Explicit Regular Quadratic Surface Intersection Curves

Barry H Dayton
http://barryhdayton.space

Quadratic Surface Intersection Curves, QSIC, can be defined implicitly as the solution set of 2 real quadratic equations in three unknowns. The question of describing them explicitly by a parametric equation is a classical problem. Only recently has this been solved in general. L . Dupont, D . Lazard, S . Lazard and S . Petitjean [J . Symbolic Computing, 3 (43), 2008] presented a black box algorithm using exact computations. They give a 65 page discussion. This has been implemented but requires integer coefficients although the integers can be large enabling numerical solutions or examples to be approximated.

In 2013 I published a paper, see http://barryhdayton.space/RQSIC.pdf giving a probabilistic numerical approach. This is implemented by **Mathematica** in my Space Curve book http://barryhdayton.space/ The present implementation uses the WeierstrassP functions rather than quadratic rational functions. The complicated formulas are hidden in **Mathematica** func - tions, however it is not black box, there are decisions required of the user .

The method consists of three parts, first we give a carefully constructed, partly random projection that takes the intersection curve to a plane cubic. Then the cubic is, again partly randomized, transformed to a cubic in Weierstrass normal form. Both of these transforms have right indices. The third step is to parameterize the normal form cubic and then map this parameteriza - tion back to $\mathbb{R}^3$ using the inverses.

This version is meant as a stand-alone paper specifically for readers who are not regular **Mathematica** users. I mention, for those not familiar with **Mathematica,** that ordered pairs or triples use {a,b} or {a,b,c}. **Mathematica** is the platform I use and there will be some snipits of code throughout the paper, but the full code will be in the appendix. A **Mathematica Notebook** version of this paper is available at my **Mathematica** community page https://community.wolfram.com/web/bhdayton/home (use attached corrected notebook to my reply).

1. Preliminaries

1.1 Projective Linear Transformations and Transformation Functions

The most important concept in working with geometry is to understand the functions which give the equivalence between various objects of study. Unfortunately most discussions of geometry give short shrift to this important part of of the subject. For our purposes projective quadric surfaces and curves are equivalent if there is a an invertible *projective linear transformation* which takes one to the other.

In this paper we will need to use projective linear transformations from real projective n space $\mathbb{P}^n$ to projective m space $\mathbb{P}^m$. Here we can think of projective d space $\mathbb{P}^d$ given by vectors of length (d+1) under the equivalence relation $r\,v \equiv v$ for any real $r \neq 0$. Such a transformation can be given by an (n+1)×(m+1) real matrix A which operates by matrix-vector multiplication as this respects scalar multiplication. The transformation is invertible if and only if the matrix A is invertible.

Unfortunately working with equivalence classes is awkward, and often computationally in $\mathbb{P}^d$ we prefer working with affine points, that is vectors of length d where the affine point $\{x_1, \ldots, x_d\}$ represents the projective point $\{x_1, \ldots, x_d, 1\}$. S.S Abhyankar devotes Chapter 2 of his book [Algebraic Geometry for Scientists and Engineers, AMS , 1990] to what he calls *Frac- tional Linear Transformations* **Mathematica** uses **Transformation Functions** to achieve this. As an example given the matrix

In[ ]:=
$$A = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \end{pmatrix}$$

Out[ ]= {{a1, a2, a3, a4}, {b1, b2, b3, b4}, {c1, c2, c3, c4}}

we get a projective linear transformation from $\mathbb{P}^3 \to \mathbb{P}^2$ taking the affine point $p = \{x_1, x_2, x_3\}$

to

$$\left\{\frac{a_4 + a_1 x_1 + a_2 x_2 + a_3 x_3}{c_4 + c_1 x_1 + c_2 x_2 + c_3 x_3}, \frac{b_4 + b_1 x_1 + b_2 x_2 + b_3 x_3}{c_4 + c_1 x_1 + c_2 x_2 + c_3 x_3}\right\} \quad (1) \qquad (1)$$

I will use the notation **fltMD[p, A]** for this, **Mathematica** has a built in function for this

fltMD[p_, A_] := TransformationFunction[A][p]

For projective points $\{x_1, x_2, x_3, x_4\}$ with $x_4 \neq 0$ this is the same as applying the projective linear transformation with the same matrix since projective $\{x_1, x_2, x_3, x_4\}$ corresponds to affine $\left\{\frac{x_1}{x_4}, \frac{x_2}{x_4}, \frac{x_3}{x_4}\right\}$.

Note however that the domain of this transformation is the set of vectors $\{x_1, x_2, x_3\}$ where $c_4 + c_1 x_1 + c_2 x_2 + c_3 x_3 \neq 0$. Those affine points where $c_4 + c_1 x_1 + c_2 x_2 + c_3 x_3 = 0$ go to infinite, that is non affine, points of the range under the corresponding projective linear transformation.

An important fact about these transformations is that matrix multiplication on the transformation matrix gives composition of functions

fltMD[p, A B] = fltMD[fltMD[p, B], A]

In particular this implies that for an invertible matrix A then the fractional linear transformation **fltMD[p,A]** is invertible with inverse **fltMD[q,Inverse[A]]**

Given a polynomial equation f=0 in the range of fractional linear transformation by substituting a formula such as (1) with specific values for a_i, b_i, c_i with the domain variables and then simplifying we can get a formula for the inverse image of f = 0. If our fractional linear transformation is invertible, the inverse image of of the inverse transformation is a formula for the range of a polynomial equation in the domain. We have code for this *push forward operator* in the appendix under the name FLT3D.

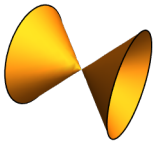
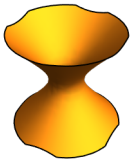
A technical warning is in order. These transformations are affine versions of projective transformations, as such the transforma - tion matrices should be seen as projective, that is a non-zero constant multiple of the transformation matrix gives the same transformation. For this reason, especially if the matrix has machine numbers entries then converting to a rational function may not behave as in equation (1) above, rather one may get equivalent, but not equal, fractions. We will see this below.

1.2 Quadratic Surfaces

Quadric surfaces are defined from our affine point of view by an equation

$$a_1 x^2 + a_2 x y + a_3 y^2 + a_4 x z + a_5 y z + a_6 z^2 + a_7 x + a_8 y + a_9 z + a_{10} = 0$$

The following chart from my Surface Story *http://barryhdayton.space/SurfaceBook/SurfaceStoryPartII.pdf* gives the possibilities.

Projective Real Quadric Surfaces					
Type	Not Surface	Degenerate	Cone	Ellipsoid	Hyperboloid
Possible Picture					
example	$(y-2x)^2+(z+3x)^2=0$	$xz=0$	$z^2=x^2+y^2$	$x^2+y^2+z^2=1$	$x^2+y^2-z^2=1$
singularity?	All	line	point	none	none
ruled?	no	two parts	single	none	double
Affine Variants	empty set point,line plane squared	parallel-planes	cylinder Cone	parabolic hyperbolic	elliptic saddle-Surface

The only possibilities for a real smooth projective surface are the the ellipsoid and hyperboloid. However note that the paraboloid, for example $z = x^2 + y^2$, and hyperbolic ellipsoid, otherwise known as hyperboloid of 2 sheets , for example $x^2 - y^2 - z^2 = 1$, are projectively equivalent to the standard ellipsoid. In affine geometry the hyperboloid in the chart is known as the elliptic hyperboloid, and the the hyperbolic hyperboloid is otherwise known as the saddle surface, for example $z = x\,y$. Again projectively they are the same. For details see my *Surface Story* , Chapter 2. The main difference between the hyperboloid and ellipsoid is that at every point the hyperboloid contains two lines through that point, but the ellipsoid contains no lines at all. Actually each point of an ellipsoid does contain two complex lines in the complex ellipsoid so in complex

geometry all smooth quadratic surfaces are projectively equivalent. But this post is only concerned with real quadratic surfaces.

2. The main Reduction

We give an algorithm to find parametric functions capable of plotting the intersection curve of 2 quadratic surfaces given by affine quadratic polynomials. This algorithm will be probabilistic in that random choices are made which most likely will work given that the two surfaces are not singular and the intersection curve is non-planar of genus 1. Ideally we would check our surfaces first and eliminate the many cases where this does not happen but one nice feature of this not being a black box algorithm is that we can plow ahead but check our progress and if is not working we can try different random choices or perhaps now check that our assumptions on the QSIC are correct. While the skeptical mathematician will have no trouble finding counter examples, the probability is that we will get a solution unless we are deliberately looking for counter examples.

The first step is our main reduction where we model the intersection curve with a plane non-singular, hence genus 1, cubic curve. In this paper I give this as a mathematical algorithm, but **Mathematica** code is in the Appendix.

Algorithm nsQSIC

Input : Two quadric polynomials Q_1, Q_2 in three variables x, y, z and a point p in the intersection. This point p could be chosen by the **Mathematica** algorithm `findQpts` in the Appendix.

Output: A cubic plane curve with equation h, hopefully non singular (check!), a fractional linear transformation $\Omega : \mathbb{R}^3 \to \mathbb{R}^2$ so that for each point s of the intersection **Ω(s)** is a point of h, and a rational polynomial function $\mho : \mathbb{R}^2 \to \mathbb{R}^3$ which is a right inverse of Ω, that is for each point q of h, $\mho$(q) is in the QSIC and Ω($\mho$(q)) = q.

Step 1 : We append 1 to p to get a vector of length 4 and normalize to norm 1, and then append a random real 3×4 matrix to obtain a 4×4 matrix which we then orthogonalize and reverse the order of rows. This gives an orthogonal matrix with a projective equivalent of p as last row which we will call A. This will cause the point p to be transformed to an infinite point on h by the fractional linear transformation determined by A. The linear fractional transformation given by the 3×4 matrix of the first 3 rows of A will be the output Ω.

Step 2: The QSIC, that is the two quadratics, are transformed to quadratics F_1, F_2 so that the fractional linear transforma - tion determined by matrix A sends all points on Q_1, Q_2 to points on F_1, F_2 respectively. These quadratics can be calculated essentially by taking the preimage of the transformations to Q_1, Q_2 given by A^{-1}. This trick which can only be done by invertible transformations is implemented by the code `FLT3D` in the appendix. With probability 1 the quadratics F_1, F_2 will have non-zero coefficient for each monomial in x, y and z of degree 3 or less.

Step 3 : The sum of all terms of total degree d of a polynomial will be called the *Form* of degree d. We let

L be the form of degree 1 of F_1
M be the form of degree 1 of F_2
R be the form of degree 2 of F_1
S be the form of degree 2 of F_2

Then h will be the expansion of L×S-R×M with variable z evaluated to 1. Since each term of L×S, R×M will be a product of of a polynomial of degree 1 times a polynomial of degree 2 it will be of at most degree 3, so h will be of degree h with probability 1. It is less obvious that Ω will take all points of the original QSIC to h but this is the trick that classical mathematicians discov - ered. This is something that should be checked.

Step 4: There is a complicated formula defining $\mho$ best described by **Mathematica** code. in the Appendix.

In[ ]:=
```
Ʊ = Take[Inverse[A].Join[#1, {1, -R/L /. Thread[{x, y, z} → Append[#1, 1]]}], 3] /
    Last[Inverse[A].Join[#1, {1, -R/L /. Thread[{x, y, z} → Append[#1, 1]]}]] &
```

Out[ ]=
```
Take[Inverse[A].Join[#1, {1, -R/L /. Thread[{x, y, z} → Append[#1, 1]]}], 3] /
  Last[Inverse[A].Join[#1, {1, -R/L /. Thread[{x, y, z} → Append[#1, 1]]}]] &
```

The motivation for this algorithm comes from a classical computation where the input QSIC already had generic coefficients. This was done in the projective setting where Ω was simply the projection of $\mathbb{P}^3 \to \mathbb{P}^2$ evaluating z to 1 and Ʊ inserted a 1 after y.

Because this algorithm depends on choosing a point *p* of the QSIC and the somewhat random construction of matrix A each time the code is evaluated there are different values of the output. The output must be saved for the rest of the rest of the calculation of the QSIC as they can't be replicated.

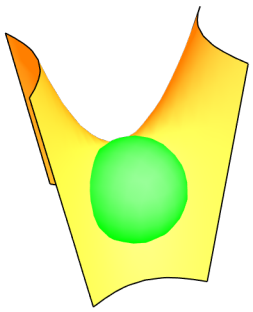
We do an example of the output of nsQSIC for

In[ ]:=
```
Q = {x y - z, (x - 1)^2 + (y + 1)^2 + (z + .9)^2 - 3.6};
```

The contour plot of the surfaces is

In[ ]:=
```
ContourPlot3D [{Q[[1]] == 0, Q[[2]] == 0}, {x, -4, 4}, {y, -4, 4}, {z, -4, 4}, Mesh → None,
 ImageSize → Small, ContourStyle → {Orange, Green}, Axes → False, Boxed → False]
```

Out[ ]=



Our point is

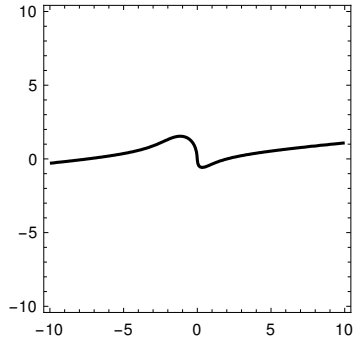
p0 ={-0.445491, -0.499856, 0.222681}

Then running nsQSIC and saving our values under the names p0, A0, h0, Ω0, Ʊ0

In[ ]:=
```
h0 = 0.0134767 + 2.68149 x - 0.966728 x^2 - 0.178344 x^3 +
  0.212614 y + 2.02179 x y + 2.06994 x^2 y + 0.0498793 y^2 + 0.0173951 x y^2 + 0.831885 y^3;
```

```
In[ ]:= ContourPlot [h0 == 0, {x, -10, 10}, {y, -10, 10}, ImageSize → Small]
```

Out[ ]=



In[ ]:=

A0

```
Out[ ]= {{0.865333 , 0.184098 , 0.108445 , 0.453372 }, {-0.000329302 , -0.0169023 , 0.974095 , -0.225508 },
  {-0.344536 , 0.893878 , 0.079216 , 0.275683 }}
```

```
Ω0[{x, y, z}]
```

$$\text{Out[ ]=}\ \left\{\frac{0.453372 + 0.865333\ x + 0.184098\ y + 0.108445\ z}{0.275683 - 0.344536\ x + 0.893878\ y + 0.079216\ z},\ \frac{-0.225508 - 0.000329302\ x - 0.0169023\ y + 0.974095\ z}{0.275683 - 0.344536\ x + 0.893878\ y + 0.079216\ z}\right\}$$

Note that we expected the numbers above to come from the matrix A0 but **Mathematica** normalized somewhat, this gives an equivalent transformation to $\Omega 0[\{x,y,z\}] = \text{fltMD}[\{x,y\},A0]$. Next

$$\text{Ʊ0}[\{x, y\}] = \left\{\frac{0.0470403 + 1.31717\ x^2 - 0.471226\ y - 0.223932\ y^2 + x\,(-0.500068 + 2.2469\ y)}{-0.52182 + 2.30415\ x + 1.\ x^2 - 0.171647\ y - 0.410038\ x\,y + 0.0298039\ y^2},\right.$$

$$\frac{1.12363 + 0.178448\ x^2 + x\,(0.895565 + 0.849293\ y) + 2.13362\ y - 0.285813\ y^2}{-0.52182 + 2.30415\ x + 1.\ x^2 - 0.171647\ y - 0.410038\ x\,y + 0.0298039\ y^2},$$

$$\left.\frac{0.101291 - 0.548793\ x - 0.235047\ x^2 - 0.855663\ y - 1.616\ x\,y - 2.14746\ y^2}{0.52182 - 2.30415\ x - 1.\ x^2 + 0.171647\ y + 0.410038\ x\,y - 0.0298039\ y^2}\right\}$$

We now check our assertions using Mathematica, first Ω0 should send p0 to an infinite point

```
In[ ]:= Ω0[p0]
```

Power : Infinite expression $\frac{1}{0.}$ encountered .

Power : Infinite expression $\frac{1}{0.}$ encountered .

```
Out[ ]= {ComplexInfinity , ComplexInfinity }
```

Next we show a somewhat random point p1 on our QSID maps to *h*

```
In[ ]:= p1 = findQpts [Q, {x, y, z}, 2][[2]]
Out[ ]= {-0.333333 , 0.00867049 , -0.00289016 }
```

Evaluating h at Ω0(p1) we get a very small residue

```
In[ ]:= h0 /. Thread[{x, y} → Ω0[p1]]
Out[ ]= 5.55112 × 10^-17
```

Now picking a random point on h0 we show ℧0 sends h to the QSIC

```
q0 = {-10.185851944205135` , 0.38297149894397153` }
```

```
In[ ]:= p3 = ℧0[q0]
Out[ ]= {1.50186 , 0.27477 , 0.412667 }
```

```
In[ ]:= Q /. Thread[{x, y, z} → p3]
Out[ ]= {1.11022 × 10^-16, 4.44089 × 10^-16}
```

And finally

```
In[ ]:= Ω0[p3] = {-10.18585194420513` , 0.3829714989439713` }
Out[ ]= {-10.1859 , 0.382971 }
```

which was q0 so our requirements were met.

Picking one pseudo-random point to test is not a proof, but it is a strong indication that the algorithm is working on this example so we can proceed. If there is failure on one of the above tests possibly run the algorithm again, it might have been a bad choice of random points. Repeated failures indicates this algorithm is not going to work on this QSIC, perhaps this QSIC is not of genus 1, maybe singular or planar.

3. Transforming the cubic to Weierstrass Normal form.

The Weierstrass Normal form for a cubic is

$$y^2 = 4x^3 - g_2 x - g_3$$

In my *Plane Curve Book* [A Numerical Approach to Real Algebraic Curves with the Wolfram Language, Wolfram Media, 2018, https://wolfr.am/Dayton] I discussed this in Chapter 7. All regular cubics are fractional linearly equivalent to a cubic in Weierstrass normal form, but the form is not unique. Again the key thing is not just the equation but the transformation taking the curve h above to this form. This is again given by a transformation function. One issue is that since I was working numerically in that book the coefficient 4 which made some hand calculations easier did not seem necessary. However our implementation of this assures that the coefficient of x^3 is 4. Our code in the Appendix slightly changed from my book to give the correct equation which will also then changes the coefficients g_2, g_3. It is actually these numbers, not the equation, that will be important, so the new algorithm displays the equation but returns the pair $\{g_2, g_3\}$ which are called the *Weierstrass Invariants*.

In addition to the cubic h, one must input a choice of inflection point for the curve. Since h is assumed to be a non-singular real cubic there will be a real inflection point, in fact 3, if one counts multiplicity. For the reader's convenience I include a **Mathematica** function in the appendix to find these along with the normal form transformation.

For the example in the previous section we can pick the inflection point

```
In[ ]:= infPt1 = {-0.646618667877992` , 0.2880193877589238` };
```

Then running our example, possibly several times to get a real normal form, important for plotting.

```
In[ ]:= {wg, B2} = weierstrassNormalForm [h, infPt1 , x, y];
```

Normal Form $-0.481377 + 4.36185\,x + 4.\,x^3 - 1.\,y^2$

So the Weierstrass invariants are

In[3]:=
```
wg
```

Out[3]=
```
{-4.36185 , 0.481377 }
```

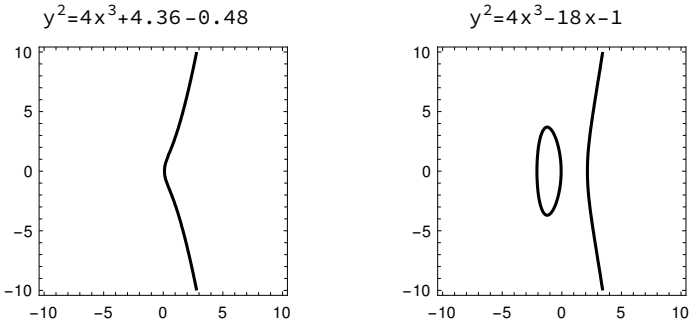
and B2 is

Out[1]//MatrixForm=
$$\begin{pmatrix} -2.09677 & -1.20156 & 1.34248 \\ -7.13704 & 1.30653 & -3.98131 \\ -1.33852 & 3.05905 & 2.4188 \end{pmatrix}$$

Again we save our output for the rest of this example since repeated runs will give different values. In particular some runs may give complex invariants which we want to avoid, so we may wish to run this several times until we get nice real invariants which we then save. Also save the corresponding transformation.

Note that in the affine plane the plot of a normal form curve can be in the topological shape of one of the two examples.

Out[5]=



In the first case we have, in the real affine plane, one topological component which has only one infinite point giving a loop in the real projective plane. In the second case we get two affine topological components which become two loops in the projective plane. Thus a regular QSIC curve can have one or two topological components only and this will be reflected in the normal form of the cubic produced by algorithm **nsQSIC**.

4. Parameterizing the normal form

The importance of the normal form is that we can easily parameterize cubics in normal form. One way that I used in earlier versions of my QSIC investigations is with is square roots in the numerator and denominator. In this case the parameterization of $y^2 = 4x^3 - g_2 x - g_3$ is

$$\left\{x, \pm\sqrt{4x\text{^}3 - g_2 x - g_3}\right\}$$

One may then worry about which sign to take for what x and what part of the curve. Piecewise parameterization will be needed. This gets particularly messy in the next section where we lift this parameterization to the QSIC with our rational functions.

In this paper I instead propose the use of the Weierstrass P functions which are conveniently and quickly implemented by **Mathematica.** The reader not familiar with these functions may wish to explore elsewhere for the construction and calculation of these functions, here we will only explain how to use them. These are complex functions of a single complex variable that are periodic with two real independent complex periods and a pole at the origin. These functions are themselves parameterized by Weierstrass invariants. Fortunately **Mathematica** also has built-in procedures to calculate the periods from the invariants, from the periods we can describe fundamental period parallelograms which completely determine the function. Unlike some theoretical expositions we put the pole in the middle of fundamental parallelogram rather than the corners.

With Mathematica we can parameterize real cubics in real Weierstrass Normal form. The Mathematica syntax is

In[ ]:=
```
μ[t_, {g2_, g3_}] := Re[{WeierstrassP [t, {g2, g3}], WeierstrassPPrime [t, {g2, g3}]}]
```

for the curve $y^2 = 4x^3 - g2\,x - g3$. The domain for t is the entire complex plane but it is enough to use only the fundamental period parallelogram to obtain the entire complex curve as this parameterization is also periodic with the same periods. We are only interested in real values of the normal form curve and if t, g2, g3 are all real then we will get a real value. However we will see that to get all real values of the normal form curve we will have to use some complex points in the fundamental period as well, especially for normal curves with 2 topological components. One technicality with the **Mathematica** implementation is

that the **WeierstrassP** functions use a numerical method to do the calculations and t must be a Mathematica machine number, that is a decimal number. One could replace t by **N[t]** in the formula above if this is a problem. If we plan to use this with **ParametricPlot** or **ParametricPlot3D** the parameter there must be real. In some cases below we need complex parame - ters but fortunately the imaginary part is constant so that constant can be added to real t in the formula above.

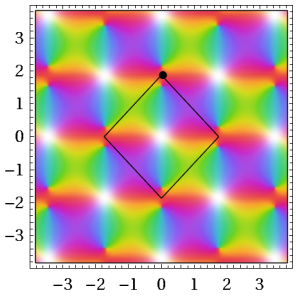
To help find the correct parameter ranges for given Weierstrass invariants I have a Mathematica procedure called wpgram - Plot[wpar] with code in the Appendix. Unlike my earlier algorithms this is deterministic so one will get the same result each time it is run with the same invariants. For the example above we get

```
In[25]:= wpgramPlot [{-4.3618451810711205` , 0.48137714411463256` }]
```

» hps {{0.870595 , 0.93462 }, {-0.870595 , 0.93462 }}

» corners {{2.22045 × 10^-16, 1.86924 }, {1.74119 , 0.}, {-2.22045 × 10^-16, -1.86924 }, {-1.74119 , 0.}}

» x-axis points {{0.0457074 , 1.86924 }}



In this example the fundamental parallelogram is given in black, the white regions surround the poles and the other colors are determined by the argument of the WeierstrassP function at that point. We don't really care about these arguments but they do help illustrate the periodicity. Note how opposite sides of the fundamental parallelogram have the same coloring, this is because periodicity requires opposite points to be the same. We saw in the plots in the last section that Weierstrass normal form cubics will intersect the x-axis in one or three points. A preimage on the boundary of the parallelogram of each of these points is shown as a black dot. The values of the half periods, corner points of the parallelogram and the pre-images of the x-axis points are given approximately and will be useful. Note in this example the x-axis point is one of the corners, approxi - mately, but by periodicity then all the corners will map to that same real value.

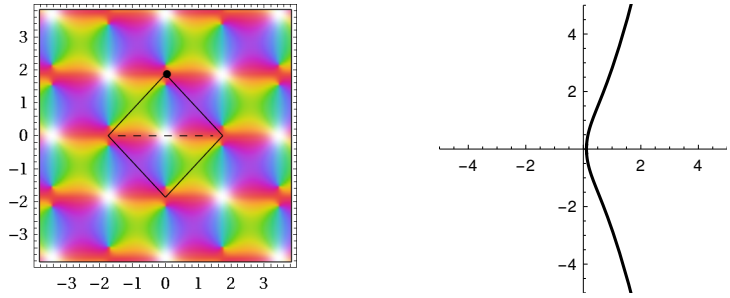
Since we are plotting a real curve in this case we will get all real values of the curve using real points of the parameter. So we can read off the value of the two real corner points from the information above. Since plotting will only needs to be accurate to 3 decimal points we can get away with approximate values , our interval is $-1.1963 \le t \le 1.1963$ we use slightly larger range for the plotting routine. Here we add our parameter interval to our graphic as a dashed line and show the parametric plot. We give the code here

```
In[27]:= Row[{
           Show[wpgramPlot [{-4.36184 , 0.481377 }], Graphics [{Black , Dashed , Line[{{-1.8 , 0}, {1.8 , 0}}]}]],
           "        ", ParametricPlot [μ[t, {-4.36184 , 0.481377 }], {t, -1.8 , 1.8},
            PlotRange → 5, ImageSize → 150]}]
```

» hps {{0.870595 , 0.93462 }, {-0.870595 , 0.93462 }}

» corners {{0. , 1.86924 }, {1.74119 , 0.}, {0. , -1.86924 }, {-1.74119 , 0.}}

» x-axis points {{0.0457075 , 1.86924 }}



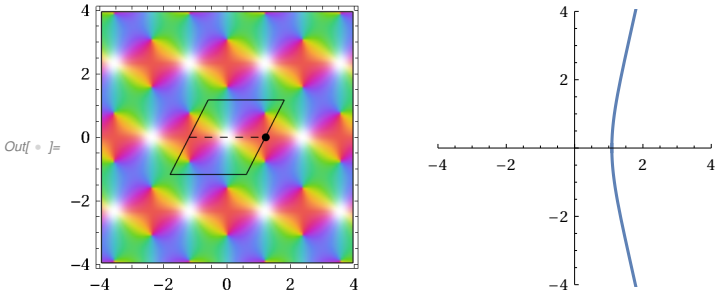
There are 4 other cases we need to consider. Again we show the code

```
In[ ]:= Row[
          {Show[wpgramPlot [{2, 3}], Graphics [{{Black , Dashed , Line[{{-1.19722` , 0}, {1.19722` , 0}}]}}]],
           "      ", ParametricPlot [μ[t, {2, 3}], {t, -1.214 , 1.214}, PlotRange → 4, ImageSize → 150]}]
```

» hps {{1.19722 , 0.}, {0.59861 , 1.17514 }}

» corners {{1.79583 , 1.17514 }, {0.59861 , -1.17514 }, {-1.79583 , -1.17514 }, {-0.59861 , 1.17514 }}

» x-axis points {{1.21402 , 0.}}

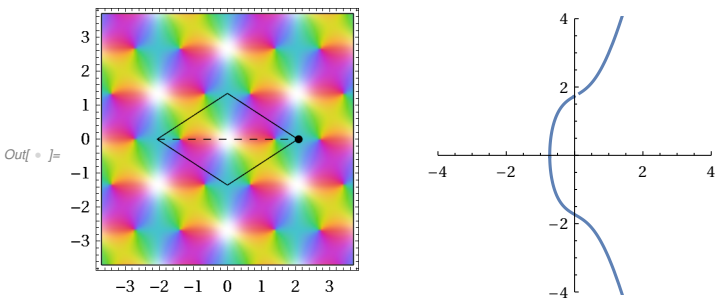


```
In[ ]:= Row[
          {Show[wpgramPlot [{-2, -3}],
             Graphics [{Black , Dashed , Line[{{-2.06563 , 0.`}, {2.06563 , 0.`}}]}]], "      ",
           ParametricPlot [μ[t, {-2, -3}], {t, -2.06563 , 2.06563 }, PlotRange → 4, ImageSize → 150]}]
```

» hps {{1.03282 , -0.674662 }, {1.03282 , 0.674662 }}

» corners $\{\{2.06563 , 4.44089 \times 10^{-16}\}, \{0. , -1.34932 \}, \{-2.06563 , -4.44089 \times 10^{-16}\}, \{0. , 1.34932 \}\}$

» x-axis points $\{\{2.09029 , -6.66134 \times 10^{-16}\}\}$



This next one requires three plots to keep the parameter in the fundamental period parallelogram. We show the parts using different colors.

In[ ]:=
```
impart = 1.19722` ;
```

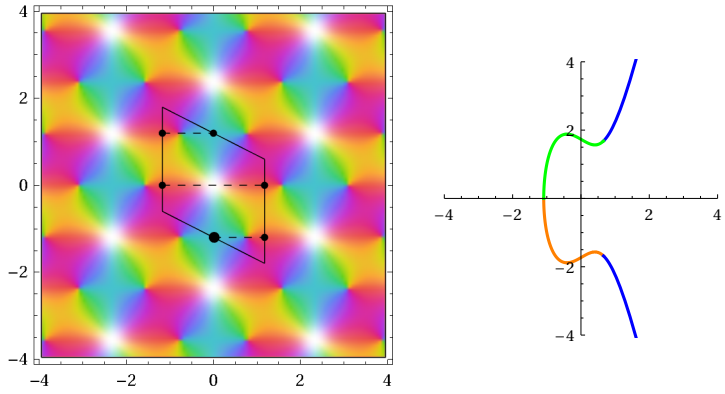
In[ ]:=
```
Row[
 {Show[wpgramPlot [{2, -3}],
   Graphics [{{Black , Dashed , Line[{{-1.175 , impart }, {0, impart }}],
      Line[{{-1.175 , 0}, {1.175 , 0}}], Line[{{0, -impart }, {1.175 , -impart }}]},
     {Black , PointSize [.02],
      Point[{{-1.175 , impart }, {-0, impart }, {-1.175 , 0}, {1.175 , 0}, {0, -impart },
        {1.175 , -impart }}]}]], ImageSize → 200], "    ",
  Show[ParametricPlot [Re[μ[t + impart I, {2, -3}]], {t, -1.175 , 0}, PlotStyle → Orange ,
    PlotRange → 4], ParametricPlot [Re[μ[t, {2, -3}]], {t, -1.175 , 1.175},
    PlotStyle → Blue , PlotRange → 4],
   ParametricPlot [Re[μ[t - impart I, {2, -3}]], {t, 0, 1.175}, PlotStyle → Green ,
    PlotRange → 4], ImageSize → 150]}]
```

» hps {{0. , -1.19722 }, {1.17514 , -0.59861 }}

» corners {{1.17514 , -1.79583 }, {-1.17514 , -0.59861 }, {-1.17514 , 1.79583 }, {1.17514 , 0.59861 }}

» x-axis points {{0.0168172 , -1.19722 }}

Out[ ]=



In this case we can consolidate to one real interval if we can go outside our fundamental parallelogram.

In[ ]:=
```
rpart = 2 * 1.1751`
Row[{Show[wpgramPlot [{2, -3}], Graphics [{Black , Dashed , Line[{{-rpart , 0}, {rpart , 0}}]}]],
  "      ", ParametricPlot [μ[t, {2, -3}], {t, -2.4 , 2.4}, PlotRange → 4, ImageSize → 150]}]
```

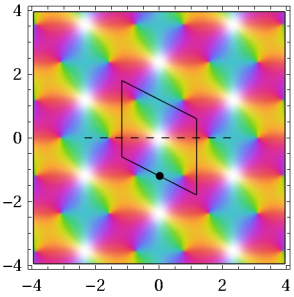
Out[ ]= 2.3502

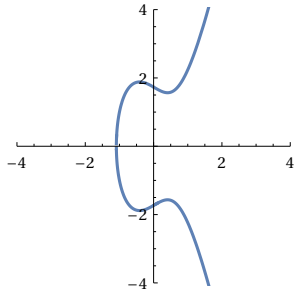
» hps {{0., -1.19722}, {1.17514, -0.59861}}

» corners {{1.17514, -1.79583}, {-1.17514, -0.59861}, {-1.17514, 1.79583}, {1.17514, 0.59861}}

» x-axis points {{0.0168172, -1.19722}}

Out[ ]=





Finally we come to the case where the normal form conic has 2 real components. In this case we are forced to use 2 intervals, one being complex.

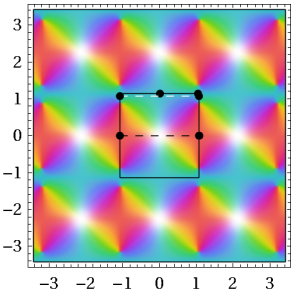
In[ ]:=
```
Row[
 {Show[wpgramPlot [{8, 1}],
   Graphics [{{White , Dashed , Line[{{-1.07255 , 1.0726 }, {1.07255 , 1.0726 }}]},
     {Black , Dashed , Line[{{-1.07255 , 0}, {1.07255 , 0}}]},
     {Black , PointSize [.03],
      Point[{{-1.07255 , 1.0726 }, {1.07255 , 1.0726 }, {-1.07255 , 0}, {1.07255 , 0}}]}]}],
"    ", Show[ParametricPlot [μ[t, {8, 1}], {t, -1.07255 , 1.07255 }, PlotRange → 4,
  ImageSize → 150], ParametricPlot [Re[μ[t + 1.14063 I, {8, 1}]], {t, -1.07255 , 1.07255 },
  PlotRange → 4, ImageSize → 150]]}]
```

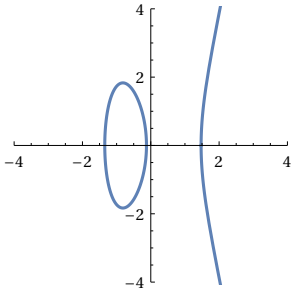
» hps {{1.07255 , 0}, {0., 1.14063 }}

» corners {{1.07255 , 1.14063 }, {1.07255 , -1.14063 }, {-1.07255 , -1.14063 }, {-1.07255 , 1.14063 }}

» x-axis points {{0.014966 , 1.14063 }, {1.04616 , 1.14063 }, {1.08396 , 0}}

Out[ ]=





5. Parameterizing the QSIC

So we have broken the problem down to 3 steps, first use algorithm nsQSIC to get a plane cubic h and rational function $\mho$. The fractional linear transformation Ω is useful for checking our work. Then we transform h to normal form saving the Weierstrass Invariants **wg** and transformation matrix **B2.** We thirdly get our parameter intervals by finding and analyzing the fundamental domain of our invariants. In some cases we may need to do this piecewise with several different intervals. In each case the parametric function is given simply in Mathematica notation

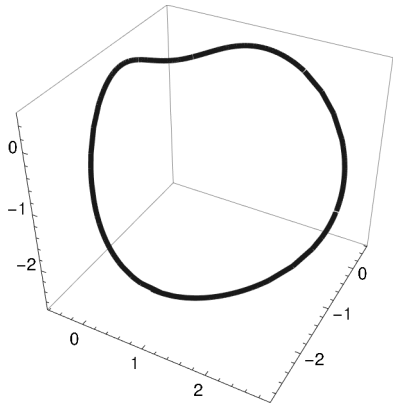
```
℧[fltMD[μ[t + ipart i, wg], Inverse[B2]]]
```

where ipart is the constant imaginary part on this interval, if any.

In our running example Q = {x y-z, $(x-1)^2+(y+1)^2+(z+.9)^2$ -3.6} we have

```
ParametricPlot3D [ʊ0[fltMD[μ[t, wg], Inverse[B2]]], {t, -1.8, 1.8}, PlotRange → Full,
 ImageSize → 200]
```

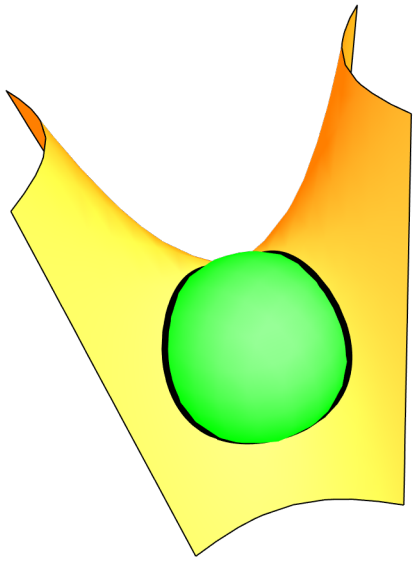
Out[ ]=



Plotting with the surfaces

In[ ]:=
```
Show[ContourPlot3D [{Q[[1]] == 0, Q[[2]] == 0}, {x, -4, 4}, {y, -4, 4}, {z, -4, 4}, Mesh → None,
  ContourStyle → {Orange, Green}],
 ParametricPlot3D [ʊ0[fltMD[μ[t, wg0], Inverse[B2]]], {t, -1.26, 1.26}, PlotRange → Full,
  PlotStyle → Directive[Thickness[.01], Black]], Axes → None, Boxed → False]
```

Out[ ]=



5.1 Some more examples

We summarize the results for two more examples . The next is the intersection of an ellipsoid and a paraboloid that is some - what off center.

In[ ]:=
```
Q2 = {(x - .3)^2/4 + (y + .8)^2/9 + (z - 2)^2 - 1, z - (2 x^2 + 2 y^2)};
```

We first run our nsQSIC routine getting the following

$$h2 = -1.70891 - 4.90155\,x - 0.436806\,x^2 + 2.76916\,x^3 - 3.04504\,y - 5.54903\,x\,y - 1.11613\,x^2\,y + 1.42088\,y^2 + 1.73389\,x\,y^2 + 0.493294\,y^3$$

$$\mho 2[\{x, y\}] = \left\{\frac{-0.44385 - 0.00288317\,x^2 + x\,(-0.386849 + 0.878168\,y) + 0.857298\,y + 0.26742\,y^2}{1.28184 + 2.20131\,x + 1.\,x^2 + 0.568622\,y + 0.519988\,x\,y + 0.124632\,y^2},\ \frac{0.00937192 - 1.2944\,x - 1.23691\,x^2 + 0.287652\,y - 0.16422\,x\,y + 0.00183044\,y^2}{1.28184 + 2.20131\,x + 1.\,x^2 + 0.568622\,y + 0.519988\,x\,y + 0.124632\,y^2},\ \frac{1.40069 + 2.20899\,x + 1.02232\,x^2 + 0.461645\,y + 0.439683\,x\,y + 0.21522\,y^2}{1.28184 + 2.20131\,x + 1.\,x^2 + 0.568622\,y + 0.519988\,x\,y + 0.124632\,y^2}\right\}$$

In[33]:=
```
wg2 = {23.827403750302487` , 20.15211351942233` }
```

Out[33]= {23.8274 , 20.1521 }

In[29]:=
```
B22 // MatrixForm
```

Out[29]//MatrixForm=

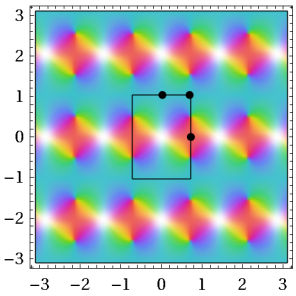
$$\begin{pmatrix} -3.2832 & -1.09932 & -2.61352 \\ -3.67212 & 2.19815 & -0.965365 \\ 1.88915 & 0.418749 & 2.00441 \end{pmatrix}$$

In[34]:=
```
wpgramPlot [wg2 , ep → .2]
```

» hps {{0.769421 , 0}, {0. , 1.10583 }}

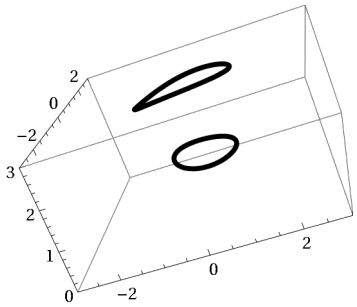
» corners {{0.769421 , 1.10583 }, {0.769421 , -1.10583 }, {-0.769421 , -1.10583 }, {-0.769421 , 1.10583 }}

» x-axis points {{0.0302383 , 1.10583 }, {0.733416 , 1.10583 }, {0.775188 , 0}}



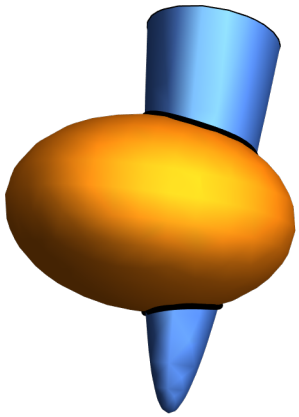
We see that we will have two components so the curve will require two parametric plots, one with complex parameters. The curve itself looks like

```
Show[ParametricPlot3D [℧2[fltMD[μ[t, wg2], Inverse [B22]]], {t, -.77 , .77},
  PlotRange → {{-3 , 3}, {-3 , 3}, {0 , 3}}, PlotStyle → Directive [Thickness [0.015 ], Black ]],
 ParametricPlot3D [℧2[fltMD[μ[t + 1.10583 I , wg2], Inverse [B22]]], {t, -.77 , .77}],
 ImageSize → 200]
```



The curve with the quadric surfaces is

```
Show[ContourPlot3D [{(x - .3)^2 / 4 + (y + .8)^2 / 9 + (z - 2)^2 == 1, z == 2 x^2 + 2 y^2},
  {x, -2, 3}, {y, -4, 4}, {z, 0, 4}, Mesh → None],
 ParametricPlot3D [ʊ2[fltMD[μ[t, wg2], Inverse[B22]]], {t, -.77, .77}, PlotRange → 3,
  PlotStyle → Directive[Thickness[0.015], Black]],
 ParametricPlot3D [ʊ2[fltMD[μ[t + 1.0583 I, wg2], Inverse[B22]]], {t, -.77, .77},
  PlotStyle → Directive[Thickness[0.015], Black]], Axes → False, Boxed → False]
```



This last example is an unbounded curve with 4 affine components consisting with the intersection of a hyperboloid and a hyperbolic ellipsoid.

```
In[ ]:= Q3 = {x^2 + y^2 - z^2 - 9, 2 (x - 1)^2 - (y - .25)^2 - z^2 - .5};
```

A point on Q3 is

```
In[ ]:= q3pt
Out[ ]= {2.46923, -1.70379, 0.}
```

Applying **nsQSIC**

```
In[ ]:= h3
Out[ ]= -0.249469 - 8.22993 x - 2.67636 x^2 + 0.567963 x^3 - 18.5466 y -
        30.8386 x y + 2.22249 x^2 y - 41.7577 y^2 - 21.0204 x y^2 - 24.7859 y^3
```

```
In[ ]:= Ʊ3[{x, y}]
```

$$\text{Out[ ]}= \left\{\frac{-18.1272 + 0.823429\ x^2 + x\,(-31.8909 - 20.0303\ y) - 64.4707\ y - 60.9937\ y^2}{10.4881 - 3.18625\ x + 1.\ x^2 + 31.085\ y + 5.91174\ x\,y + 20.2212\ y^2},\right.$$
$$\frac{34.1146 - 10.7836\ x - 3.34171\ x^2 + 60.4533\ y - 34.7029\ x\,y + 23.2529\ y^2}{10.4881 - 3.18625\ x + 1.\ x^2 + 31.085\ y + 5.91174\ x\,y + 20.2212\ y^2},$$
$$\left.\frac{22.4144 + 22.7967\ x + 1.68673\ x^2 + 13.2424\ y + 27.4305\ x\,y - 24.1011\ y^2}{10.4881 - 3.18625\ x + 1.\ x^2 + 31.085\ y + 5.91174\ x\,y + 20.2212\ y^2}\right\}$$

We now find the Weierstrass Normal Form

```
In[ ]:= wg3
```

```
Out[ ]= {37.0854 , -33.4408 }
```

```
In[ ]:= B23
```

$$\begin{pmatrix} 0.578932 & -2.02309 & -0.722756 \\ 1.79988 & 10.5295 & 9.75449 \\ -0.376334 & -1.02545 & -0.0287017 \end{pmatrix}$$

So far our work is not replicable because of random choices, but our complex plot of WeierstrassP is.

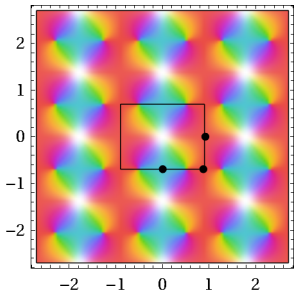
```
In[ ]:= wpgramPlot [wg3 , ep → .3]
```

» hps {{0. , -0.695683 }, {0.901368 , 0.}}

» corners

{{0.901368 , -0.695683 }, {-0.901368 , -0.695683 }, {-0.901368 , 0.695683 }, {0.901368 , 0.695683 }}

» x-axis points {{0.0057994 , -0.695683 }, {0.877145 , -0.695683 }, {0.919792 , 0.}}

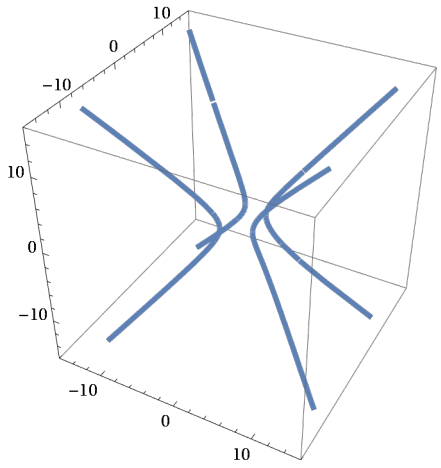


It does indicate there are two components .

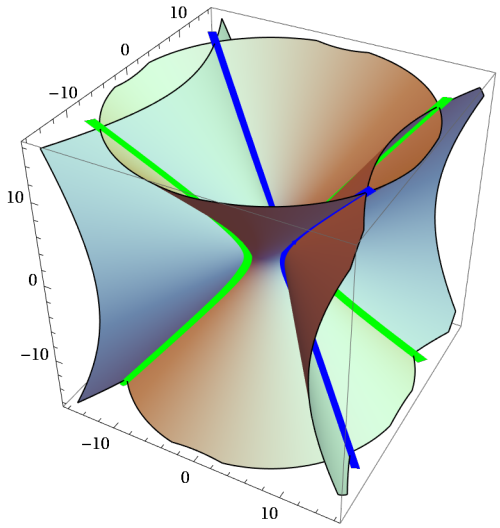
Note that the blue and black give projectively one loop while the green and orange give the other.

Adding our quadratic surfaces to the right hand plot we get

```
Show[ParametricPlot3D [Ʊ3[fltMD[μ[t, wg3], Inverse[B22]]], {t, -.903 , .903},
  PlotRange → 16], ParametricPlot3D [Ʊ3[fltMD[μ[t - .695683 I, wg3], Inverse[B22]]],
  {t, -.903 , .903}, PlotRange → 16], ImageSize → 250]
```



Plotting this with our surfaces gives



6. Conclusion

The QSIC problem is hard because of the many different cases and the complexity of the parameterizations in the genus 1 cases . To actually do examples requires using a computer. Although there are 4 different cases just within the genus 1 cases all of these use the same basic parameterized function

```
υ[fltMD[μ[t + ipart ī, wg], Inverse[B2]]]
```

although in some cases we must use several uses of this with different parameter intervals to get a piecewise parameterization.

This method at present does leave demands on the user to monitor the probabilistic randomized routines and to make the proper choices on parameter values. If the method does not pass its tests it may be that the QSIC is rational, then the various rational cases must be determined. An advantage of our use of **Mathematica** is that all of the difficult computations are done with built-in functions that are fast. The rational cases also can be also handled relatively easily on this platform. Perhaps in the future AI can be taught to do all the user work to get a black box solution to this problem using our methods.

Appendix to Explicit Regular QSIC

Here is the code for **Mathematica** functions, other than those built in to Mathematica, discussed in the body of this paper, mostly without comment here. Also included are subroutines necessary for these functions. If you will be using **Mathematica** to use or experiment with this code the notebook version of my original paper available at the end of the web version at https://-community.wolfram.com/web/bhdayton/home is a better choice to use.

```
In[ ]:= dTol := 1.*^-12
        tDegMD[f_, X_] := Max[Total /@ Keys[CoefficientRules [f, X]]]

In[ ]:= fltMD[p_, A_] := TransformationFunction [A][p]

        FLT3D[F_, A_, X_] := Module[{B, d, g, h, t, n},
          n = Length[X];
          If[Dimensions [A] ≠ {n + 1, n + 1}, Echo[{n + 1, n + 1}, "need A to be of size"]; Abort[]];
          If[MatrixRank [A] ≠ n + 1, Echo["A must be invertible "]; Abort[]];
          B = Inverse [A].Append [X, t];
          Reap[Do[
              d = tDegMD[f, X];
              g = Expand[t^d (f /. Thread [X → X / t])];
              h = Expand [g /. Thread [Append [X, t] → B]];
              Sow[Chop[h /. {t → 1}, 1.*^-12 ]], {f, F}]][[2, 1]]
```

Note : FLT3D requires F to be a list of equations and A to be invertible. This applies the push forward to each polynomial in F separately and returns a list. For a single equation f one should use
FLT3D[{f}, A, X][[1]]

```
In[ ]:= findQpts[Q_, X_, n_] := Re[N[X /. FindInstance [Q == 0, X, Reals , n]]]

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

In[ ]:= nsQSIC[Q_, p_, {x_, y_, z_}] := Module[{dTol , p0, A, F, h, L, M, R, S, Ω, ʊ},
          dTol = 1.*^-12 ;
          p0 = Normalize [Append [p, 1]];
          A = Reverse[Orthogonalize [Prepend [RandomReal [{-1, 1}, {3, 4}], p0]]];
          F = FLT3D[Q, A, {x, y, z}];
          L = formMD [F[[1]], 1, {x, y, z}];
          M = formMD [F[[2]], 1, {x, y, z}];
          R = formMD [F[[1]], 2, {x, y, z}];
          S = formMD [F[[2]], 2, {x, y, z}];
          h = Expand [L * S - R * M] /. {z → 1};
          Ω = Take[A, 3];
          ʊ = Take[Inverse [A].Join[#, {1, (-R / L) /. Thread [{x, y, z} → Append [#, 1]]}], 3] /
              Last[Inverse [A].Join[#, {1, (-R / L) /. Thread [{x, y, z} → Append [#, 1]]}]] &;
          {h, Ω, ʊ}]
```

```
In[ ]:= allInflectionPointsC3 [f_, x_, y_] := Module[{d, H, fh, z, l, ips, ips2, sol},
          d = tDegMD[f, {x, y}];
          fh = Expand[z^d (f /. Thread[{x → x / z, y → y / z}])];
          H = Det[D[fh, {{x, y, z}, 2}]];
          l = RandomReal[{-10, 10}, 3].{x, y, z};
          sol = NSolve[{fh, H, l - 1}, {x, y, z}];
          If[Length[sol] == 0, Return[{}]];
          ips = {x, y, z} /. sol;
          ips2 =
           Chop[If[Abs[#1〚3〛] > 1.*^-5, Take[#1, 2] / #1〚3〛, Append[Take[#1, 2], 0]], 1.*^-10] & /@
            ips;
          DeleteDuplicatesBy [ips2, Abs[#] < 1.*^-6]
         ];

In[ ]:= cTransform2D [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};
```

```
In[ ]:= weierstrassNormalForm [f_, ip_, x_, y_] :=
  Module[{Rnm, ff, ipp, B1, B2, B3, B4, B5, B, f1, f2, f3, f4, k, cy2, wg},
   Rnm = RandomReal [{-3, 3}, {3, 3}];
   ipp = fltMD[ip, Rnm];
   ff = FLT3D[{f}, Rnm, {x, y}][[1]];
   B1 = cTransform2D [ff, ipp, x, y];
   f1 = Chop[FLT3D[{ff}, B1, {x, y}][[1]], 1.*^-9 ];
   cy2 = Coefficient [f1, y^2];
   If[Length[cy2] > 0, Print["Please check inflection point or smoothness "]; Abort[]];
   k = Expand[Coefficient [f1, y] / cy2 / 2 + y];
   B2 = {{1, 0, 0}, {Coefficient [k, x], 1, k[[1]]}, {0, 0, 1}};
   f2 = FLT3D[{f1}, B2, {x, y}][[1]];
   B3 = {{-(Coefficient [f2, x^3] / Coefficient [f2, y^2])^(1 / 3), 0, 0}, {0, 1, 0}, {0, 0, 1}};
   f3 = FLT3D[{f2}, B3, {x, y}][[1]];
   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 = Expand[4 FLT3D[{f3}, B5.B4, {x, y}][[1]]];
   Echo[f4, "Normal Form"];
   wg = {-Coefficient [f4, x], -f4 /. Thread[{x, y} → 0]};
   B = B5.B4.B3.B2.B1.Rnm;
   {wg, B}];

In[ ]:= μ[t_, {g2_, g3_}] := Re[{WeierstrassP [t, {g2, g3}], WeierstrassPPrime [t, {g2, g3}]}]

In[ ]:= Options[wpgramPlot ] = {ep → 0.103};
  wpgramPlot [{g2_, g3_}, OptionsPattern []] :=
   Module[{hps, hp1, hp2, mn, x, y, zero, z, olz, opts, one, onelz, onepts},
    hps = WeierstrassHalfPeriods [1. {g2, g3}];
    Echo[ReIm[hps], "hps"];
    hp1 = ReIm[hps[[1]]];
    hp2 = ReIm[hps[[2]]];
    mn = 3 Max[Norm[hp1], Norm[hp2]];
    Echo[{hp1 + hp2, hp1 - hp2, -hp1 - hp2, -hp1 + hp2}, "corners "];
    zero = NSolveValues [y^2 == 4 x^3 - g2 x - g3 && y == OptionValue [ep], {x, y}, Reals];
    olz = Length[zero];
    opts = Table[ReIm[InverseWeierstrassP [zero[[j]], {g2, g3}]], {j, olz}];
    Echo[opts, "x-axis points "];
    Show[ComplexPlot [WeierstrassP [z, {g2, g3}], {z, -mn - mn I, mn + mn I}, ImageSize → 150],
     Graphics [{{Black, Line[{hp1 + hp2, hp1 - hp2, -hp1 - hp2, -hp1 + hp2, hp1 + hp2}],
         PointSize [.03], Point[opts]}}]]]
```