]
        bt = nh2Tor836[smOvalfh836[[11]]]
        ct = nh2Tor836[smOvalfh836[[13]]]
Out[ ]= {-4.32594, -1.39133 × 10^-15, 1.03185}
Out[ ]= {-1.91519, -4.01803 × 10^-15, -0.981497}
Out[ ]= {-2.60398, 2.63169 × 10^-15, -1.47593}
```

we deduce that this small oval goes to the curve on the torus $y = 0$ and $x < 0$. On the other hand computing some points on the large oval of nh836 we get results such as

```
In[ ]:= dc = NSolveValues [{nh836 , y – 9}, {x , y}][[2]]
        dt = nh2Tor836 [dc]
        ec = NSolveValues [{nh836 , x – 6}, {x , y}][[2]]
        et = nh2Tor836 [ec]
        fc = NSolveValues [{nh836 , y + 9}, {x , y}][[2]]
        ft = nh2Tor836 [fc]
Out[ ]= {2.68852 , 9.}
Out[ ]= {4.01089 , 0. , 1.30833}
Out[ ]= {6. , -0.949031}
Out[ ]= {1.83558 , 0. , -0.872775}
Out[ ]= {-17.8383 , -9.}
Out[ ]= {2.86122 , 0. , 1.54557}
```

Similarly we can deduce that these go the curve $y = 0$ and $x > 0$. In particular the real solution set of nh836 goes to the set $y = 0$ of the torus.

The infinite points of nh836 are real and given by

```
In[ ]:= rinf836 = infiniteRealPoints2D [nh836 , x , y]
Out[ ]= {{5.37825 , 4.96196 , 0}, {-4.66193 , -0.194615 , 0}, {-0.0830776 , 2.15705 , 0}}
```

We can' t handle these directly with nh2Tor836 so we work manually

On wfn836 these points go to

```
In[ ]:= rinfwfn836 = fltiMD[#, A836] & /@ rinf836
Out[ ]= {{1.33433 , 1.88016}, {53.8773 , 790.686}, {1.13286 , -1.14443}}
```

Then on the Weierstrass Fundamental domain they go to

```
In[ ]:= rinfW = inverseWP[#, wfv836] & /@ rinfwfn836
Out[ ]= {2.68921 + 0. i, 3.5126 + 0. i, 1.09589}
```

In the pipi square

```
In[ ]:= rinfpipi = (reduce2pipi [fltMD[ReIm[#], B836]]) & /@ rinfW
Out[ ]= {{-1.65247 , 0.}, {-0.234611 , 0.}, {1.88708 , 0.}}
```

so they end up in the torus by

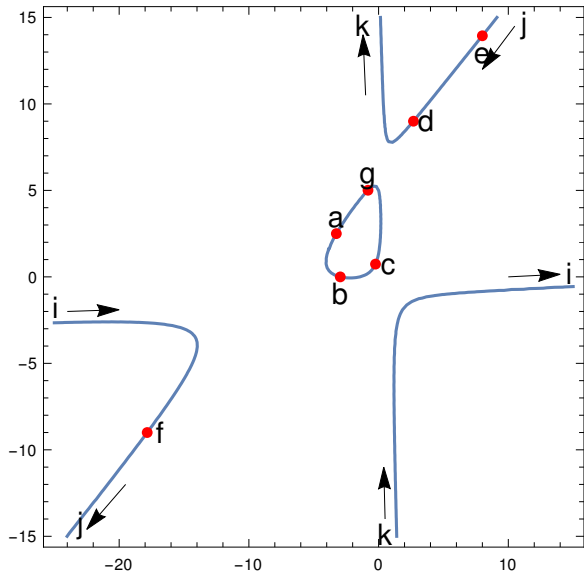
```
In[ ]:= {it, jt, kt} = Tpar /. Thread[{r, t} → #] & /@ rinfpipi
Out[ ]= {{3.22326 , 0. , 1.56867}, {1.80541 , 0. , 0.825839}, {3.45787 , 0. , -1.53863}}
```

The raw data is hidden in the print version . Here ares selected points of the complex projective solu-

tion set of nh836 with their value in the solution space and their image value on the torus, only top half is shown.

$$1 + x + 0.2\,x^2 - 0.008\,x^3 - y + x\,y + 0.2\,x^2\,y + 0.2\,y^2 - 0.2\,x\,y^2 - 0.008\,y^3 = 0$$

Out[ ]=

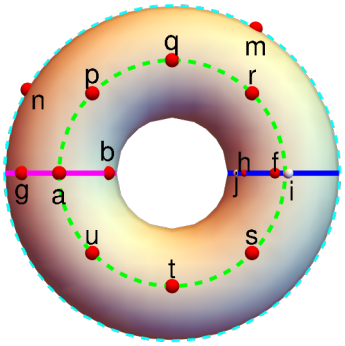


Out[ ]=

Selected Points

point	solution nh836	torus
a	{-3.25717 , 2.50039 }	{-3.14159 , 0, 1.5708 }
b	{-2.95346 , -0.00125406 }	{-1.80541 , 0. , 0.825839 }
c	{-0.237742 , 0.738193 }	{-2.60398 , 0, -1.47593 }
d	{2.68852 , 9.}	{4.01089 , 0. , 1.30833 }
e	{8. , 13.9397 }	{1.83558 , 0. , -0.872775 }
f	{-17.8383 , -9.}	{2.86122 , 0. , 1.54557 }
g	{-0.816527 , 5.00896 }	{-4.18879 , 0. , 1.1708 }
h	{-26.1733 , -2.68129 }	{2.0208 , 0. , 1.10055 }
i	infinite point	{3.22326 , 0. , 1.56867 }
j	infinite point	{1.80541 , 0. , 0.825839 }
k	infinite point	{3.45787 , 0. , -1.53863 }
m	{0.498908 - 0.491756 *i*, 6.95827 + 0.239189 *i*}	{2.35619 , 4.08105 , 0.}
n	{-3.46778 - 1.14937 *i*, 2.45776 - 1.43264 *i*}	{-4.08105 , 2.35619 , 0.}
p	{-0.15189 - 0.228384 *i*, 5.36036 + 0.0838272 *i*}	{-2.22144 , 2.22144 , 1.5708 }
q	{-4.1648 - 3.14172 *i*, 2.24528 - 3.50171 *i*}	{0. , 3.14159 , 1.5708 }
r	{-6.08816 - 8.56169 *i*, 1.02551 - 8.39626 *i*}	{2.22144 , 2.22144 , 1.5708 }
s	{-6.08816 + 8.56169 *i*, 1.02551 + 8.39626 *i*}	{2.22144 , -2.22144 , 1.5708 }
t	{-4.1648 + 3.14172 *i*, 2.24528 + 3.50171 *i*}	{0. , -3.14159 , 1.5708 }
u	{-3.46778 + 1.14937 *i*, 2.45776 + 1.43264 *i*}	{-2.22144 , -2.22144 , 1.5708 }

Out[ ]=



Note that the real part of our curve on the torus consists of the magenta and blue circles.

5.7.2 Second Example (Weierstrass Example 8)

For our second example we will use the pseudo random cubic

In[ ]:=
```
f8 = 0.8054578518310507` - 0.9960158007224074` x + 4 x^3 - y^2
```

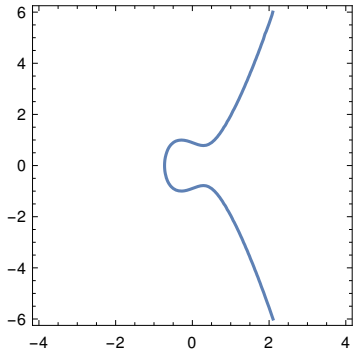
Out[ ]= $0.805458 - 0.996016\, x + 4\, x^3 - y^2$

But rather than plot our solution set on the torus we will plot it on the topologically equivalent saddle surface because this looks nicer.

This is already in Weierstrass normal form, which will save a little work, but is an interesting cubic by itself .

In[ ]:=
```
ContourPlot[f8 == 0, {x, -4, 4}, {y, -6, 6}, ImageSize → Small]
```

Out[ ]=



Our Weierstrass vector is

In[ ]:=
```
wvf8 = weierstrassVector [f8]
```

Out[ ]= {0.996016 , -0.805458 }

and our half periods are

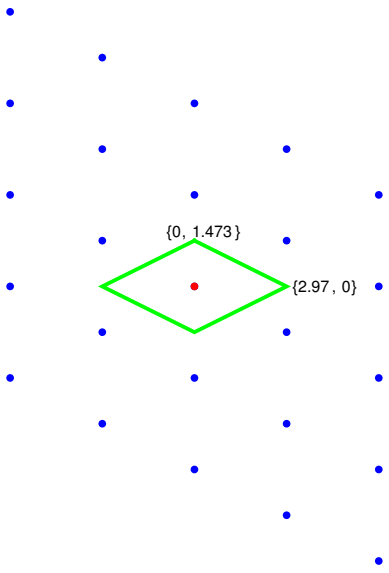
In[ ]:=
```
whp8 = ReIm[WeierstrassHalfPeriods [wvf8]]
```

Out[ ]= {{0. , -1.47304}, {1.4832 , -0.736519}}

Our lattice here is not rectangular

In[ ]:=
```
Graphics[{{Blue, PointSize[.02],
    Point[Flatten[Table[2 i whp8[[1]] + 2 j whp8[[2]], {i, -2, 2}, {j, -2, 2}], 1]]},
  {Red, PointSize[.02], Point[{0, 0}]}, {Green, Thickness[.01],
   Line[{-whp8[[1]], {2 whp8[[2, 1]], 0}, whp8[[1]], -{2 whp8[[2, 1]], 0}, -whp8[[1]]}]},
  {Black, Text[{0, 1.473}, -whp8[[1]] + .3], Text[{2.97, 0}, {4.2, 0}]}}, ImageSize → 200]
```

Out[ ]=



The region enclosed by the green diamond is a fundamental domain.

In[ ]:=
```
B8 = {{1.0590557059044317` , -2.1327298486921644` },
    {1.0590557059044317` , 2.1327298486921644` }};
```

In[ ]:=
```
B8 // MatrixForm
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 1.05906 & -2.13273 \\ 1.05906 & 2.13273 \end{pmatrix}$$

Then the linear rotation given by **B8** takes the side of the fundamental domain between the 2 marked points to the π - π square.

In[ ]:=
```
B8 . -whp8[[1]]
B8 . {2 whp8[[2, 1]], 0}
```

Out[ ]=
```
{-2.77266 , 0.662988 }
```

Part : Part 2 of {{0. , -1.02932 }, {1.82442 , 0.}}[{0.996016 , -0.805458 }] does not exist .

Out[ ]=
```
{0. + 2.11811 {{0. , -1.02932}, {1.82442 , 0.}}[{0.996016 , -0.805458 }][[2, 1]],
  0. + 2.11811 {{0. , -1.02932}, {1.82442 , 0.}}[{0.996016 , -0.805458 }][[2, 1]]}
```

We will focus on the map from the π - π square to and from the normal form cubic f8. We could then transport this to the torus, saddle surface or even hyperboloid if we wish. We will call these the **Up** and **Down** maps. The **Up** map is fairly easy

```
In[ ]:= wUp8[{u_, v_}] := With[{ip = Inverse[B8].{u, v}}, wPar[ip[[1]] + I ip[[2]], wvf8]]
```

The input is any real pair $\{u, v\}$ with $-\pi \le u, v \le \pi$ and output is a complex solution of f8 = 0.

Testing we have somewhat uneven accuracy

```
In[ ]:= Do[Echo[f8 /. Thread[{x, y} → wUp8[RandomReal[{-Pi, Pi}, 2]]]], {3}]
» -1.11022 × 10^-15 + 1.94289 × 10^-15 i
» -6.99441 × 10^-15 + 1.03806 × 10^-14 i
» -1.11355 × 10^-13 - 1.92069 × 10^-13 i
```

This will be good enough for plotting purposes, but we must use the more complicated inverse

```
In[ ]:= wDown8[{x_, y_}] := reduce2pipi[B8.ReIm[inverseWP[{x, y}, wvf8]]]
```

Here the input is a complex solution of f8 = 0 and output is a pair of real numbers in the π - π square.

```
In[ ]:= p1 = {-0.7180023637877024` , 0.2`};
        f8 /. Thread[{x, y} → p]
Out[ ]= -2.60817 + 1.72716 i

In[ ]:= q1 = wDown8[p1]
Out[ ]= {-3.06133 , -3.06133}

In[ ]:= Chop[wUp8[q1]]
Out[ ]= {-0.718002 , 0.2}
```

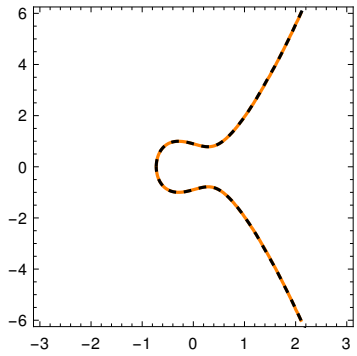
A pseudo random complex example is

```
In[ ]:= q2 = {-1.1200065916117765` , 3.034594424900483` }
Out[ ]= {-1.12001 , 3.03459}

In[ ]:= p2 = wUp8[q2]
        f8 /. Thread[{x, y} → p2]
Out[ ]= {0.0369951 - 0.468161 i, 0.940683 + 0.461922 i}
Out[ ]= -2.10942 × 10^-15 - 1.11022 × 10^-15 i

In[ ]:= q2a = wDown8[p2]
        q2 - q2a
Out[ ]= {-1.12001 , 3.03459}
Out[ ]= {0. , 3.55271 × 10^-15}
```

These illustrate how these maps go where they are supposed to do and are two sided inverses. Note above that p1 was a real solution of f8 and the result of wDown8 was a pair of equal real numbers. In the complex case one did not get a pair of equal real numbers. This suggests that the real diagonal of the π - π square maps to the real locus and conversely. For example note

```
In[ ]:= Show[ContourPlot[f8 == 0, {x, -3, 3}, {y, -6, 6},
  ContourStyle → Directive[Thickness[.01], Orange]], ParametricPlot[Chop[wUp8[{t, t}]],
  {t, -Pi, Pi}, PlotStyle → Directive[Black, Dashed]], ImageSize → Small]
```

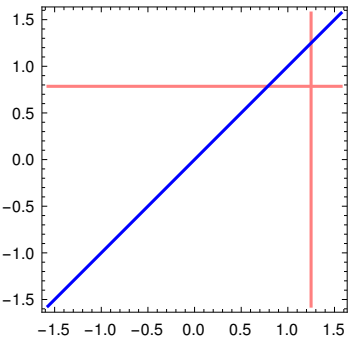
Out[ ]=



Now we decide to plot the complex solution set of f8 on the saddle surface instead of the torus, so we modify our up and down maps slightly, multiplying wDown8 by .5 and wUp8 by 2 to have the correct domain $\pi/2 \le s, t \le \pi/2$ for our SSpar parameterization from section 5.2.

```
In[ ]:= Show[SSparSpace, ParametricPlot[{t, t}, {t, -Pi / 2, Pi / 2}, PlotStyle → Blue]]
```

Out[ ]=



The red lines are the locus of those points in this domain that go to infinite points under SSpar, the blue obviously the diagonal. Since the red lines are given by x = ssa and y = ssb the intersection points are {ssa,ssa} and {ssb,ssb} where

```
In[ ]:= ssa = N[ArcTan[3]]
        ssb = N[ArcTan[1]]
```

```
Out[ ]= 1.24905
```

```
Out[ ]= 0.785398
```

These correspond to infinite points on the saddle surface and the following points of f8

```
In[ ]:= jf = Chop[wUp8[2 {ssa, ssa}]]
        kf = Chop[wUp8[2 {ssb, ssb}]]
```

```
Out[ ]= {-0.339802, -0.993461}
```

```
Out[ ]= {0.427876, -0.832241}
```

On the other hand the curve f8 has a single infinite point {0,1,0}. But the infinite point for the Weier-

strassP function is 0 so

In[ ]:= **wUp8[{0, 0}]**

Power : Infinite expression $\frac{1}{(0. + 0.\,i)^2}$ encountered .

Power : Infinite expression $\frac{1}{(0. + 0.\,i)^3}$ encountered .

Out[ ]= {ComplexInfinity , ComplexInfinity }

and so

In[ ]:= **is = SSpar[{0, 0}]**

Out[ ]= {2.20657 , -1.44126 , -3.18024}

is the corresponding point on the saddle surface.

As in the first example we can plot selected points as in the following table. One comment is that since **wDown8** has a corrector step one can type in an approximate point in **wDown8** to get an accurate point in the saddle surface and then use **wUp8** to get a very accurate point in the curve. For example for my first point

In[ ]:= **f8 /. Thread[{x, y} → {.373 , .801}]**

Out[ ]= -0.0000765738

In[ ]:= **as = SSpar[.5 wDown8[{.373 , .801}]]**

Out[ ]= {1.17821 , -0.910332 , -1.07256}

In[ ]:= **af = Chop[wUp8[2 InvSS[as]]]**
f8 /. Thread[{x, y} → af]

Out[ ]= {0.373058 , 0.800976}

Out[ ]= -1.11022×10^{-15}

so I have improved my residue by a factor of 10^{10}.

It is also worth noting that it is easy to find points on the saddle surface, just think of two real numbers, say r,s then {r,s,r s} is a point. One then then use **WUp8** to find a point in the solution set. For example let

In[ ]:= **gs = {3, 1.1, 3.3}**

Out[ ]= {3, 1.1, 3.3}

In[ ]:= **gf = Chop[wUp8[2 InvSS[gs]]]**

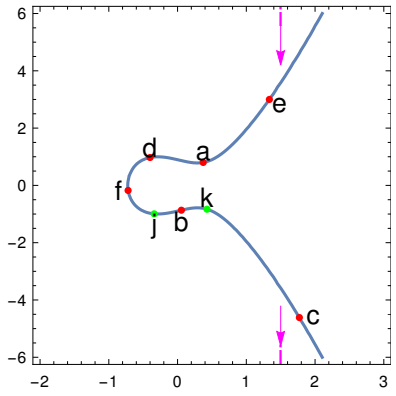
Out[ ]= {0.454513 - 0.21764 i, -0.713384 + 0.197311 i}

In[ ]:= **f8 /. Thread[{x, y} → gf]**

Out[ ]= $3.88578 \times 10^{-16} + 9.4369 \times 10^{-16}\,i$

$$0.805458 - 0.996016\,x + 4\,x^3 - y^2 = 0$$

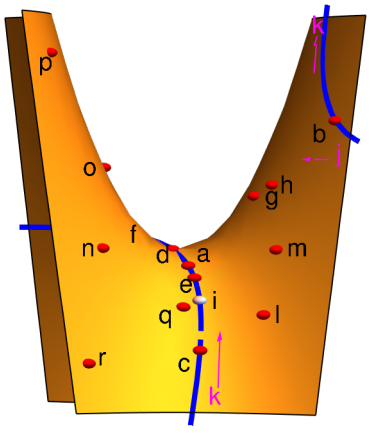
Out[ ]=



Out[ ]=

point	solution f8	saddle surface
a	{0.373058 , 0.800976 }	{1.17821 , -0.910332 , -1.07256 }
b	{0.0557982 , -0.866358 }	{7.05402 , 1.13234 , 7.98754 }
c	{1.77165 , -4.61346 }	{2.84656 , -2.19231 , -6.24053 }
d	{-0.4 , 0.973583 }	{0.145976 , -0.651173 , -0.095056 }
e	{1.33535 , 3.}	{1.65126 , -1.10044 , -1.81711 }
f	{-0.719703 , -0.176495 }	{-2.15759 , -0.378341 , 0.816306 }
g	{0.454513 - 0.21764 *i*, -0.713384 + 0.197311 *i*}	{3, 1.1, 3.3}
h	{0.249724 - 0.0718865 *i*, -0.776975 - 0.0124136 *i*}	{4, 1, 4}
i	infinite Point	{2.20657 , -1.44126 , -3.18024 }
j	{-0.339802 , -0.993461 }	infinite Point
k	{0.427876 , -0.832241 }	infinite Point
l	{-1.21094 - 0.307908 *i*, -1.12035 + 2.22907 *i*}	{5, -0.8, -4}
m	{-0.0439841 - 0.17124 *i*, -0.935075 - 0.0998141 *i*}	{5, 0, 0}
n	{-0.293159 + 0.652481 *i*, -1.61489 + 0.336897 *i*}	{0, -5, 0}
o	{0.219746 + 0.391215 *i*, -0.585984 + 0.343408 *i*}	$\{-\sqrt{5}, -\sqrt{5}, 5\}$
p	{-0.0428533 + 0.392137 *i*, -1.01089 + 0.30821 *i*}	{-2, -6, 12}
q	{-9.98835 - 0.196065 *i*, 1.85994 - 63.0407 *i*}	{1.8, -2, -3.6}
r	{-1.37753 + 0.802384 *i*, -2.99584 - 2.57118 *i*}	{1, -7, -7}

Out[ ]=



References

[Plane Curve Book] Barry H. Dayton, *A numerical approach to Real Algebraic Curves with the Wolfram Language,* Wolfram Media, 2018. The print version is available only from Amazon.com but for Mathematica users notebook versions are available free from Wolfram media (Zip file) or the author's website https://barryhdayton.space/curvebook/ChapterNotebooks.html on a chapter by chapter basis. The latter notebooks contain some of the corrections noted on the author's website.

[Space Curve Book] Available on author's website https://barryhdayton.space/SpaceCurves/spindex.html

[Surface Story] Available on author's website https://barryhdayton.space/SurfaceBook/surfdex.html

[Croom] Fred H. Croom, *Principles of Topology* , Dover Publications, 1989.